Advances and Frontiers of LLM-based Issue Resolution in Software Engineering: A Comprehensive Survey

Caihua Li[1], Lianghong Guo[1], Yanlin Wang[1]*, Daya Guo[1], Wei Tao[2]*, Zhenyu Shan[3], Mingwei Liu[1], Jiachi Chen[4], Haoyu Song[5], Duyu Tang[5], Hongyu Zhang[6], Zibin Zheng[1]

[1]Sun Yat-sen University, [2]Independent Researcher, [3]Hangzhou Normal University, [4]Zhejiang University, [5]Huawei Technologies Co, Ltd, [6]Chongqing University

{lich535, guolh8, guody5}@mail2.sysu.edu.cn, {wangylin36, zhzibin}@mail.sysu.edu.cn, wtao@ieee.org, 20100119@hznu.edu.cn, chenjiachi317@gmail.com, {songhaoyu1, tangduyu}@huawei.com, hyzhang@cqu.edu.cn

Abstract

Issue resolution, a complex Software Engineering (SWE) task integral to real-world development, has emerged as a compelling challenge for artificial intelligence. The establishment of benchmarks like SWE-bench revealed this task as profoundly difficult for large language models, thereby significantly accelerating the evolution of autonomous coding agents. This paper presents a systematic survey of this emerging domain. We begin by examining data construction pipelines, covering automated collection and synthesis approaches. We then provide a comprehensive analysis of methodologies, spanning training-free frameworks with their modular components to training-based techniques, including supervised fine-tuning and reinforcement learning. Subsequently, we discuss critical analyses of data quality and agent behavior, alongside practical applications. Finally, we identify key challenges and outline promising directions for future research. An open-source repository is maintained at https://github.com/DeepSoftwareAnalytics/Awesome-Issue-Resolution to serve as a dynamic resource in this field.

1 Introduction

The vision of constructing a true AI software engineer has long been an appealing prospect in computer science. In pursuit of this, researchers initially relied on function-level code generation benchmarks such as HumanEval (Chen et al., 2021). Driven by the remarkable success of Large Language Models (LLMs), the prospect of automating software engineering to function akin to a human developer has become increasingly attainable. However, they often struggle to handle the dynamic interactions with development environments and human collaboration in realistic scenarios. To address this limitation and align evaluation with authentic software development scenarios, Jimenez et al. (2024) introduced SWE-bench and defined the task of issue resolution. This task requires an automatic approach to help LLMs navigate complex, multi-file repositories to resolve issues in GitHub (see Figure 1 and Section 2). By revealing the difficulty of repository-level engineering, SWE-bench catalyzed a research frontier focused on navigating and modifying complex environments (Pan et al., 2025a). It marks a departure from initial software generation as explored in ChatDev (Qian et al., 2024) and MetaGPT (Hong et al., 2024), to the subsequent stages of software maintenance and evolution.

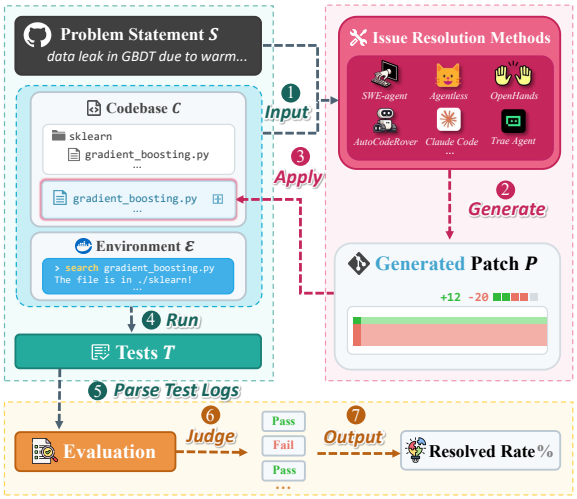


Figure 1: Issue Resolution

Despite the surge of research in this new frontier, the literature remains fragmented. Current surveys primarily focus on code generation, failing to address the far more complex challenge of issue resolution. This paper aims to bridge this gap by providing the first in-depth survey of this domain.

We conducted a comprehensive survey of publicly available literature, including 135 papers and online resources relevant to issue resolution. We established a tailored classification framework to provide a structured perspective on this rapidly

*Corresponding authors.

evolving domain. Consequently, our contributions can be summarized as follows: We present a **systematic survey** on issue resolution, organized by a **structured taxonomy** covering Data, Methods, and Analysis. Furthermore, we identify **key challenges and future directions**, and provide a **open-source repository** to support the community.

2 Task formulation

Issue resolution requires LLMs to synthesize a valid code change (also called a patch) $\mathcal{P}$ to resolve the issue, interacting with a codebase (as shown in Figure 1). Formally, an instance of the task can be expressed as $\mathcal{I} = (\mathcal{D}, \mathcal{C}, \mathcal{T})$, comprising an issue description $\mathcal{D}$, the codebase $\mathcal{C}$, and a corresponding test $\mathcal{T}$. The observable parts of the instance, including $\mathcal{D}$ and $\mathcal{C}$, are available during the resolving process, which contains supplementary information on the corresponding environment $\mathcal{E}$ that can be explored. So a method $\mathcal{M}$ is expected to achieve:

$$\mathcal{P} = \mathcal{M}(\mathcal{D}, \mathcal{C}, \mathcal{E}) \tag{1}$$

After the patch $\mathcal{P}$ is applied to $\mathcal{C}$, the evaluation is conducted through running tests on the modified codebase $\mathcal{C}' = \text{Apply}(\mathcal{C}, \mathcal{P})$. The resolution outcome $r \in \{0, 1\}$ is then determined by the execution of $\mathcal{T}$, denoted as $\text{Exec}(\mathcal{C}', \mathcal{T})$. On a dataset including n instances($\mathbb{I} = \{\mathcal{I}_i\}_{i=1}^{N}$), overall performance metric is:

$$\text{Resolved Rate} = \frac{1}{|\mathbb{I}|} \sum_{i=1}^{|\mathbb{I}|} r_i \tag{2}$$

3 Data

Data is fundamental to the issue-resolution task, serving as both an evaluation benchmark and a training resource. Thus, datasets are classified into evaluation datasets (§ 3.1.1) and training datasets (§ 3.1.2). Construction approaches are divided into two types: data collection (§ 3.2.1) from real-world online sources, and data synthesis (§ 3.2.2) achieved by rewriting real-world data or rule-based generation. Statistics for all datasets are provided in § A.3.

3.1 Datasets

3.1.1 Evaluation datasets

SWE-bench (Jimenez et al., 2024) established the datasets by collecting issue–Pull Request (PR) pairs from popular Python repositories, pairing each issue with a full repository snapshot for resolution. However, invalid tests and underspecified descriptions in the original dataset make many instances unsuitable for evaluation. To address this issue and ensure data quality, SWE-bench Verified (OpenAI, 2024) was introduced, offering a subset of manually validated samples as a trusted benchmark.

While most evaluation datasets target Python, researchers extend these tasks to ten different programming languages to broaden the linguistic scope, as seen in SWE-bench Multilingual and Multi-SWE-bench, et al. (Zan et al., 2024, 2025; Rashid et al., 2025; Guo et al., 2025b; Yang et al., 2025c; Mhatre et al., 2025). To address the limitation of relying solely on textual data, researchers have focused on aggregating multimodal information, primarily derived from images such as UI screenshots and diagrams (Yang et al., 2024b; Zhang et al., 2025e; Guo et al., 2025b). For instance, Yang et al. (2024b) integrates these visual contexts and introduces a novel visual tester to validate the correctness of visual modifications.

To bridge the significant gap between initial evaluation datasets and realistic software development scenarios, researchers introduced datasets (Miserendino et al., 2025; Deng et al., 2025; Tarasova et al., 2025) that incorporate enterprise-level complexity and diverse domains. Recent advancements have also shifted towards refining metrics for issue resolution, such as efficiency and safety (He et al., 2025a; Ma et al., 2025a; Xu et al., 2025a).

3.1.2 Training datasets

Textual data The initial training data typically consists of raw task instances in the form of static issue-PR, as exemplified by the training set in SWE-bench (Jimenez et al., 2024), to equip models with fundamental capabilities to resolve issues before they engage with interactive environments.

Training Environment Environment datasets aim to address the limitations of static text by constructing interactive code environments that provide LLMs with execution feedback. In early initiatives, researchers attempted to equip each data instance with a corresponding Conda or Docker environment, enabling LLMs to incorporate code execution results as feedback during the training process, as seen in benchmarks like Multi-SWE-RL (Zan et al., 2025). Nevertheless, these datasets often

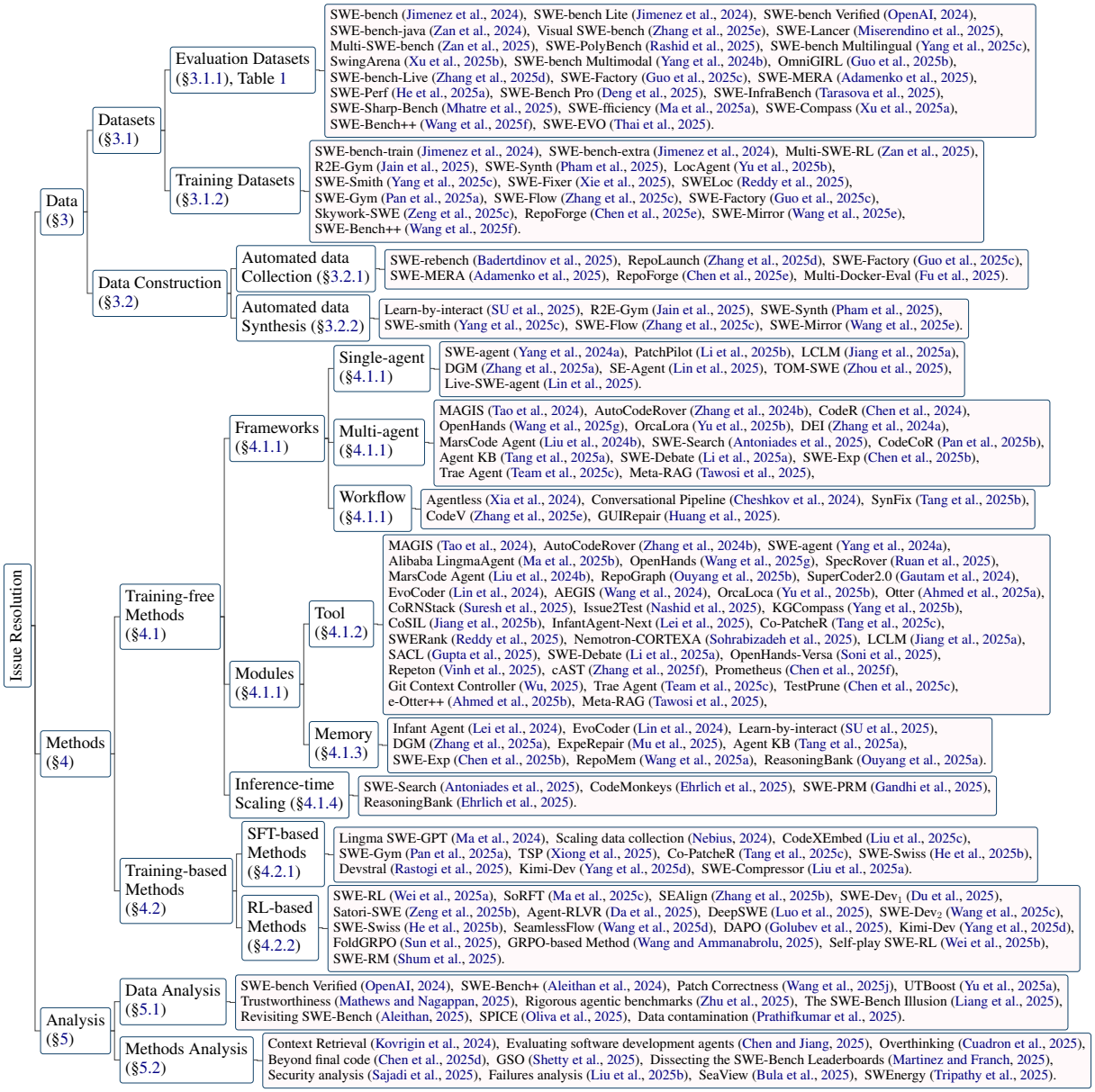


Figure 2: Overall perspective on data, methods, and analysis for SWE tasks, featuring corresponding papers.

overlook the specific interface design required for effective LLM-environment interaction. To address this, Jain et al. (2025) introduced a more interactive Gym environment (R2E-Gym) that utilizes LLM-synthesized test cases to verify environment usability, thereby synthesizing large-scale environment data. Similarly, Pan et al. (2025a) constructed Gym environments based on real-world GitHub issues.

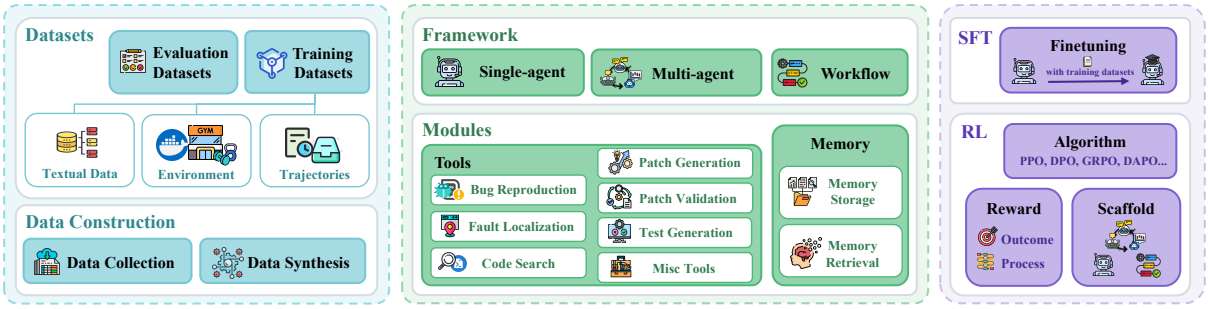
Trajectories Trajectory datasets capture the procedural interplay between language models and execution environments through tool invocations and feedback loops (Nebius, 2024; Jain et al., 2025; Pan et al., 2025a; Pham et al., 2025; Xie et al., 2025; Guo et al., 2025c)(See § A.3). To obtain high-quality trajectories, researchers typically employ inference-time scaling strategies to generate many candidates, then apply verifiers to filter and select the best (Nebius, 2024; Yang et al., 2025c).

3.2 Data construction

The construction of training data for software agents involves transitioning from manual to automated engineering pipelines. Detailed background information is provided in § A.2.

3.2.1 Automated data collection

Static datasets often suffer from rigidity, high maintenance costs, and limited scale, which impede the effective training of robust models. In response, the field is evolving towards scalable, automated data collection methods. Those automated pipelines usually leverage LLMs to explore repository configurations, identify files related to environment setup, and generate corresponding dependency installation commands to build Docker images for in-



(a) Data (b) Training-free Methods (c) Training-based Methods

Figure 3: A classification overview of data, training-free methods, and training methods for solving SWE tasks.

dividual issues. Subsequently, they employ existing testing frameworks and predefined log parsers to analyze test execution results, encompassing both workflow-based and agent-based paradigms, such as SWE-rebench and RepoLaunch (Badertdinov et al., 2025; Zhang et al., 2025d). Notably, SWE-Factory employs a memory-enhanced framework for environment setup and verification, while utilizing an exit-code-based automatic grading method to design parsers capable of automatically interpreting execution results across diverse testing frameworks in different programming languages (Guo et al., 2025c). More recently, RepoForge (Chen et al., 2025e) enhanced the pipeline's automation by incorporating a automatic verification mechanism based on SPICE (Oliva et al., 2025) following data construction, effectively replacing the need for human expert validation.

3.2.2 Automated data synthesis

To overcome the challenges of limited real-world data availability and expensive manual verification, researchers have increasingly adopted automated data synthesis approaches. For instance, SWE-Synth (Pham et al., 2025) achieves this by rewriting target code and generating corresponding tests. Drawing on test-driven development, SWE-Flow (Zhang et al., 2025c) employs a runtime dependency graph to derive incremental code and requirement documents directly from unit tests. More recently, addressing the need to minimize the storage footprint while scaling, SWE-Smith (Yang et al., 2025c) expands the dataset by paraphrasing descriptions and injecting bugs, leveraging a shared environment strategy for data derived from the same source. Similarly, SWE-Mirror (Wang et al., 2025e) transplants real-world issues into target repositories, generating verifiable tasks to further scale the data using shared environmental configurations.

4 Methods

4.1 Training-free method

To overcome constraints such as fixed context windows and static knowledge, training-free methods utilizes external components and sophisticated prompting. As shown in Figure 3, we classify these methods into three categories based on the underlying framework: (1) *Frameworks*, encompassing high-level architectures like single-agent, multi-agent, and fixed-workflow designs; (2) *Modules*, providing plug-and-play augmentations such as *Tools* for repository interaction and *Memory* for experience accumulation; and (3) *Inference-time Scaling*(or test-time scaling), employing search or parallelization strategies to enhance success rates without modifying model parameters.

4.1.1 Frameworks

To handle the multi-stage execution required for issue resolution, current research structures LLM activities into either dynamic agent-based or rigid workflow-based frameworks.

Single-agent Analogous to software engineers who write code and invoke diverse tools to resolve issues, single-agent frameworks were initially constructed to execute tasks via tool-based interaction paradigms. SWE-agent (Yang et al., 2024a) pioneers the agent-computer interface, which enables autonomous file navigation, code editing, and test execution, bridging natural language understanding with repository-level operations.

However, granting full autonomy for every decision often leads to redundant action sequences due to reasoning imprecision, resulting in prohibitive operational costs. To address it, Li et al. (2025b) reduces overhead by either constraining specific

phases into rigid workflows.

To further enhance generalization across diverse issue types, self-evolutionary frameworks have emerged to autonomously refine agent capabilities. For instance, Darwin Gödel Machine employs an evolutionary process starting from a minimal baseline, where the LLM generates, scores, and selects optimal candidate agent implementations over successive iterations to evolve its structure (Zhang et al., 2025a; Xia et al., 2025; Lin et al., 2025).

Multi-agent Introduced concurrently with single-agent systems, multi-agent frameworks focus on collaboration and task allocation, often performed in the form of human software development team (Tao et al., 2024; Zhang et al., 2024b; Liu et al., 2024b; Antoniades et al., 2025; Yu et al., 2025b; Team et al., 2025c). For instance, MAGIS (Tao et al., 2024) firstly implements this by assigning agents to roles such as Manager, Developer, and QA Engineer. This setup allows agents to role-play and autonomously convene meetings for effective communication.

However, those works primarily used text-based contexts for information exchange but failed to explicitly model how agents collaborate. To address this limitation, CodeR (Chen et al., 2024) tackled the challenge of unreliable agent collaboration by introducing task graphs, a formal data structure to convert a high-level plan into a parsable, directed graph to ensure precise execution. Also based on graphs, SWE-Debate (Li et al., 2025a) organizes a three-round debate among specialized agents, each embodying distinct reasoning perspectives along the fault propagation trace in the code dependency graph to forge a more concrete solution.

With the proliferation of diverse agent frameworks, recent research has shifted towards constructing unified platforms capable of orchestrating collaboration among agents with varying architectures. (Wang et al., 2025g; Zhang et al., 2024a)For example, Zhang et al. (2024a). proposed DEIBase, a framework designed to leverage LLMs to score and rank solutions generated by multiple agent frameworks. By integrating multiple agents in this manner, DEIBase achieves superior performance compared to single-agent approaches.

Workflow Fixed-workflow architectures improve stability by enforcing predefined steps instead of allowing open-ended exploration. Some systems adopt a linear pipeline—localization, repair, and validation—to ensure efficiency and reproducibility (Xia et al., 2024). For visual tasks, researchers integrate Vision Language Models to directly parse UI screenshots into code (Zhang et al., 2025e; Huang et al., 2025). Additionally, to handle complex codebases, recent work utilizes dependency graphs to guide precise, repository-wide modifications instead of relying on random search (Tang et al., 2025b).

4.1.2 Tool modules

In training-free frameworks, LLMs rely on specialized tools to augment reasoning without fine-tuning. These tools are organized by the standard repair pipeline, progressing from bug reproduction, fault localization, and code search to patch generation, validation, and test generation.

Bug reproduction tools. These tools automate the critical first step of debugging by generating executable scripts that trigger reported defects. Implementations typically leverage historical interaction data to adapt to repository-specific conventions (Lin et al., 2024), or employ Finite State Machines to govern behavior via multi-dimensional feedback, as in AEGIS (Wang et al., 2024).

Fault localization tools. Once a bug is reproduced, these tools pinpoint suspicious code regions to narrow the search space. Common approaches include method-level Spectrum-Based Fault Localization (SBFL) (Zhang et al., 2024b) and graph-based methods that construct code dependency graphs to trace fault propagation (Li et al., 2025a).

Code search tools. These tools retrieve relevant dependency context after localization. Strategies range from interactive retrieval using BM25 or AST-based APIs (Tao et al., 2024; Yang et al., 2024a), to graph-based global understanding via Knowledge Graphs and Language Server Protocols (Ma et al., 2025b; Liu et al., 2024b), and dynamic managers that balance exploration breadth and depth (Yu et al., 2025b; Jiang et al., 2025b).

Patch generation tools. These tools guide LLM output quality through structured methodologies. Key techniques include augmenting input context via specification inference (Ruan et al., 2025), employing robust editing formats such as AutoDiff to bypass line-numbering failures (Yang et al., 2024a; Liu et al., 2024b), and ensemble selection mechanisms that filter candidates via regression testing (Team et al., 2025c).

Patch validation tools. These tools confirm correctness and prevent regressions through external verification. Standard approaches include dynamic execution orchestration using sandboxed environments (Zhang et al., 2024b; Liu et al., 2024b), and static analysis mechanisms leveraging QA agents or Language Server Protocols for immediate diagnostic feedback (Tao et al., 2024; Liu et al., 2024b).

Test generation tools. These tools generate reproduction test cases to validate intent and guide resolution. Systems typically employ feedback-driven iterative mechanisms that utilize error classification to synthesize failing tests reproducing reported defects, as in Otter (Ahmed et al., 2025a) and Issue2Test (Nashid et al., 2025).

Other extensions. Recent extensions focus on equipping agents with versatile tools to handle broader scenarios, including multimodal challenges. Strategies involve multimodal browsing and unified information access, which standardizes heterogeneous data into Markdown for seamless processing (Soni et al., 2025; Lei et al., 2025).

4.1.3 Memory modules

Memory integration empowers agents to transcend isolated problem-solving by accumulating historical context to guide future actions. Initial architectures focused on establishing hierarchical storage structures, such as segregating general knowledge from repository-specific details to mitigate rigidity (Lin et al., 2024), or archiving populations of agent variants to support open-ended evolution (Zhang et al., 2025a). To overcome the limitations of static prompting, researchers have subsequently incorporated dual-process cognitive architectures that synergize episodic records of concrete repairs with semantic layers of abstract insight, enabling dynamic retrieval based on current context (Mu et al., 2025). Current frontiers prioritize distilling transferable reasoning strategies, effectively shifting the paradigm from storing raw data to abstracting high-level policies from both successful and failed trajectories. This evolution allows agents to leverage multi-faceted experience banks to guide strategic search frameworks like MCTS (Chen et al., 2025b) and to prevent error repetition through generalized, rule-based learning (Ouyang et al., 2025a).

4.1.4 Inference-time scaling

While specialized tools and memory systems enhance specific agent capabilities, relying solely on linear execution paths often limits the exploration of complex solution spaces. To address this, inference-time scaling has emerged as a critical paradigm to expand the search breadth and depth during problem-solving. To overcome the rigidity of sequential workflows, recent research focuses on enabling non-linear exploration via Monte Carlo Tree Search (MCTS), which facilitates flexible backtracking and qualitative feedback loops to prevent agents from stagnating in repetitive cycles (Antoniades et al., 2025). Complementing this algorithmic shift, strategies for scaling computational resources deploy multiple independent state machines in parallel, maximizing solution coverage while amortizing the high costs of context identification without the need for model retraining (Ehrlich et al., 2025). Furthermore, advanced frameworks are now integrating memory-driven scaling, utilizing time-travel mechanisms to generate diverse experiences that serve not just to resolve the immediate task, but to distill generalizable reasoning strategies for long-term agent evolution (Ouyang et al., 2025a).

4.2 Training-based method

Training-based methods encompass Supervised Fine-Tuning-based (SFT-based) methods and Reinforcement Learning-based (RL) methods, utilizing resources from Section 3.1.2 to enhance the fundamental programming capabilities of LLMs.

4.2.1 SFT-based method

Supervised Fine-Tuning (SFT) serves as the primary mechanism for grounding base models in software engineering protocols. Recent efforts to achieve robust domain adaptation focus on three key dimensions: (1) *Data Scaling*. Strategies increasingly prioritize the expansion of data scale and diversity via synthesized corpora. Frameworks employ iterative generation and filtering pipelines—augmented by automatic test generation or mid-training on billions of GitHub tokens—to comprehensively cover diverse repair scenarios (Ma et al., 2025b; Wang et al., 2025c; Yang et al., 2025d). (2) *Curriculum Learning*. Beyond raw volume, research emphasizes multi-stage curriculum learning. Models are refined through phased training sequences that progress from broad trajectory ingestion to strictly filtered, high-quality

subsets or specialized tasks like localization and testing (He et al., 2025b; Rastogi et al., 2025; Liu et al., 2025a). (3) *Rejection Sampling*. To bridge the gap toward reinforcement learning, current methods employ rejection sampling pipelines. By fine-tuning exclusively on successful trajectories, these methods establish a strong baseline policy while simultaneously training verifiers to re-rank solutions at inference time (Pan et al., 2025a). See Table 3 for detailed statistics on these SFT-trained models.

4.2.2 RL-based method

Reinforcement learning optimizes issue resolution strategies through iterative interaction. This process hinges on the synergy of three core components: the algorithm for policy updates, reward design for guiding exploration, and the scaffold for managing environment rollouts. A statistical overview of recent models and their implementations across these dimensions is presented in Table 4 and Table 5.

Algorithm. The optimization of agent behaviors leverages various policy gradient and alignment strategies to stabilize learning. We discuss three common algorithmic choices as follows. (1) *Group Relative Policy Optimization (GRPO)*. A dominant approach employs GRPO, which enhances reasoning capabilities by normalizing advantages across group sampling without the heavy computational burden of a critic model (Wei et al., 2025a; Sun et al., 2025). (2) *Proximal Policy Optimization (PPO)*. Beyond group-based methods, some approaches utilize PPO for stable updates focused on subtasks (Ma et al., 2025c). (3) *Direct Preference Optimization (DPO)*. Other work integrates MCTS with DPO to align complex, multi-step decision processes with high-quality preferred trajectories (Zhang et al., 2025b).

Reward design. Effective feedback mechanisms typically incorporate both sparse outcome-based rewards and dense process-oriented signals. Most systems employ outcome reward models to provide terminal signals by utilizing strict metrics like patch similarity or detailed subtask verification ranging from localization to editing, thereby rigorously aligning outputs with ground truth (Wei et al., 2025a; Ma et al., 2025c). To mitigate the challenge of signal sparsity in long-horizon reasoning, researchers increasingly adopt process reward models and potential-based shaping techniques; these mechanisms provide dense, step-by-step feedback or token-level incentives, offering reward signals for intermediate behaviors such as context management and trajectory search throughout the reasoning process (Zeng et al., 2025b; Da et al., 2025; Sun et al., 2025).

Scaffold. In the context of RL, a scaffold serves as the inference framework for rollouts. As statistics in Table 4 indicate, OpenHands is the most prevalent scaffold, followed by workflow-based methods (notably Agentless and two-stage workflows). Environment-native frameworks like R2E-Gym and SWE-Gym are also frequently adopted due to their seamless alignment with training data.

5 Analysis

Beyond developing new methodologies, a complementary line of research focuses on empirical analysis of existing data and methods, which provide critical insights into the limitations of current approaches and offer valuable perspectives for future research directions.

5.1 Data analysis

Recent scrutiny has exposed hidden benchmark defects, revealing that agent success rates are frequently inflated by solution leakage, ambiguous issue descriptions, and weak test suites that fail to catch incorrect patches (OpenAI, 2024; Aleithan et al., 2024; Wang et al., 2025j). Recognizing that manual cleanup is too costly and inconsistent for large-scale datasets, the field is shifting toward automating validation workflows, utilizing model-based consensus mechanisms to reliably distinguish valid fixes from false positives without human intervention (Oliva et al., 2025).

5.2 Methods analysis

Research has shifted beyond measuring simple success rates to investigate the behavioral pathology of agents. A primary focus involves diagnosing internal reasoning failures, specifically examining the tendency of models to prioritize prolonged internal deliberation over necessary environmental interaction—a maladaptive pattern that leads to analysis paralysis and rogue actions (Cuadron et al., 2025). Complementing this, efforts to manage the complexity of massive, 128k-token interaction logs have led to streamlining trajectory inspection, utilizing novel visual interfaces to transform cryptic

output streams into navigable workflows for rapid error analysis (Bula et al., 2025).

6 Application

The industrial deployment of software engineering AI has progressed from localized IDE assistance to fully autonomous systems capable of handling complex enterprise workflows. Due to space constraints, we provide a detailed discussion on these application scenarios in § A.5.

7 Challenges and Opportunities

High computational overhead. The scalability of SWE agents is bottlenecked by the high costs of sandboxed environments and long-context inference. Optimization strategies are required to streamline these resource-intensive loops without sacrificing performance.

Opacity in resource consumption. Benchmarks often overlook efficiency, masking the high costs of techniques like inference-time scaling. Standardized reporting of latency and token usage is crucial for guiding the development of cost-effective agents.

Limited visually-grounded reasoning. Reliance on text proxies for UI interpretation limits effectiveness. Future research can adopt intrinsic multimodal solutions, such as code-centric MLLMs, to better bridge the gap between visual rendering and underlying code logic.

Safety risks in autonomous resolution. High autonomy carries risks of destructive actions, such as accidental code deletion. Future systems should integrate safeguards, such as Git-based version control, to ensure autonomous modifications remain secure and reversible.

Lack of fine-grained reward signals. Reinforcement learning is hindered by sparse, binary feedback. Integrating fine-grained signals from compiler diagnostics and execution traces is necessary to guide models through complex reasoning steps.

Data leakage and contamination. As benchmarks approach saturation, evaluation validity is compromised by data leakage. Future frameworks must strictly enforce decontamination protocols to ensure fairness and reliability.

Lack of universality across SE domains. While current issue resolution tasks mirror development workflows, they represent only a fraction of the full Software Development Life Cycle (SDLC). Future research should broaden the scope of issue resolution tasks to develop more versatile automated software generation methods.

8 Conclusion

In this paper, we conduct a systematic survey of Issue Resolution. We offer a comprehensive review of this domain. Specifically, we summarize the rapidly growing ecosystem of data, methods, and analysis through a meticulous taxonomy that offers novel perspectives. Moreover, we delve into the current frontiers, delineate the key challenges and future directions, and engage in a discussion about open problems and critical analyses. To the best of our knowledge, this paper is the first systematic survey dedicated specifically to this domain. We hope that this survey serves as an introduction for researchers and fosters future research in this area.

9 Limitations

As the first dedicated survey on Issue Resolution, we prioritize high-level summaries over exhaustive details due to space constraints. Our search methodology relied on citation tracking (e.g., of SWE-bench) and snowballing; while thorough, this may overlook niche or nascent works. To address this rapid evolution, we commit to continuously updating our open-source repository.

References

Pavel Adamenko, Mikhail Ivanov, Aidar Valeev, Rodion Levichev, Pavel Zadorozhny, Ivan Lopatin, Dmitry Babayev, Alena Fenogenova, and Valentin Malykh. 2025. Swe-mera: A dynamic benchmark for agenticly evaluating large language models on software engineering tasks. *Preprint*, arXiv:2507.11059.

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, and 1 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*.

Toufique Ahmed, Jatin Ganhotra, Rangeet Pan, Avraham Shinnar, Saurabh Sinha, and Martin Hirzel. 2025a. Otter: Generating tests from issues to validate swe patches. *Preprint*, arXiv:2502.05368.

Toufique Ahmed, Jatin Ganhotra, Avraham Shinnar, and Martin Hirzel. 2025b. Execution-feedback

driven test generation from swe issues. *Preprint*, arXiv:2508.06365.

Toufique Ahmed, Martin Hirzel, Rangeet Pan, Avraham Shinnar, and Saurabh Sinha. 2024. Tdd-bench verified: Can llms generate tests for issues before they get resolved? *Preprint*, arXiv:2412.02883.

Reem Aleithan. 2025. Revisiting swe-bench: On the importance of data quality for llm-based code models. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 235–236.

Reem Aleithan, Haoran Xue, Mohammad Mahdi Mohajer, Elijah Nnorom, Gias Uddin, and Song Wang. 2024. Swe-bench+: Enhanced coding benchmark for llms. *Preprint*, arXiv:2410.06992.

Antonis Antoniades, Albert Örwall, Kexun Zhang, Yuxi Xie, Anirudh Goyal, and William Wang. 2025. Swe-search: Enhancing software agents with monte carlo tree search and iterative refinement. *Preprint*, arXiv:2410.20285.

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. *Preprint*, arXiv:2505.20411.

Timothy Bula, Saurabh Pujar, Luca Buratti, Mihaela Bornea, and Avirup Sil. 2025. Seaview: Software engineering agent visual interface for enhanced workflow. *Preprint*, arXiv:2504.08696.

Aili Chen, Aonian Li, Bangwei Gong, Binyang Jiang, Bo Fei, Bo Yang, Boji Shan, Changqing Yu, Chao Wang, Cheng Zhu, and 1 others. 2025a. Minimax-m1: Scaling test-time compute efficiently with lightning attention. *arXiv preprint arXiv:2506.13585*.

Dong Chen, Shaoxin Lin, Muhan Zeng, Daoguang Zan, Jian-Gang Wang, Anton Cheshkov, Jun Sun, Hao Yu, Guoliang Dong, Artem Aliev, Jie Wang, Xiao Cheng, Guangtai Liang, Yuchi Ma, Pan Bian, Tao Xie, and Qianxiang Wang. 2024. Coder: Issue resolving with multi-agent and task graphs. *Preprint*, arXiv:2406.01304.

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, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Silin Chen, Shaoxin Lin, Xiaodong Gu, Yuling Shi, Heng Lian, Longfei Yun, Dong Chen, Weiguo Sun, Lin Cao, and Qianxiang Wang. 2025b. Swe-exp: Experience-driven software issue resolution. *Preprint*, arXiv:2507.23361.

Yang Chen, Toufique Ahmed, Reyhaneh Jabbarvand, and Martin Hirzel. 2025c. When old meets new: Evaluating the impact of regression tests on swe issue resolution. *Preprint*, arXiv:2510.18270.

Zhi Chen and Lingxiao Jiang. 2025. Evaluating software development agents: Patch patterns, code quality, and issue complexity in real-world github scenarios. In *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, page 657–668. IEEE.

Zhi Chen, Wei Ma, and Lingxiao Jiang. 2025d. Beyond final code: A process-oriented error analysis of software development agents in real-world github scenarios. *Preprint*, arXiv:2503.12374.

Zhilong Chen, Chengzong Zhao, Boyuan Chen, Dayi Lin, Yihao Chen, Arthur Leung, Gopi Krishnan Rajbahadur, Gustavo A. Oliva, Haoxiang Zhang, Aaditya Bhatia, Chong Chun Yong, and Ahmed E. Hassan. 2025e. Repoforge: Training a sota fast-thinking swe agent with an end-to-end data curation pipeline synergizing sft and rl at scale. *Preprint*, arXiv:2508.01550.

Zimin Chen, Yue Pan, Siyu Lu, Jiayi Xu, Claire Le Goues, Martin Monperrus, and He Ye. 2025f. Prometheus: Unified knowledge graphs for issue resolution in multilingual codebases. *Preprint*, arXiv:2507.19942.

Anton Cheshkov, Pavel Zadorozhny, Rodion Levichev, Evgeny Maslov, and Ronaldo Franco Jaldin. 2024. Exploring the potential of conversational test suite based program repair on swe-bench. *Preprint*, arXiv:2410.04485.

Alejandro Cuadron, Dacheng Li, Wenjie Ma, Xingyao Wang, Yichuan Wang, Siyuan Zhuang, Shu Liu, Luis Gaspar Schroeder, Tian Xia, Huanzhi Mao, Nicholas Thumiger, Aditya Desai, Ion Stoica, Ana Klimovic, Graham Neubig, and Joseph E. Gonzalez. 2025. The danger of overthinking: Examining the reasoning-action dilemma in agentic tasks. *Preprint*, arXiv:2502.08235.

Jeff Da, Clinton Wang, Xiang Deng, Yuntao Ma, Nikhil Barhate, and Sean Hendryx. 2025. Agent-rlvr: Training software engineering agents via guidance and environment rewards. *Preprint*, arXiv:2506.11425.

DeepSeek-AI, Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenhao Xu, Chong Ruan, Damai Dai, Daya Guo, Dejian Yang, and 245 others. 2025. Deepseek-v3.2: Pushing the frontier of open large language models. *Preprint*, arXiv:2512.02556.

Xiang Deng, Jeff Da, Edwin Pan, Yannis Yiming He, Charles Ide, Kanak Garg, Niklas Lauffer, Andrew Park, Nitin Pasari, Chetan Rane, Karmini Sampath, Maya Krishnan, Srivatsa Kundurthy, Sean Hendryx, Zifan Wang, Chen Bo Calvin Zhang, Noah Jacobson,

Bing Liu, and Brad Kenstler. 2025. Swe-bench pro: Can ai agents solve long-horizon software engineering tasks? *Preprint*, arXiv:2509.16941.

Yaxin Du, Yuzhu Cai, Yifan Zhou, Cheng Wang, Yu Qian, Xianghe Pang, Qian Liu, Yue Hu, and Siheng Chen. 2025. Swe-dev: Evaluating and training autonomous feature-driven software development. *Preprint*, arXiv:2505.16975.

Ryan Ehrlich, Bradley Brown, Jordan Juravsky, Ronald Clark, Christopher Ré, and Azalia Mirhoseini. 2025. Codemonkeys: Scaling test-time compute for software engineering. *Preprint*, arXiv:2501.14723.

Kelin Fu, Tianyu Liu, Zeyu Shang, Yingwei Ma, Jian Yang, Jiaheng Liu, and Kaigui Bian. 2025. Multi-docker-eval: A 'shovel of the gold rush' benchmark on automatic environment building for software engineering. *Preprint*, arXiv:2512.06915.

Shubham Gandhi, Jason Tsay, Jatin Ganhotra, Kiran Kate, and Yara Rizk. 2025. When agents go astray: Course-correcting swe agents with prms. *Preprint*, arXiv:2509.02360.

Spandan Garg, Benjamin Steenhoek, and Yufan Huang. 2025. Saving swe-bench: A benchmark mutation approach for realistic agent evaluation. *Preprint*, arXiv:2510.08996.

Anmol Gautam, Kishore Kumar, Adarsh Jha, Mukunda NS, and Ishaan Bhola. 2024. Supercoder2.0: Technical report on exploring the feasibility of llms as autonomous programmer. *Preprint*, arXiv:2409.11190.

Alexander Golubev, Maria Trofimova, Sergei Polezhaev, Ibragim Badertdinov, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Sergey Abramov, Andrei Andriushchenko, Filipp Fisin, Sergei Skvortsov, and Boris Yangel. 2025. Training long-context, multi-turn software engineering agents with reinforcement learning. *Preprint*, arXiv:2508.03501.

Jiale Guo, Suizhi Huang, Mei Li, Dong Huang, Xingsheng Chen, Regina Zhang, Zhijiang Guo, Han Yu, Siu-Ming Yiu, Pietro Lio, and Kwok-Yan Lam. 2025a. A comprehensive survey on benchmarks and solutions in software engineering of llm-empowered agentic system. *Preprint*, arXiv:2510.09721.

Lianghong Guo, Wei Tao, Runhan Jiang, Yanlin Wang, Jiachi Chen, Xilin Liu, Yuchi Ma, Mingzhi Mao, Hongyu Zhang, and Zibin Zheng. 2025b. Omnigirl: A multilingual and multimodal benchmark for github issue resolution. *Proceedings of the ACM on Software Engineering*, 2(ISSTA):24–46.

Lianghong Guo, Yanlin Wang, Caihua Li, Pengyu Yang, Jiachi Chen, Wei Tao, Yingtian Zou, Duyu Tang, and Zibin Zheng. 2025c. Swe-factory: Your automated factory for issue resolution training data and evaluation benchmarks. *Preprint*, arXiv:2506.10954.

Dhruv Gupta, Gayathri Ganesh Lakshmy, and Yiqing Xie. 2025. Sacl: Understanding and combating textual bias in code retrieval with semantic-augmented reranking and localization. *Preprint*, arXiv:2506.20081.

Xinyi He, Qian Liu, Mingzhe Du, Lin Yan, Zhijie Fan, Yiming Huang, Zejian Yuan, and Zejun Ma. 2025a. Swe-perf: Can language models optimize code performance on real-world repositories? *Preprint*, arXiv:2507.12415.

Zhenyu He, Qingping Yang, Wei Sheng, Xiaojian Zhong, Kechi Zhang, Chenxin An, Wenlei Shi, Tianle Cai, Di He, Jiaze Chen, and Jingjing Xu. 2025b. Swe-swiss: A multi-task fine-tuning and rl recipe for high-performance issue resolution. https://www.notion.so/SWE-Swiss-A-Multi-Task-Fine-Tuning-and-RL-Recipe-for-High-Performance-Issue-Resolution-21e174dedd4880ea829ed4c861c44f88. Notion Blog.

Sirui Hong, Mingchen Zhuge, Jiaqi Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. Metagpt: Meta programming for a multi-agent collaborative framework. *Preprint*, arXiv:2308.00352.

Hao Hu, Hongyu Zhang, Jifeng Xuan, and Weigang Sun. 2014. Effective bug triage based on historical bug-fix information. In *Proceedings of the 2014 IEEE 25th International Symposium on Software Reliability Engineering*, ISSRE '14, page 122–132, USA. IEEE Computer Society.

Kai Huang, Zhengzi Xu, Su Yang, Hongyu Sun, Xuejun Li, Zheng Yan, and Yuqing Zhang. 2023. A survey on automated program repair techniques. *Preprint*, arXiv:2303.18184.

Kai Huang, Jian Zhang, Xiaofei Xie, and Chunyang Chen. 2025. Seeing is fixing: Cross-modal reasoning with multimodal llms for visual software issue fixing. *Preprint*, arXiv:2506.16136.

Naman Jain, Jaskirat Singh, Manish Shetty, Liang Zheng, Koushik Sen, and Ion Stoica. 2025. R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents. *Preprint*, arXiv:2504.07164.

Juyong Jiang, Fan Wang, Jiasi Shen, Sungju Kim, and Sunghun Kim. 2024. A survey on large language models for code generation. *arXiv preprint arXiv:2406.00515*.

Mingjian Jiang, Yangjun Ruan, Luis Lastras, Pavan Kapanipathi, and Tatsunori Hashimoto. 2025a. Putting it all into context: Simplifying agents with lclms. *Preprint*, arXiv:2505.08120.

Zhonghao Jiang, Xiaoxue Ren, Meng Yan, Wei Jiang, Yong Li, and Zhongxin Liu. 2025b. Issue localization via llm-driven iterative code graph searching. *Preprint*, arXiv:2503.22424.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? *Preprint*, arXiv:2310.06770.

Alexander Kovrigin, Aleksandra Eliseeva, Yaroslav Zharov, and Timofey Bryksin. 2024. On the importance of reasoning for context retrieval in repository-level code editing. *Preprint*, arXiv:2406.04464.

Bin Lei, Weitai Kang, Zijian Zhang, Winson Chen, Xi Xie, Shan Zuo, Mimi Xie, Ali Payani, Mingyi Hong, Yan Yan, and Caiwen Ding. 2025. Infantagent-next: A multimodal generalist agent for automated computer interaction. *Preprint*, arXiv:2505.10887.

Bin Lei, Yuchen Li, Yiming Zeng, Tao Ren, Yi Luo, Tianyu Shi, Zitian Gao, Zeyu Hu, Weitai Kang, and Qiuwu Chen. 2024. Infant agent: A tool-integrated, logic-driven agent with cost-effective api usage. *Preprint*, arXiv:2411.01114.

Han Li, Yuling Shi, Shaoxin Lin, Xiaodong Gu, Heng Lian, Xin Wang, Yantao Jia, Tao Huang, and Qianxiang Wang. 2025a. Swe-debate: Competitive multi-agent debate for software issue resolution. *Preprint*, arXiv:2507.23348.

Hongwei Li, Yuheng Tang, Shiqi Wang, and Wenbo Guo. 2025b. Patchpilot: A cost-efficient software engineering agent with early attempts on formal verification. *Preprint*, arXiv:2502.02747.

Wei Li, Xin Zhang, Zhongxin Guo, Shaoguang Mao, Wen Luo, Guangyue Peng, Yangyu Huang, Houfeng Wang, and Scarlett Li. 2025c. Fea-bench: A benchmark for evaluating repository-level code generation for feature implementation. *Preprint*, arXiv:2503.06680.

Shanchao Liang, Spandan Garg, and Roshanak Zilouchian Moghaddam. 2025. The swe-bench illusion: When state-of-the-art llms remember instead of reason. *Preprint*, arXiv:2506.12286.

Jiaye Lin, Yifu Guo, Yuzhen Han, Sen Hu, Ziyi Ni, Licheng Wang, Mingguang Chen, Hongzhang Liu, Ronghao Chen, Yangfan He, Daxin Jiang, Binxing Jiao, Chen Hu, and Huacan Wang. 2025. Seagent: Self-evolution trajectory optimization in multi-step reasoning with llm-based agents. *Preprint*, arXiv:2508.02085.

Yalan Lin, Yingwei Ma, Rongyu Cao, Binhua Li, Fei Huang, Xiaodong Gu, and Yongbin Li. 2024. Llms as continuous learners: Improving the reproduction of defective code in software issues. *Preprint*, arXiv:2411.13941.

Bingchang Liu, Chaoyu Chen, Zi Gong, Cong Liao, Huan Wang, Zhichao Lei, Ming Liang, Dajun Chen, Min Shen, Hailian Zhou, Wei Jiang, Hang Yu, and Jianguo Li. 2024a. Mftcoder: Boosting code llms with multitask fine-tuning. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, KDD '24, page 5430–5441, New York, NY, USA. Association for Computing Machinery.

Shukai Liu, Jian Yang, Bo Jiang, Yizhi Li, Jinyang Guo, Xianglong Liu, and Bryan Dai. 2025a. Context as a tool: Context management for long-horizon swe-agents. *Preprint*, arXiv:2512.22087.

Simiao Liu, Fang Liu, Liehao Li, Xin Tan, Yinghao Zhu, Xiaoli Lian, and Li Zhang. 2025b. An empirical study on failures in automated issue solving. *Preprint*, arXiv:2509.13941.

Ye Liu, Rui Meng, Shafiq Joty, silvio savarese, Caiming Xiong, Yingbo Zhou, and Semih Yavuz. 2025c. CodeXEmbed: A generalist embedding model family for multilingual and multi-task code retrieval. In *Second Conference on Language Modeling*.

Yizhou Liu, Pengfei Gao, Xinchen Wang, Jie Liu, Yexuan Shi, Zhao Zhang, and Chao Peng. 2024b. Marscode agent: Ai-native automated bug fixing. *Preprint*, arXiv:2409.00899.

Michael Luo, Naman Jain, Jaskirat Singh, Sijun Tan, Ameen Patel, Qingyang Wu, Alpay Ariyak, Colin Cai, Tarun Venkat, Shang Zhu, Ben Athiwaratkun, Manan Roongta, Ce Zhang, Li Erran Li, Raluca Ada Popa, Koushik Sen, and Ion Stoica. 2025. Deepswe: Training a state-of-the-art coding agent from scratch by scaling rl. https://pretty-radio-b75.notion.site/DeepSWE-Training-a-Fully-Open-sourced-State-of-the-Art-Coding-Agent-by-Scaling-RL-22281902c1468193aabbe9a8c59bbe33. Notion Blog.

Jeffrey Jian Ma, Milad Hashemi, Amir Yazdanbakhsh, Kevin Swersky, Ofir Press, Enhui Li, Vijay Janapa Reddi, and Parthasarathy Ranganathan. 2025a. Swefficiency: Can language models optimize real-world repositories on real workloads? *Preprint*, arXiv:2511.06090.

Yingwei Ma, Rongyu Cao, Yongchang Cao, Yue Zhang, Jue Chen, Yibo Liu, Yuchen Liu, Binhua Li, Fei Huang, and Yongbin Li. 2024. Lingma swe-gpt: An open development-process-centric language model for automated software improvement. *Preprint*, arXiv:2411.00622.

Yingwei Ma, Qingping Yang, Rongyu Cao, Binhua Li, Fei Huang, and Yongbin Li. 2025b. Alibaba lingmaagent: Improving automated issue resolution via comprehensive repository exploration. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering*, FSE Companion '25, page 238–249. ACM.

Zexiong Ma, Chao Peng, Pengfei Gao, Xiangxin Meng, Yanzhen Zou, and Bing Xie. 2025c. Sorft: Issue resolving with subtask-oriented reinforced fine-tuning. *Preprint*, arXiv:2502.20127.

Matias Martinez and Xavier Franch. 2025. Dissecting the swe-bench leaderboards: Profiling submitters and architectures of llm- and agent-based repair systems. *Preprint*, arXiv:2506.17208.

Noble Saji Mathews and Meiyappan Nagappan. 2025. Is your automated software engineer trustworthy? *Preprint*, arXiv:2506.17812.

Sanket Mhatre, Yasharth Bajpai, Sumit Gulwani, Emerson Murphy-Hill, and Gustavo Soares. 2025. Swe-sharp-bench: A reproducible benchmark for c# software engineering tasks. *Preprint*, arXiv:2511.02352.

Samuel Miserendino, Michele Wang, Tejal Patwardhan, and Johannes Heidecke. 2025. Swe-lancer: Can frontier llms earn $1 million from real-world freelance software engineering? *Preprint*, arXiv:2502.12115.

Fangwen Mu, Junjie Wang, Lin Shi, Song Wang, Shoubin Li, and Qing Wang. 2025. Experepair: Dual-memory enhanced llm-based repository-level program repair. *Preprint*, arXiv:2506.10484.

Niels Mündler, Mark Niklas Müller, Jingxuan He, and Martin Vechev. 2024. Swt-bench: Testing and validating real-world bug-fixes with code agents. In *Advances in Neural Information Processing Systems*, volume 37, pages 81857–81887. Curran Associates, Inc.

Noor Nashid, Islem Bouzenia, Michael Pradel, and Ali Mesbah. 2025. Issue2test: Generating reproducing test cases from issue reports. *Preprint*, arXiv:2503.16320.

Nebius. 2024. Scaling data collection for training SWE agents. `https://nebius.com/blog/posts/scaling-data-collection-for-training-swe-agents`. Accessed: 2025-12-12.

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. *Preprint*, arXiv:2507.09108.

OpenAI. 2024. Introducing swe-bench verified | openai. [Online; accessed 2025-09-22].

Siru Ouyang, Jun Yan, I-Hung Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long T. Le, Samira Daruki, Xiangru Tang, Vishy Tirumalashetty, George Lee, Mahsan Rofouei, Hangfei Lin, Jiawei Han, Chen-Yu Lee, and Tomas Pfister. 2025a. Reasoningbank: Scaling agent self-evolving with reasoning memory. *Preprint*, arXiv:2509.25140.

Siru Ouyang, Wenhao Yu, Kaixin Ma, Zilin Xiao, Zhihan Zhang, Mengzhao Jia, Jiawei Han, Hongming Zhang, and Dong Yu. 2025b. Repograph: Enhancing ai software engineering with repository-level code graph. *Preprint*, arXiv:2410.14684.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2025a. Training software engineering agents and verifiers with swe-gym. *Preprint*, arXiv:2412.21139.

Ruwei Pan, Hongyu Zhang, and Chao Liu. 2025b. Codecor: An llm-based self-reflective multi-agent framework for code generation. *Preprint*, arXiv:2501.07811.

Minh V. T. Pham, Huy N. Phan, Hoang N. Phan, Cuong Le Chi, Tien N. Nguyen, and Nghi D. Q. Bui. 2025. Swe-synth: Synthesizing verifiable bug-fix data to enable large language models in resolving real-world bugs. *Preprint*, arXiv:2504.14757.

Thanosan Prathifkumar, Noble Saji Mathews, and Meiyappan Nagappan. 2025. Does swe-bench-verified test agent ability or model memory? *Preprint*, arXiv:2512.10218.

Binhang Qi, Hailong Sun, Wei Yuan, Hongyu Zhang, and Xiangxin Meng. 2022. Dreamloc: A deep relevance matching-based framework for bug localization. *IEEE Transactions on Reliability*, 71(1):235–249.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, page 15174–15186. Association for Computational Linguistics.

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. 2025. Swe-polybench: A multi-language benchmark for repository level evaluation of coding agents. *Preprint*, arXiv:2504.08703.

Abhinav Rastogi, Adam Yang, Albert Q. Jiang, Alexander H. Liu, Alexandre Sablayrolles, Amélie Héliou, Amélie Martin, Anmol Agarwal, Andy Ehrenberg, Andy Lo, Antoine Roux, Arthur Darcet, Arthur Mensch, Baptiste Bout, Baptiste Rozière, Baudouin De Monicault, Chris Bamford, Christian Wallenwein, Christophe Renaudin, and 84 others. 2025. Devstral: Fine-tuning language models for coding agent applications. *Preprint*, arXiv:2509.25193.

Revanth Gangi Reddy, Tarun Suresh, JaeHyeok Doo, Ye Liu, Xuan Phi Nguyen, Yingbo Zhou, Semih Yavuz, Caiming Xiong, Heng Ji, and Shafiq Joty. 2025. Swerank: Software issue localization with code ranking. *Preprint*, arXiv:2505.07849.

Haifeng Ruan, Yuntong Zhang, and Abhik Roychoudhury. 2025. Specrover: Code intent extraction via llms. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, page 963–974. IEEE.

Amirali Sajadi, Kostadin Damevski, and Preetha Chatterjee. 2025. Are ai-generated fixes secure? analyzing llm and agent patches on swe-bench. *Preprint*, arXiv:2507.02976.

ByteDance Seed, :, Jiaze Chen, Tiantian Fan, Xin Liu, Lingjun Liu, Zhiqi Lin, Mingxuan Wang, Chengyi Wang, Xiangpeng Wei, Wenyuan Xu, Yufeng Yuan, Yu Yue, Lin Yan, Qiying Yu, Xiaochen Zuo, Chi Zhang, Ruofei Zhu, Zhecheng An, and 255 others. 2025. Seed1.5-thinking: Advancing superb reasoning models with reinforcement learning. *Preprint*, arXiv:2504.13914.

Manish Shetty, Naman Jain, Jinjian Liu, Vijay Kethanaboyina, Koushik Sen, and Ion Stoica. 2025. Gso: Challenging software optimization tasks for evaluating swe-agents. *Preprint*, arXiv:2505.23671.

KaShun Shum, Binyuan Hui, Jiawei Chen, Lei Zhang, X. W., Jiaxi Yang, Yuzhen Huang, Junyang Lin, and Junxian He. 2025. Swe-rm: Execution-free feedback for software engineering agents. *Preprint*, arXiv:2512.21919.

Atefeh Sohrabizadeh, Jialin Song, Mingjie Liu, Rajarshi Roy, Chankyu Lee, Jonathan Raiman, and Bryan Catanzaro. 2025. Nemotron-CORTEXA: Enhancing LLM agents for software engineering tasks via improved localization and solution diversity. In *Forty-second International Conference on Machine Learning*.

Aditya Bharat Soni, Boxuan Li, Xingyao Wang, Valerie Chen, and Graham Neubig. 2025. Coding agents with multimodal browsing are generalist problem solvers. *Preprint*, arXiv:2506.03011.

Hongjin SU, Ruoxi Sun, Jinsung Yoon, Pengcheng Yin, Tao Yu, and Sercan O Arik. 2025. Learn-by-interact: A data-centric framework for self-adaptive agents in realistic environments. In *The Thirteenth International Conference on Learning Representations*.

Weiwei Sun, Miao Lu, Zhan Ling, Kang Liu, Xuesong Yao, Yiming Yang, and Jiecao Chen. 2025. Scaling long-horizon llm agent via context-folding. *Preprint*, arXiv:2510.11967.

Tarun Suresh, Revanth Gangi Reddy, Yifei Xu, Zach Nussbaum, Andriy Mulyar, Brandon Duderstadt, and Heng Ji. 2025. CoRNStack: High-quality contrastive data for better code retrieval and reranking. In *The Thirteenth International Conference on Learning Representations*.

Xiangru Tang, Tianrui Qin, Tianhao Peng, Ziyang Zhou, Daniel Shao, Tingting Du, Xinming Wei, Peng Xia, Fang Wu, He Zhu, Ge Zhang, Jiaheng Liu, Xingyao Wang, Sirui Hong, Chenglin Wu, Hao Cheng, Chi Wang, and Wangchunshu Zhou. 2025a. Agent kb: Leveraging cross-domain experience for agentic problem solving. *Preprint*, arXiv:2507.06229.

Xunzhu Tang, Jiechao Gao, Jin Xu, Tiezhu Sun, Yewei Song, Saad Ezzini, Wendkûuni C. Ouédraogo, Jacques Klein, and Tegawendé F. Bissyandé. 2025b. SynFix: Dependency-aware program repair via RelationGraph analysis. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 4878–4894, Vienna, Austria. Association for Computational Linguistics.

Yuheng Tang, Hongwei Li, Kaijie Zhu, Michael Yang, Yangruibo Ding, and Wenbo Guo. 2025c. Co-patcher: Collaborative software patching with component(s)-specific small reasoning models. *Preprint*, arXiv:2505.18955.

Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. 2024. Magis: Llm-based multi-agent framework for github issue resolution. *Preprint*, arXiv:2403.17927.

Natalia Tarasova, Enrique Balp-Straffon, Aleksei Iancheruk, Yevhenii Sielskyi, Nikita Kozodoi, Liam H. Byrne, Jack Butler, Dayuan jiang, Marcin Czelej, Andrew Ang, Yash Shah, Roi Blanco, and Sergei Ivanov. 2025. SWE-infrabench: Evaluating language models on cloud infrastructure code. In *NeurIPS 2025 Workshop on Evaluating the Evolving LLM Lifecycle: Benchmarks, Emergent Abilities, and Scaling*.

Vali Tawosi, Salwa Alamir, Xiaomo Liu, and Manuela Veloso. 2025. Meta-rag on large codebases using code summarization. *Preprint*, arXiv:2508.02611.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, and 150 others. 2025a. Kimi k2: Open agentic intelligence. *Preprint*, arXiv:2507.20534.

Meituan LongCat Team, Anchun Gui, Bei Li, Bingyang Tao, Bole Zhou, Borun Chen, Chao Zhang, Chao Zhang, Chengcheng Han, Chenhui Yang, Chi Zhang, Chong Peng, Chuyu Zhang, Cong Chen, Fengcun Li, Gang Xu, Guoyuan Lin, Hao Jiang, Hao Liang, and 108 others. 2025b. Introducing longcat-flash-thinking: A technical report. *Preprint*, arXiv:2509.18883.

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. 2025c. Trae agent: An llm-based agent for software engineering with test-time scaling. *Preprint*, arXiv:2507.23370.

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. *Preprint*, arXiv:2512.18470.

Arihant Tripathy, Ch Pavan Harshit, and Karthik Vaidhyanathan. 2025. Swenergy: An empirical study on energy efficiency in agentic issue resolution frameworks with slms. *Preprint*, arXiv:2512.09543.

Nguyen Phu Vinh, Anh Chung Hoang, Chris Ngo, and Truong-Son Hy. 2025. Repeton: Structured bug repair with react-guided patch-and-test cycles. *Preprint*, arXiv:2506.08173.

Boshi Wang, Weijian Xu, Yunsheng Li, Mei Gao, Yujia Xie, Huan Sun, and Dongdong Chen. 2025a. Improving code localization with repository memory. *Preprint*, arXiv:2510.01003.

Chaozheng Wang, Zezhou Yang, Shuzheng Gao, Cuiyun Gao, Ting Peng, Hailiang Huang, Yuetang Deng, and Michael Lyu. 2025b. Rag or fine-tuning? a comparative study on lcms-based code completion in industry. *arXiv preprint arXiv:2505.15179*.

Haoran Wang, Zhenyu Hou, Yao Wei, Jie Tang, and Yuxiao Dong. 2025c. Swe-dev: Building software engineering agents with training and inference scaling. *Preprint*, arXiv:2506.07636.

Jinghui Wang, Shaojie Wang, Yinghan Cui, Xuxing Chen, Chao Wang, Xiaojiang Zhang, Minglei Zhang, Jiarong Zhang, Wenhao Zhuang, Yuchen Cao, Wankang Bao, Haimo Li, Zheng Lin, Huiming Wang, Haoyang Huang, Zongxian Feng, Zizheng Zhan, Ken Deng, Wen Xiang, and 8 others. 2025d. Seamlessflow: A trainer agent isolation rl framework achieving bubble-free pipelines via tag scheduling. *Preprint*, arXiv:2508.11553.

Junhao Wang, Daoguang Zan, Shulin Xin, Siyao Liu, Yurong Wu, and Kai Shen. 2025e. Swe-mirror: Scaling issue-resolving datasets by mirroring issues across repositories. *Preprint*, arXiv:2509.08724.

Lilin Wang, Lucas Ramalho, Alan Celestino, Phuc Anthony Pham, Yu Liu, Umang Kumar Sinha, Andres Portillo, Onassis Osunwa, and Gabriel Maduekwe. 2025f. Swe-bench++: A framework for the scalable generation of software engineering benchmarks from open-source repositories. *Preprint*, arXiv:2512.17419.

Ruiyi Wang and Prithviraj Ammanabrolu. 2025. A practitioner's guide to multi-turn agentic reinforcement learning. *Preprint*, arXiv:2510.01132.

Xinchen Wang, Pengfei Gao, Xiangxin Meng, Chao Peng, Ruida Hu, Yun Lin, and Cuiyun Gao. 2024. Aegis: An agent-based framework for general bug reproduction from issue descriptions. *Preprint*, arXiv:2411.18015.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, and 5 others. 2025g. Openhands: An open platform for ai software developers as generalist agents. *Preprint*, arXiv:2407.16741.

Yanlin Wang, Yanli Wang, Daya Guo, Jiachi Chen, Ruikai Zhang, Yuchi Ma, and Zibin Zheng. 2025h. Rlcoder: Reinforcement learning for repository-level code completion. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, page 1140–1152. IEEE.

Yanlin Wang, Wanjun Zhong, Yanxian Huang, Ensheng Shi, Min Yang, Jiachi Chen, Hui Li, Yuchi Ma, Qianxiang Wang, and Zibin Zheng. 2025i. Agents in software engineering: survey, landscape, and vision. *Automated Software Engineering*, 32(2).

You Wang, Michael Pradel, and Zhongxin Liu. 2025j. Are "solved issues" in SWE-bench really solved correctly? an empirical study. *Preprint*, arxiv:2503.15223 [cs].

Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2025k. CodeRAG-bench: Can retrieval augment code generation? In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3199–3214, Albuquerque, New Mexico. Association for Computational Linguistics.

Yuxiang Wei, Olivier Duchenne, Jade Copet, Quentin Carbonneaux, Lingming Zhang, Daniel Fried, Gabriel Synnaeve, Rishabh Singh, and Sida I. Wang. 2025a. Swe-rl: Advancing llm reasoning via reinforcement learning on open software evolution. *Preprint*, arXiv:2502.18449.

Yuxiang Wei, Zhiqing Sun, Emily McMilin, Jonas Gehring, David Zhang, Gabriel Synnaeve, Daniel Fried, Lingming Zhang, and Sida Wang. 2025b. Toward training superintelligent software agents through self-play swe-rl. *Preprint*, arXiv:2512.18552.

W. Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A survey on software fault localization. *IEEE Trans. Softw. Eng.*, 42(8):707–740.

Di Wu, Wasi Uddin Ahmad, Dejiao Zhang, Murali Krishna Ramanathan, and Xiaofei Ma. 2024. Repoformer: Selective retrieval for repository-level code completion. *arXiv preprint arXiv:2403.10059*.

Junde Wu. 2025. Git context controller: Manage the context of llm-based agents like git. *Preprint*, arXiv:2508.00031.

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

Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. 2025. Live-swe-agent: Can software engineering agents self-evolve on the fly? *Preprint*, arXiv:2511.13646.

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. *Preprint*, arXiv:2501.05040.

Bojian Xiong, Yikun Lei, Xikai Liu, Shaowei Zhang, Pengyun Zhu, Yan Liu, Yongqi Leng, Ling Shi, Meizhi Zhong, Yurong Zhang, Yan Gao, Yiwu, Yao Hu, and Deyi Xiong. 2025. Think-search-patch: A retrieval-augmented reasoning framework for repository-level code repair. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1555–1566, Suzhou (China). Association for Computational Linguistics.

Jingxuan Xu, Ken Deng, Weihao Li, Songwei Yu, Huaixi Tang, Haoyang Huang, Zhiyi Lai, Zizheng Zhan, Yanan Wu, Chenchen Zhang, Kepeng Lei, Yifan Yao, Xinping Lei, Wenqiang Zhu, Zongxian Feng, Han Li, Junqi Xiong, Dailin Li, Zuchen Gao, and 20 others. 2025a. Swe-compass: Towards unified evaluation of agentic coding abilities for large language models. *Preprint*, arXiv:2511.05459.

Wendong Xu, Jing Xiong, Chenyang Zhao, Qiujiang Chen, Haoran Wang, Hui Shen, Zhongwei Wan, Jianbo Dai, Taiqiang Wu, He Xiao, Chaofan Tao, Z. Morley Mao, Ying Sheng, Zhijiang Guo, Hongxia Yang, Bei Yu, Lingpeng Kong, Quanquan Gu, and Ngai Wong. 2025b. Swingarena: Competitive programming arena for long-context github issue solving. *Preprint*, arXiv:2505.23932.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Boyang Yang, Jiadong Ren, Shunfu Jin, Yang Liu, Feng Liu, Bach Le, and Haoye Tian. 2025b. Enhancing repository-level software repair via repository-aware knowledge graphs. *Preprint*, arXiv:2503.21710.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. 2024a. Swe-agent: Agent-computer interfaces enable automated software engineering. *Preprint*, arXiv:2405.15793.

John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida I. Wang, and Ofir Press. 2024b. Swe-bench multimodal: Do ai systems generalize to visual software domains? *Preprint*, arXiv:2410.03859.

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

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, and 2 others. 2025d. Kimi-dev: Agentless training as skill prior for swe-agents. *Preprint*, arXiv:2509.23045.

Boxi Yu, Yuxuan Zhu, Pinjia He, and Daniel Kang. 2025a. Utboost: Rigorous evaluation of coding agents on swe-bench. *Preprint*, arXiv:2506.09289.

Zhongming Yu, Hejia Zhang, Yujie Zhao, Hanxian Huang, Matrix Yao, Ke Ding, and Jishen Zhao. 2025b. Orcaloca: An llm agent framework for software issue localization. *Preprint*, arXiv:2502.00350.

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. 2025. Multi-swe-bench: A multilingual benchmark for issue resolving. *Preprint*, arXiv:2504.02605.

Daoguang Zan, Zhirong Huang, Ailun Yu, Shaoxin Lin, Yifan Shi, Wei Liu, Dong Chen, Zongshuai Qi, Hao Yu, Lei Yu, Dezhi Ran, Muhan Zeng, Bo Shen, Pan Bian, Guangtai Liang, Bei Guan, Pengjie Huang, Tao Xie, Yongji Wang, and Qianxiang Wang. 2024. Swe-bench-java: A github issue resolving benchmark for java. *Preprint*, arXiv:2408.14354.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, and 1 others. 2025a. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*.

Guangtao Zeng, Maohao Shen, Delin Chen, Zhenting Qi, Subhro Das, Dan Gutfreund, David Cox, Gregory Wornell, Wei Lu, Zhang-Wei Hong, and Chuang Gan. 2025b. Satori-swe: Evolutionary test-time scaling for sample-efficient software engineering. *Preprint*, arXiv:2505.23604.

Liang Zeng, Yongcong Li, Yuzhen Xiao, Changshi Li, Chris Yuhao Liu, Rui Yan, Tianwen Wei, Jujie He, Xuchen Song, Yang Liu, and Yahui Zhou. 2025c. Skywork-swe: Unveiling data scaling laws for software engineering in llms. *Preprint*, arXiv:2506.19290.

Zizheng Zhan, Ken Deng, Jinghui Wang, Xiaojiang Zhang, Huaixi Tang, Minglei Zhang, Zhiyi Lai, Haoyang Huang, Wen Xiang, Kun Wu, Wenhao Zhuang, Shaojie Wang, Shangpeng Yan, Kepeng Lei, Zongxian Feng, Huiming Wang, Zheng Lin, Mengtong Li, Mengfei Xie, and 21 others. 2025. Kat-coder technical report. *Preprint*, arXiv:2510.18779.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. 2025a. Darwin godel machine: Open-ended evolution of self-improving agents. *Preprint*, arXiv:2505.22954.

Kechi Zhang, Huangzhao Zhang, Ge Li, Jinliang You, Jia Li, Yunfei Zhao, and Zhi Jin. 2025b. Sealign: Alignment training for software engineering agent. *Preprint*, arXiv:2503.18455.

Kexun Zhang, Weiran Yao, Zuxin Liu, Yihao Feng, Zhiwei Liu, Rithesh Murthy, Tian Lan, Lei Li, Renze Lou, Jiacheng Xu, Bo Pang, Yingbo Zhou, Shelby Heinecke, Silvio Savarese, Huan Wang, and Caiming Xiong. 2024a. Diversity empowers intelligence: Integrating expertise of software engineering agents. *Preprint*, arXiv:2408.07060.

Lei Zhang, Jiaxi Yang, Min Yang, Jian Yang, Mouxiang Chen, Jiajun Zhang, Zeyu Cui, Binyuan Hui, and Junyang Lin. 2025c. Swe-flow: Synthesizing software engineering data in a test-driven manner. *Preprint*, arXiv:2506.09003.

Linghao Zhang, Shilin He, Chaoyun Zhang, Yu Kang, Bowen Li, Chengxing Xie, Junhao Wang, Maoquan Wang, Yufan Huang, Shengyu Fu, Elsie Nallipogu, Qingwei Lin, Yingnong Dang, Saravan Rajmohan, and Dongmei Zhang. 2025d. Swe-bench goes live! *Preprint*, arXiv:2505.23419.

Linhao Zhang, Daoguang Zan, Quanshun Yang, Zhirong Huang, Dong Chen, Bo Shen, Tianyu Liu, Yongshun Gong, Huang Pengjie, Xudong Lu, Guangtai Liang, Lizhen Cui, and Qianxiang Wang. 2025e. CodeV: Issue resolving with visual data. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 7350–7361, Vienna, Austria. Association for Computational Linguistics.

Quanjun Zhang, Chunrong Fang, Yuxiang Ma, Weisong Sun, and Zhenyu Chen. 2023. A survey of learning-based automated program repair. *ACM Trans. Softw. Eng. Methodol.*, 33(2).

Yilin Zhang, Xinran Zhao, Zora Zhiruo Wang, Chenyang Yang, Jiayi Wei, and Tongshuang Wu. 2025f. cast: Enhancing code retrieval-augmented generation with structural chunking via abstract syntax tree. *Preprint*, arXiv:2506.15655.

Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. 2024b. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA '24, page 1592–1604. ACM.

Zejun Zhang, Jian Wang, Qingyun Yang, Yifan Pan, Yi Tang, Yi Li, Zhenchang Xing, Tian Zhang, Xuandong Li, and Guoan Zhang. 2025g. A benchmark for localizing code and non-code issues in software projects. *Preprint*, arXiv:2509.25242.

Dewu Zheng, Yanlin Wang, Ensheng Shi, Xilin Liu, Yuchi Ma, Hongyu Zhang, and Zibin Zheng. 2025. Top general performance = top domain performance? domaincodebench: A multi-domain code generation benchmark. *Preprint*, arXiv:2412.18573.

Jian Zhou, Hongyu Zhang, and David Lo. 2012. Where should the bugs be fixed? - more accurate information retrieval-based bug localization based on bug reports. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, page 14–24. IEEE Press.

Xuhui Zhou, Valerie Chen, Zora Zhiruo Wang, Graham Neubig, Maarten Sap, and Xingyao Wang. 2025. Tom-swe: User mental modeling for software engineering agents. *Preprint*, arXiv:2510.21903.

Yuxuan Zhu, Tengjun Jin, Yada Pruksachatkun, Andy Zhang, Shu Liu, Sasha Cui, Sayash Kapoor, Shayne Longpre, Kevin Meng, Rebecca Weiss, Fazl Barez, Rahul Gupta, Jwala Dhamala, Jacob Merizian, Mario Giulianelli, Harry Coppock, Cozmin Ududec, Jasjeet Sekhon, Jacob Steinhardt, and 6 others. 2025. Establishing best practices for building rigorous agentic benchmarks. *Preprint*, arXiv:2507.02825.

A Appendix

A.1 Related work

Code generation. The application of LLMs in the programming domain has witnessed explosive growth. Early research focused primarily on function-level code generation, with benchmarks such as HumanEval (Chen et al., 2021) serving as standard metrics. However, generic benchmarks often fail to capture the nuances of real-world development. To bridge this gap, recent initiatives (Zheng et al., 2025; Li et al., 2025c) have attempted to extend evaluation tasks to align more closely with realistic software development scenarios, revealing the limitations of general models in specialized domains. Concurrently, methods are also evolving to capture these broader contexts. While foundational approaches primarily relied on SFT (Liu et al., 2024a) or standard retrieval-augmented generation (Wu et al., 2024; Wang et al., 2025k), RL-based methods emerged as a pivotal direction for handling complex coding tasks (Wang et al., 2025h).

Automated software generation. The primary goal of this task is to autonomously construct complete and executable software systems starting from high-level natural language requirements. Unlike code completion, it necessitates covering the SDLC, including requirement analysis, system design, coding, and testing. To address the complexity and potential logic inconsistencies in this process, state-of-the-art frameworks like ChatDev (Qian et al., 2024) and MetaGPT (Hong et al., 2024) leverage multi-agent collaboration, simulating human development teams to decompose complex tasks into streamlined and verifiable workflows.

Automated software maintenance. Issue resolution is intrinsically linked to the broader domain of automated software maintenance. Methodologies established in this field are frequently encapsulated as callable tools to augment the capabilities of LLMs in software development tasks. Key techniques utilized to enhance LLM performance include fault localization approaches such as SBFL (Wong et al., 2016; Zhou et al., 2012; Qi et al., 2022; Hu et al., 2014), code search (Zhang et al., 2025g), and test generation (Ahmed et al., 2024; Mündler et al., 2024). These tools provide agents with precise error locations, relevant code context, and verification mechanisms necessary for effective resolution. Furthermore, recent research has expanded the scope of automated software maintenance by utilizing issue resolution data to construct dialogue datasets that capture real-world human-computer collaboration (Garg et al., 2025).

Related surveys. Existing surveys primarily focus on code generation (Jiang et al., 2024; Wang et al., 2025b) or other tasks within the software engineering domain (Huang et al., 2023; Zhang et al., 2023; Guo et al., 2025a; Wang et al., 2025i). This paper bridges this gap by offering the first systematic survey dedicated to the entire spectrum of Issue Resolution, ranging from non-agent approaches to the latest agentic advancements.

A.2 Detailed discussions on background

The Issue Resolution task aims to automatically resolve reported issues. As illustrated in Figure 1 , it is formally defined by an instance represented as a triple $\mathcal{I} = (\mathcal{D}, \mathcal{C}, \mathcal{T})$, where these components map directly to the benchmark's metadata structure:

- $\mathcal{D}$: Issue description, which is the original GitHub issue text description detailing the bug or feature request.
- $\mathcal{C}$: Codebase, the collection of source files at the specific commit state before the issue was resolved. This state is precisely defined by the repository identifier (owner/repo) and a base commit hash, requiring the evaluation environment to perform a git clone and subsequent git checkout to establish the exact pre-fix version.
- $\mathcal{T}$: Tests, the complete set of unit and system tests associated with the issue, derived from the original developer's test patch. This set is explicitly categorized into two subsets: $T_{\text{fail}\to\text{pass}}$, containing tests that fail on C and must pass after the patch (verifying the fix), and $T_{\text{pass}\to\text{pass}}$, containing tests that must pass both before and after (ensuring no regression).

Given only the inputs $(\mathcal{D}, \mathcal{C})$, the model proposes an edit to resolve the issue. The model's output is a Patch $\mathcal{P}$ (represented as a patch file). (The target solution is the gold patch provided in the dataset, which is not revealed to the model). The patch $\mathcal{P}$ must be successfully applied to $\mathcal{C}$ using a standard patch application utility. The codebase resulting from this application is $\mathcal{C}'$. Crucially, the tests $\mathcal{T}$ are not revealed to the model during this process.

Evaluation. To evaluate a proposed solution (patch $\mathcal{P}$), it is first applied to the original codebase $\mathcal{C}$ to obtain $\mathcal{C}'$. The repository's test suite $\mathcal{T}$ is then executed on $\mathcal{C}'$. The evaluation framework captures the output in a test log, which is subsequently processed by a log parser to determine the test results (e.g., pass or fail). Based on the parsed results, the solution is scored: it is considered successful if the patch applies correctly and all tests in $\mathcal{T}$ pass. The final benchmark metric is the Resolve Rate, defined as the percentage of tasks that are successfully resolved.

Data Construction. This process is structured into four main stages.

First, (1) *Repo Selection and Data Scraping* involves collecting a large set of PRs from popular, well-maintained open-source repositories (e.g., 12 popular Python repositories for the original SWE-bench).

Second, (2) *Attribute-based Filtering* narrows down the candidates, selecting only merged PRs that are documented to resolve a specific GitHub issue and that make modifications to the repository's test files (indicating that tests were contributed).

Third, (3) *Execution-based Filtering* is a critical stage that ensures tasks are reproducible and valid. To guarantee reliable environment construction, recent works increasingly leverage CI/CD configurations—specifically GitHub Actions workflows found under `.github/workflows/`—as the ground truth for dependency management. For instance, Multi-SWE-bench (Zan et al., 2025) extracts build steps from these workflows to create isolated Dockerized environments. Similarly, SWE-Sharp-Bench (Mhatre et al., 2025) confirms build viability by executing local GitHub Actions workflows, ensuring the repository successfully builds and passes tests at the latest commit. By filtering out instances that fail these automated installation or runtime checks, this stage establishes a stable foundation. Finally, it verifies the PR's corrective nature by executing the test suite before and after applying the patch, retaining only instances that demonstrate at least one clear Fail-to-Pass (F2P) test transition.

Finally, (4) *Manual Verification* is performed to ensure the quality and usability of the filtered tasks. This step often involves human inspection to check for the clarity of the problem description (issue), the self-contained nature of the task, and its overall suitability for the benchmark.

While this initial pipeline was effective for creating a static dataset (e.g., the 2,294 SWE-bench instances), its reliance on a complex, manual-heavy environment setup (especially in stages 3 and 4) and its susceptibility to data contamination limited its long-term scalability and dynamism. Consequently, subsequent research has focused on enhancing these data methods, leading to two major axes of improvement: Data Collection for building more dynamic benchmarks, and Data Synthesis for creating high-quality synthetic datasets for training large language models.

A.3 Detailed discussions on Data

Table 1 provides a comprehensive survey of the rapidly evolving domain of SWE-bench related datasets. While the original SWE-bench and its refined iterations (such as SWE-bench Lite and SWE-bench Verified) set the standard for Python-based issue resolution, recent works have significantly expanded the scope of evaluation along three main axes: language diversity, modality, and scale.

Multilingual Expansion. A major limitation of early benchmarks was their exclusive focus on Python. To address this, datasets such as SWE-bench-java, Multi-SWE-bench, SWE-bench Multilingual, and SWE-PolyBench have been introduced. These datasets extend evaluation capabilities to a wide array of popular languages including Java, C++, Go, Rust, JavaScript, and TypeScript, encompassing thousands of repositories to test the generalization ability of agents across different syntaxes and ecosystems.

Multimodality. Recognizing that modern software development often involves visual elements (particularly in frontend development), datasets like SWE-bench Multimodal, CodeV, and OmniGIRL have incorporated visual contexts. These benchmarks require agents to process not only code and text but also visual data, targeting languages like HTML, CSS, and TypeScript.

Scale and Training Resources. To support the training of more robust agents, several large-scale datasets have been curated. SWE-Smith, SWE-Fixer, and SWE-bench-extra dramatically increase the volume of data, offering up to 115k instances (in the case of SWE-Fixer). Notably, as indicated in the "Environment" column, the field is shifting towards execution-based validation; unlike earlier static datasets, the majority of recent benchmarks

Dataset	Language	Multimodal	Repos	Amount	Environment	Link
SWE-bench-train	Python	✗	37	19k	✗	
SWE-bench	Python	✗	12	2294	✓	
SWE-bench Lite	Python	✗	12	300	✓	
SWE-bench Verified	Python	✗	/	500	✓	
SWE-bench-java	Java	✗	19	1797	✓	
SWE-bench Multimodal	JS,TS,HTML,CSS	✓	17	619	✓	
SWE-bench-extra	Python	✗	2k	6.38k	✓	
Visual SWE-bench	Python	✓	11	133	✓	
SWE-Lancer	JS, TS	✗	/	1488	✓	
Multi-SWE-bench	Java, JS, TS, Go, Rust, C, C++	✗	76	4,723	✓	
R2E-Gym	Python	✗	10	8,135	✓	
SWE-PolyBench	Python, Java, JS, TS	✗	21	2110	✓	
Loc-Bench	Python	✗	/	560	✗	
SWE-smith	Python	✗	128	50k	✓	
SWE-bench Multilingual	C, C++, Go, Java, JS, TS, Rust, Python, Ruby, PHP	✗	42	300	✓	
SWE-Fixer	Python	✗	856	115406	✗	
OmniGIRL	Python, TS, Java, JS	✓	15	959	✓	
SWE-rebench	Python	✗	30,000	21,336	✓	
SWE-bench-Live	Python	✗	93	1319	✓	
SWE-Gym	Python	✗	11	2,438	✓	
SWE-Flow	Python	✗	74	18081	✓	
SWE-Factory	Python, Java, JS, TS	✗	12	430	✓	
SWE-Bench-CL	Python	✗	8	273	✓	
Skywork-SWE	Python	✗	2531	10169	✓	/
SWE-MERA	Python	✗	200	300	✓	
SWE-Perf	Python	✗	12	140	✓	
RepoForge	Python	✗	/	7.3k	✓	/
SWE-Mirror	Python, Rust, Go	✗	40	60k	✓	/
SWE-Bench Pro	Go, TS, Python	✗	41	1865	✓	
SWE-InfraBench	Python, TS	✗	/	100	✓	/
SWE-Sharp-Bench	C#	✗	17	150	✓	
SWE-fficiency	Python, Cython	✗	9	498	✓	
SWE-Compass	Python, JS, TS, Java, C, C++, Go, Rust, Kotlin, C#	✗	/	2000	✓	
SWE-bench++	Python, Go, TS, JS, Ruby, PHP, Java, Rust, C++, C#, C	✗	3,971	1,782	✓	
SWE-EVO	Python	✗	7	48	✓	

Table 1: A comprehensive survey and statistical overview of issue resolution datasets. We categorize these datasetsbased on programming language, modality support, source repositories, data scale (Amount), and the availability ofreproducible execution environments.

now provide reproducible dockerized environments to rigorously verify agent generated patches.

A.4 Detailed discussions on training methods

Table 3 provides a comprehensive overview of recent SFT-based approaches.

Table 4 categorizes specialized models that have undergone further alignment, predominantly via Reinforcement Learning. The table sorts models by parameter size, revealing that smaller, dense models (e.g., 7B-32B) can achieve competitive performance against larger baselines when optimized with domain-specific rewards. The trends observed in the Reward column align with the design principles detailed in Section 4.2.2. While sparse outcome rewards based on test verification constitute the predominant approach, the data reveals a growing integration of process rewards. These dense feedback signals prove critical for stabilizing the training of smaller models during long-horizon tasks, thereby addressing the sparse signal challenges associated with complex repository-level debugging. Additionally, the heterogeneity observed in the Training Scaffolds category indicates a tendency to deploy RL atop established agent frameworks to enhance decision-making policies.

Finally, Table 5 lists general-purpose foundation models evaluated on issue resolution (Zhan et al., 2025; Seed et al., 2025; DeepSeek-AI et al., 2025; Team et al., 2025a; Yang et al., 2025a; Zeng et al., 2025a; Agarwal et al., 2025; Chen et al., 2025a; Team et al., 2025b). In contrast to the specialized models above, these systems rely entirely on external inference scaffolds to bridge the gap between general reasoning and repository-level engineering. This comparison serves as a control group, highlighting the specific performance gains attributable to the SFT and RL pipelines described in Section 4.2.

Dataset	Language	Repos	Amount	Link
R2E-Gym	Python	10	3,321	
SWE-Gym	Python	11	491	
SWE-Synth	Python	11	3,018	
SWE-Fixer	Python	856	69,752	
SWE-Factory	Python	10	2,809	

Table 2: A survey of trajectory datasets used for agent training or analysis. We list the programming language, number of source repositories, and total trajectories for each dataset.

Model Name	Base Model	Size	Arch.	Training Scaffold	Res.(%)	Code	Data	Model
Devstral	Mistral Small 3	22B	Dense	OpenHands	46.8	/		
Co-PatcheR	Qwen2.5-Coder	14B	Dense	Agentless-mini, PatchPilot	46.0		/	
SWE-Swiss-32B	Qwen2.5-32B-Instruct	32B	Dense	Agentless	45.0			
Lingma SWE-GPT	Qwen2.5-Instruct/Coder	72B/7B	Dense	SWESynInfer	30.2		/	/
SWE-Gym-Qwen-32B	Qwen2.5-Coder-32B	32B	Dense	OpenHands, MoatlessTools	20.6		/	
SWE-Gym-Qwen-14B	Qwen2.5-Coder-14B	14B	Dense	OpenHands, MoatlessTools	16.4		/	
SWE-Gym-Qwen-7B	Qwen2.5-Coder-7B	7B	Dense	OpenHands, MoatlessTools	10.6		/	

Table 3: Overview of SFT-based methods for issue resolution. This table categorizes models by their base architecture and training scaffold (Sorted by Performance).

A.5 Detailed discussions on applications

The evolution of AI in software engineering is characterized by four distinct stages of increasing autonomy and architectural sophistication:

Stage 1: Developer Augmentation. The initial phase is dominated by AI pair programmers, such as GitHub Copilot, which integrate directly into Integrated Development Environments (IDEs). These tools focus on real-time code completion and suggestions. Empirical evidence from enterprise adoption, such as at Accenture, indicates that these assistants can increase developer productivity by up to 55%, leading to widespread implementation at major technology firms like Shopify.

Stage 2: Workflow Automation. The subsequent level introduces AI junior developers, represented by tools like Sweep AI. These agents automate the asynchronous lifecycle of software maintenance. Unlike isolated code completions, they autonomously parse GitHub issues, analyze the relevant codebase, generate necessary modifications, and submit complete pull requests for human review, effectively handling routine engineering tasks without constant supervision.

Stage 3: End-to-End Autonomy. The most advanced operational stage involves fully autonomous agents like Devin. Functioning within secure sandboxed environments equipped with a shell, editor, and browser, these agents handle complex multi-step engineering tasks ranging from planning to execution. Their industrial impact is significant; for instance, during a major code migration at the fintech company Nubank, such autonomous agents reportedly achieved a 12x improvement in engineering efficiency compared to traditional methods.

Stage 4: Ecosystem Integration. The latest trend emphasizes platform interoperability and compliance. Emerging tools like Claude Code focus on embedding AI capabilities into existing enterprise workflows with rigorous security governance. Concurrently, platforms like Trae utilize the Model Context Protocol (MCP) to orchestrate multi-agent architectures, fostering an extensible ecosystem where diverse AI tools can collaborate seamlessly.

Model Name	Base Model	Size	Arch.	Train. Scaffold	Reward	Res.(%)	Code	Data	Model
560B Models (MoE)									
LongCat-Flash-Think	LongCatFlash-Base	560B-A27B	MoE	R2E-Gym	Outcome	60.4		/	
72B Models									
Kimi-Dev	Qwen 2.5-72B-Base	72B	Dense	BugFixer + TestWriter	Outcome	60.4		/	
Multi-turn RL(Nebius)	Qwen2.5-72B-Instruct	72B	Dense	SWE-agent	Outcome	39.0	/	/	/
Agent-RLVR-RM-72B	Qwen2.5-Coder-72B	72B	Dense	Localization + Repair	Outcome	27.8	/	/	/
Agent-RLVR-72B	Qwen2.5-Coder-72B	72B	Dense	Localization + Repair	Outcome	22.4	/	/	/
70B Models									
SWE-RL	Llama-3.3-70B-Instruct	70B	Dense	Agentless-mini	Outcome	41.0		/	/
36B Models									
FoldAgent	Seed-OSS-36B-Instruct	36B	Dense	FoldAgent	Process	58.0			/
32B Models									
OpenHands Critic	Qwen2.5-Coder-32B	32B	Dense	SWE-Gym	/	66.4		/	
KAT-Dev-32B	Qwen3-32B	32B	Dense	/	/	62.4	/	/	
SWE-Swiss-32B	Qwen2.5-32B-Instruct	32B	Dense	/	Outcome	60.2			
SeamlessFlow-32B	Qwen3-32B	32B	Dense	SWE-agent	Outcome	45.8		/	/
DeepSWE	Qwen3-32B	32B	Dense	R2E-Gym	Outcome	42.2			
SA-SWE-32B	/	32B	Dense	SkyRL-Agent	/	39.4	/	/	/
OpenHands LM v0.1	Qwen2.5-Coder-32B	32B	Dense	SWE-Gym	/	37.2		/	
SWE-Dev-32B	Qwen2.5-Coder-32B	32B	Dense	OpenHands	Outcome	36.6		/	
Satori-SWE	Qwen2.5-Coder-32B	32B	Dense	Retriever + Code editor	Outcome	35.8			
SoRFT-32B	Qwen2.5-Coder-32B	32B	Dense	Agentless	Outcome	30.8	/	/	/
Agent-RLVR-32B	Qwen2.5-Coder-32B	32B	Dense	Localization + Repair	Outcome	21.6	/	/	/
14B Models									
Agent-RLVR-14B	Qwen2.5-Coder-14B	14B	Dense	Localization + Repair	Outcome	18.0	/	/	/
SEAlign-14B	Qwen2.5-Coder-14B	14B	Dense	OpenHands	Process	17.7	/	/	/
9B Models									
SWE-Dev-9B	GLM-4-9B	9B	Dense	OpenHands	Outcome	13.6		/	
8B Models									
SeamlessFlow-8B	Qwen3-8B	8B	Dense	SWE-agent	Outcome	27.4		/	/
SWE-Dev-8B	Llama-3.1-8B	8B	Dense	OpenHands	Outcome	18.0		/	
7B Models									
SWE-Dev-7B	Qwen2.5-Coder-7B	7B	Dense	OpenHands	Outcome	23.4		/	
SoRFT-7B	Qwen2.5-Coder-7B	7B	Dense	Agentless	Outcome	21.4	/	/	/
SEAlign-7B	Qwen2.5-Coder-7B	7B	Dense	OpenHands	Process	15.0	/	/	/

Table 4: A comprehensive overview of specialized models for issue resolution, categorized by parameter size. The table details each model's base architecture, the training scaffold used for rollout, the type of reward signal employed (Outcome vs. Process), and their performance results (Res. %) on issue resolution benchmarks.

Model Name	Size	Arch.	Inf. Scaffold	Reward	Res.(%)	Code	Model
KAT-Coder	/	/	Claude Code	Outcome	73.4	/	
Deepseek V3.2	671B-A37B	MoE	Claude Code, RooCode	/	73.1		
Kimi-K2-Instruct	1T	MoE	Agentless	Outcome	71.6	/	
Qwen3-Coder	480B-A35B	MoE	OpenHands	Outcome	69.6		
GLM-4.6	355B-A32B	MoE	OpenHands	Outcome	68.0	/	
gpt-oss-120b	116.8B-A5.1B	MoE	Internal tool	Outcome	62.0		
Minimax M2	230B-10B	MoE	R2E-Gym	Outcome	61.0		
gpt-oss-20b	20.9B-A3.6B	MoE	Internal tool	Outcome	60.0		
GLM-4.5-Air	106B-A12B	MoE	OpenHands	Outcome	57.6	/	/
Minimax M1-80k	456B-A45.9B	MoE	Agentless	Outcome	56.0		
Minimax M1-40k	456B-A45.9B	MoE	Agentless	Outcome	55.6		
Seed1.5-Thinking	200B-A20B	MoE	/	Outcome	47.0		/
Llama 4 Maverick	400B-A17B	MoE	mini-SWE-agent	Outcome	21.0		
Llama 4 Scout	109B-17B	MoE	mini-SWE-agent	Outcome	9.1		

Table 5: Overview of general foundation models evaluated on issue resolution. The table details the specific inference scaffolds (e.g., OpenHands, Agentless) employed during the evaluation process to achieve the reported results.