

SWE-CRITIX: Post-Training Generalist Verifiers for Assessing Coding-Agent Trajectories

Jiajun Hu
Leibniz Research Center
Huawei
Hong Kong SAR, China
hu.jiajun@huawei.com

Chun Yong Chong
School of Information Technology
Monash University Malaysia
Subang Jaya, Malaysia
chong.chunyong@monash.edu

Abstract—Coding agents are increasingly deployed for end-to-end software engineering tasks, where a readily available oracle evaluator that defines the success criteria is often absent. This motivates verifications for coding agents that can reliably judge and explain their behavior. However, verification is non-trivial as coding agents are increasingly capable of solving long-horizon software engineering tasks. Their execution trajectories can be extremely long, containing hundreds of iterative interactions with code repositories and external tools. Existing verifiers either rely on general-purpose LLMs without specialized training or train dedicated critics that predominantly focus on producing scalar rewards, highlighting the relatively limited exploration of training-based generalist verification. We introduce SWE-CRITIX, a post-training pipeline for training generalist verifiers of coding-agent trajectories. Given a task and its execution trajectory, the learned verifier predicts the completion label and generates a natural-language analysis that explains its judgment. Our pipeline combines supervised fine-tuning (SFT) with reinforcement learning (RL) and trains a series of 30B models on trajectories from CodeForge-Preview. On a held-out test set, SWE-CRITIX improves classification accuracy from 57% for the base Qwen3-30B-A3B reasoning model to 66% after SFT and 78% after RL. We release our models, datasets, and training code to facilitate further research.

I. INTRODUCTION

Recent advances in large language models (LLMs) have enabled a shift from code generation toward autonomous coding agents that can interact with environments to solve complex software engineering tasks. Rather than producing one-shot code snippets, coding agents can inspect repositories, edit files, execute commands, run tests, interpret feedback, and iteratively refine their solutions through interactions with external tools [1]–[3]. This agentic paradigm has substantially expanded the scope of automated programming from isolated function synthesis to repository-level issue resolution [4] and long-horizon software development tasks [5], [6].

As coding agents become more capable and autonomous, reliably verifying their execution trajectories (i.e., step-by-step record of an agent solving a task) becomes increasingly important. In realistic end-to-end tasks, the criteria for successful completion are often not explicitly specified by a readily available ground-truth oracle. A verifier is therefore needed to determine whether an agent has accomplished the task and, when it fails, identify the causes of failure. However, such verification is challenging because coding-agent trajectories

TABLE I
A 2D TAXONOMY OF CODING-AGENT VERIFIERS.

	Outcome	Process
Training-free	Agentless [7]; LLM-as-a-Verifier [8]; Trae [9]; Agentic Rubrics [10]	SWE-PRM [11]; Wink [12]; TrajDebug [13]; SWE-Prime [14]; ProcCtrlBench [15] RETRACE [16]; LivePlan [17]; AgentLens [18];
Training-based <i>Scalar</i>	SWE-Gym [19]; R2E-Gym [20]; DeepSWE [21]; SWE-RM [22]; Dockerless [23]; SWE-TRACE [24]	OpenHands Critic [25]; Zainullina et al. [26]; Critic Rubrics [27]
Training-based <i>Generalist</i>	Agentic Rubrics [10]; ExecCritic [28]; SWE-Review [29]	Gandhi et al. [30]; Scout [31]; SWE-Critic

can contain hundreds of interleaved user inputs, environment feedback, and model responses, making them substantially more difficult to inspect than a one-shot model response. For instance, trajectories for tasks in LoLBench [6] span 53–532 turns and contain millions of tokens on average.

To address the challenge, existing work have proposed different techniques from two orthogonal dimensions: verification *target* and *method*, as summarized in Table I. The former distinguishes whether a verifier evaluates only the agent’s final outcome (e.g., the submitted patch) or also examines the process to detect undesirable intermediate behaviors. The latter distinguishes whether a verifier is constructed with or without training. Training-free methods directly employ general-purpose LLMs or conventional rule-based judges, whereas training-based methods learn dedicated critics, which can be further characterized by whether they produce scalar rewards or richer natural-language assessments.

This taxonomy reveals that training-based, process-oriented generalist verification remains relatively underexplored. To fill the gap, we propose SWE-CRITIX (Figure 1), a post-training pipeline that combines supervised fine-tuning (SFT) and reinforcement learning (RL) to train a generalist trajectory verifier. Given a task and its execution trajectory, the learned verifier not only determines whether the task has been successfully completed but also justifies the rationale

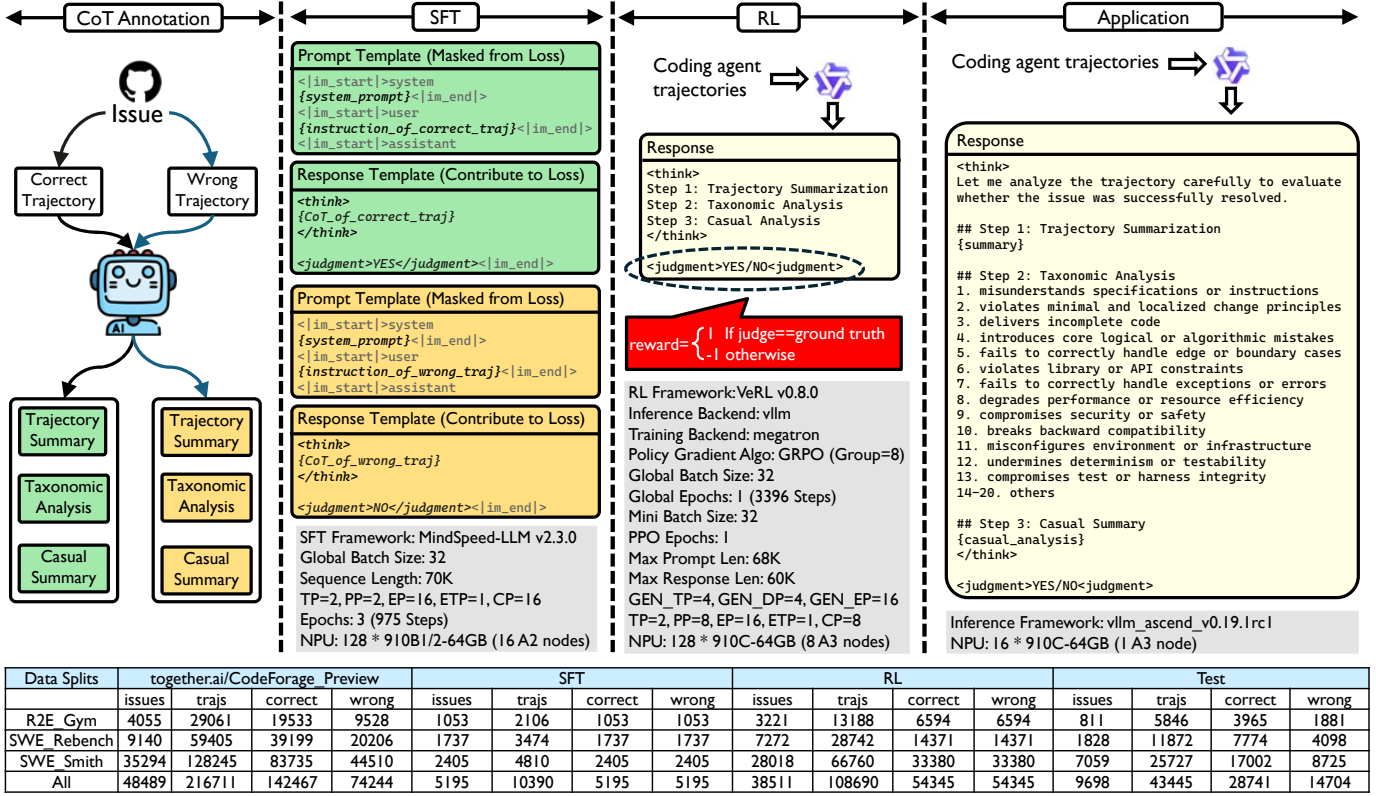


Fig. 1. SWE-CRITIX Post-Training Pipeline and Data Splits.

underlying its judgment. Compared with the two existing approaches in this category, SWE-CRITIX differs from them in terms of verification target (planning phase [30] vs. whole trajectory), annotation strategy (per-step dense labeling [31] vs. differential analysis), scale (2–8K training samples and 4–8B models [30], [31] vs. 119K and 30B), learning algorithms (DPO [30] vs. GRPO), and reward design (process reward [31] vs. outcome reward).

Using this pipeline, we train a Qwen3-30B MoE verifier on a distributed cluster equipped with Huawei Ascend NPUs, using coding-agent trajectories from CodeForge-Preview [32]. Our preliminary evaluation shows that the proposed post-training substantially improves the model’s ability to discriminate between successful and failed trajectories, increasing classification accuracy from 57% for the base model to 78% after reinforcement learning on the held-out test set. We open-source SWE-CRITIX on GitHub [33] and Hugging Face [34].

II. SWE-CRITIX

Figure 1 shows the post-training pipeline of SWE-CRITIX, which learns an LLM (i.e., a generalist verifier) for evaluating coding-agent trajectories. The verifier takes an issue-solving trajectory as input and determines whether the issue has been successfully resolved. Before making the binary judgment, the verifier produces a chain-of-thought (CoT) rationale that casually infers its decision (Figure 1-Application).

To train the verifier, SWE-CRITIX first applies a more capable teacher LLM to review a pair of successful and failed

trajectories that address the same issue. Through differential analysis, the teacher summarizes the respective procedures, mistake patterns, and key factors leading to the success or failure of each trajectory (Figure 1-CoT Annotation). These summaries are then incorporated into a small amount of SFT training samples, providing supervision for the base model to learn the reasoning process underlying trajectory evaluation (Figure 1-SFT). Finally, RL is applied to a substantial number of samples without CoT annotations, where unit-testing results are used as ground truth for computing the reward signal (Figure 1-RL).

A. Data Collection

We collect test-verified trajectories from CodeForge-Preview [32], a large open dataset containing 258K trajectories (155K pass vs. 103K fail) spanning 51K issues across 1,655 repositories. These trajectories were generated using the OpenHands scaffold [35] with Qwen3-Coder-480B [36] as the underlying model, on three downstream task sources [20], [37], [38] that provide executable gym environments. We further prune the dataset by discarding any trajectory that exceeds 64K tokens or does not contain a submitted patch (e.g., the task is incomplete). After filtering out 16% of the data, we split the remaining trajectories into SFT, RL, and test sets as follows:

1. We randomly choose 20% of the issues and use all associated trajectories as the test set.

2. Among the remaining issues, we randomly select one successful and one failed trajectory from each issue that contains both types, and assign them to the SFT set.
3. From the remaining trajectories, we include all failed trajectories in the RL set and then sample successful trajectories in a round-robin fashion across issues until the two classes are balanced. This maximizes the coverage of distinct issues.

The resulting data is released on Hugging Face [34] and its statistics is summarized in Figure 1-Data Splits.

B. Chain-of-Thought (CoT) Annotation

It has been widely acknowledged that generating a CoT before providing the final answer significantly improves the performance of LLMs to resolve complex tasks [39]–[41]. To bootstrap the reasoning capabilities of our target verifier, SWE-CRITIX leverages a teacher LLM (GPT-5.5) to generate a structured CoT trace by comparatively analyzing each pair of successful and failed trajectories in the SFT set. We adopt this pairwise comparison approach based on prior findings that LLMs perform better in pairwise evaluations than in individual judgments [42], [43]. We then post-process the generated CoTs to adapt them for pointwise judgment, which is the intended evaluation setting for the trained verifier. The prompt templates and more details are released on GitHub [33]. The resulting CoT trace of a trajectory includes a concise summary of its issue-resolving procedure, a characterization of mistake patterns based on a predefined taxonomy, and a causal summary of the factors contributing to the success or failure of the trajectory. We apply CoT annotation only to the SFT set, which contains 10,390 trajectories and accounts for 8.7% of the training set.

C. Supervised Fine-Tuning (SFT)

We assemble the annotated CoTs into SFT samples using the templates shown in Figure 1-SFT, where the green and orange boxes represent the templates applied on successful and failed trajectories, respectively. We retain the thinking template of Qwen, as we choose to fine-tune from Qwen3-30B-A3B-Thinking-2507 that already possesses some reasoning capabilities [36]. We minimize the negative log-likelihood of the responses over the SFT set, which is the standard SFT cross-entropy loss:

$$\mathcal{L}_{\text{SFT}}(\theta) = -\mathbb{E}_{(x,y) \sim \mathcal{D}_{\text{SFT}}} \left[\sum_{t=1}^{|y|} \log \pi_{\theta}(y_t \mid x, y_{<t}) \right] \quad (1)$$

where x denotes prompts and y denotes the corresponding response, including both the CoT and the final judgment.

We use MindSpeed-LLM [44] to fine-tune the Qwen3 MoE model on 128 Huawei Ascend 910B1/B2-64GB NPUs across 16 nodes. Training for three epochs (975 iterations) takes approximately 15 hours. The SFT loss curve is illustrated in Figure 2. Training code and hyperparameter settings are available on GitHub [33]. We denote the checkpoint at iteration 975 as π_{SFT} and open-source its weights on Hugging Face [34].



Fig. 2. Training dynamics of SWE-CRITIX. The upper plot shows the LM loss during SFT, while the lower plot shows the reward during RL training.

D. Reinforcement Learning (RL)

The SFT stage utilizes a small fraction of the training data with CoT annotations to help the model learn the reasoning patterns before making the final judgment. To further enhance the model’s reasoning ability, we apply RL, a more powerful learning paradigm [45]–[47], to the remaining 91.3% of the training data, which contain unit-testing results but no CoT annotations. We follow the standard policy gradient algorithms to maximize the expected rewards obtained by a policy π_{θ} :

$$\max_{\pi_{\theta}} \mathbb{E}_{x \sim \mathcal{D}_{\text{RL}}, y \sim \pi_{\theta}(\cdot|x)} [r(x, y)] - \beta \mathbb{D}_{\text{KL}} [\pi_{\theta} \parallel \pi_{\text{SFT}}] \quad (2)$$

where the regularization term $\mathbb{D}_{\text{KL}}$ prevents π_{θ} from deviating too far from π_{SFT} .

We define a simple yet effective rule-based outcome reward:

$$r(x, y) = \begin{cases} +1, & \text{if } y_{\text{ans}} = \text{YES} \wedge x_{\text{traj}} \text{ passes tests} \\ +1, & \text{if } y_{\text{ans}} = \text{NO} \wedge x_{\text{traj}} \text{ fails tests} \\ -1, & \text{otherwise.} \end{cases} \quad (3)$$

where x_{traj} denotes the coding-agent trajectory in prompt x and y_{ans} denotes the final answer extracted from the response y . A response receives a reward of +1 if its predicted judgment

TABLE II
PERFORMANCE COMPARISON OF QWEN3-30B-A3B, SWE-CRITIX SFT, AND SWE-CRITIX RL ON THE TEST SET AND ITS THREE SUBSETS.

Dataset	Model	Accuracy	Precision+	Recall+	F1+	Precision-	Recall-	F1-
R2E-Gym	Qwen3-30B-A3B	0.60 (3495/5846)	0.68 (3483/5128)	0.88 (3483/3965)	0.77	0.80 (12/15)	0.01 (12/1881)	0.01
	SWE-Critix SFT	0.52 (3027/5846)	0.73 (1817/2488)	0.46 (1817/3965)	0.56	0.36 (1210/3358)	0.64 (1210/1881)	0.46
	E-Critix RL	0.62 (3646/5846)	0.75 (2671/3577)	0.67 (2671/3965)	0.71	0.43 (975/2269)	0.52 (975/1881)	0.47
SWE-Rebench	Qwen3-30B-A3B	0.56 (6675/11872)	0.65 (6656/10170)	0.86 (6656/7774)	0.74	0.68 (19/28)	0.00 (19/4098)	0.01
	SWE-Critix SFT	0.71 (8476/11872)	0.84 (5369/6360)	0.69 (5369/7774)	0.76	0.56 (3107/5512)	0.76 (3107/4098)	0.65
	E-Critix RL	0.78 (9308/11872)	0.87 (6158/7106)	0.79 (6158/7774)	0.83	0.66 (3150/4766)	0.77 (3150/4098)	0.71
SWE-Smith	Qwen3-30B-A3B	0.56 (14417/25727)	0.66 (14397/21793)	0.85 (14397/17002)	0.74	0.80 (20/25)	0.00 (20/8725)	0.00
	SWE-Critix SFT	0.66 (17018/25727)	0.76 (12154/16015)	0.71 (12154/17002)	0.74	0.50 (4864/9712)	0.56 (4864/8725)	0.53
	E-Critix RL	0.82 (20971/25727)	0.82 (15670/19094)	0.92 (15670/17002)	0.87	0.80 (5301/6633)	0.61 (5301/8725)	0.69
All	Qwen3-30B-A3B	0.57 (24587/43445)	0.66 (24536/37091)	0.85 (24536/28741)	0.75	0.75 (51/68)	0.00 (51/14704)	0.01
	SWE-Critix SFT	0.66 (28521/43445)	0.78 (19340/24863)	0.67 (19340/28741)	0.72	0.49 (9181/18582)	0.62 (9181/14704)	0.55
	E-Critix RL	0.78 (33925/43445)	0.82 (24499/29777)	0.85 (24499/28741)	0.84	0.69 (9426/13668)	0.64 (9426/14704)	0.66

is consistent with the ground truth derived from unit testing, and -1 otherwise.

We use π_{SFT} as the initial RL checkpoint and conduct RL training using 128 Huawei Ascend 910C-64GB NPUs across 8 nodes with VeRL [48]. We use GRPO [46] with group size as 8 to optimize the objective in Equation 2. One epoch consists of 3,396 iterations and takes approximately 533 hours to complete. The reward curve during training is shown in Figure 2. We refer to the resulting checkpoint as π_{RL} and release its weights on Hugging Face [34].

III. PRELIMINARY EVALUATION

As a preliminary evaluation, we focus on the model’s ability to distinguish between successful and failed trajectories. We evaluate π_{RL} , π_{SFT} , and the base Qwen3 MoE model on the test split created in Section II-A. We serve these models using vLLM Ascend [49] on one node containing 16 Huawei Ascend 910C-64GB NPUs. Given the large scale of the test set (over 43K long-trajectory samples), we evaluate each model on each sample only once. We report *accuracy*, as well as *precision*, *recall*, and *F1* scores for both the positive (successful trajectory) and negative (failed trajectory) classes.

We present the results in Table II. The results reveal a clear progression from general-purpose language modeling to specialized trajectory verification. The base Qwen3 MoE model exhibits a strong bias toward predicting trajectories as correct, as its negative recall is nearly zero across all subsets. It also has limited instruction following capability on this task. We observed that 9.6% of its responses do not adhere to the required output format, preventing reliable extraction of the predicted judgments. SFT substantially alleviates both issues. All responses produced by π_{SFT} strictly follow the instructions required by the prompt, and the model becomes considerably less biased toward positive predictions. Further RL training yields another significant improvement, increasing accuracy from 66% after SFT to 78% on the overall test set. These results suggest that SFT primarily addresses the base model’s strong positive bias, while subsequent RL further strengthens

its discriminative capability by recovering positive recall from 67% to 85% without substantially sacrificing negative recall.

Despite these gains, two limitations remain. First, performance on R2E-Gym is notably lower than those on SWE-Rebench and SWE-Smith. One possible explanation is its relatively smaller data volume. Second, π_{RL} remains less effective at recognizing failed trajectories as the negative recall reaches only 64% overall. This suggests that failed trajectories remain more challenging to distinguish from successful ones, potentially due to the greater diversity of failure patterns and the subtle failure signals that are difficult to identify.

IV. FUTURE WORK

Our current evaluation focuses on the discriminative ability of SWE-CRITIX. In future, we plan to conduct more comprehensive evaluations along the following five dimensions.

1. Training Data: All samples in CodeForge-Preview [32] are generated using OpenHands [35] with Qwen3-Coder-480B [36]. To investigate the impact of data mixtures, we plan to incorporate trajectories from other sources, such as OpenSWE-Traces [50], and synthesize trajectories using different combinations of LLMs and harnesses. Furthermore, beyond the balanced positive-to-negative ratio, we plan to investigate the effects of different class ratios, such as 2:1, 1:2, and 4:1.

2. Training Methods: It remains an open question whether RL alone is sufficient to train an effective verifier. We therefore plan to establish a cold-start RL baseline by removing the CoT annotation and SFT stages from our pipeline. In addition, we plan to conduct reward ablations in the RL stage by incorporating signals not only from the final test results but also from the intermediate reasoning process.

3. Downstream Applications: We plan to explore downstream scenarios that can benefit from the trained verifier. For example, can the verifier help recover a correct solution by identifying and rectifying errors in a failed trajectory? Can the verifier serve as a proxy reward for RL training when ground-truth test results are unavailable?

4. Baselines: We plan to compare SWE-CRITIX with frontier open/closed-source models [51]–[53] and representative methods from the categories summarized in Table I.

5. Qualitative Analysis: Beyond the quantitative evaluations described above, we plan to conduct a qualitative analysis of the model’s reasoning traces to assess whether its analyses are logically coherent and grounded in evidence. We will also investigate cases in which the verifier makes incorrect judgments to better understand the sources of its errors.

V. RELATED WORK

Prior work proposing verifiers for reviewing coding-agent executions can be classified into two major categories depending on whether a specialized verification model is trained. One is training-free verification, where an evaluator agent uses general-purpose LLMs to assess coding agent executions through test generation [7], [9], rubric-based grading [10]–[12], verification scaling [8], metamorphic relations [16], failure attribution [13], or anti-pattern detection [14], [15], [17], [18]. Another major category is training-based verification, where a specialized critic is trained to evaluate trajectories using different learning paradigms such as TD learning [25], [26], autoregressive next-token prediction [10], [19]–[23], [28]–[31], multi-label classification [27], DPO [30], Bradley-Terry [24], and policy gradient [28], [31].

VI. CONCLUSION

In this paper, we proposed a post-training pipeline SWE-CRITIX that integrates SFT with RL to learn a process-level generalist verifier for assessing coding-agent trajectories. Preliminary experiments on coding-agent trajectories from CodeForge-Preview demonstrate a substantial improvement in the model’s discriminative ability.

REFERENCES

- [1] DeepSeek-AI, “Deepseek harness: Everything is a plugin,” <https://github.com/deepseek-ai/deepseek-harness>, 2026.
- [2] Codex, <https://openai.com/codex/>, 2026.
- [3] Claude Code, <https://claude.com/product/claude-code>, 2026.
- [4] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. R. Narasimhan, “SWE-bench: Can language models resolve real-world github issues?” in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: <https://openreview.net/forum?id=VTF8yNQM66>
- [5] X. Deng, J. Da, E. Pan, Y. Y. He, C. Ide, K. Garg, N. Lauffer, A. Park, N. Pasari, C. Rane, K. Sampath, M. Krishnan, S. Kundurthy, S. Hendryx, Z. Wang, V. Bharadwaj, J. Holm, R. Aluri, C. B. C. Zhang, N. Jacobson, B. Liu, and B. Kenstler, “Swe-bench pro: Can ai agents solve long-horizon software engineering tasks?” 2025. [Online]. Available: <https://arxiv.org/abs/2509.16941>
- [6] Y. Peng, Z. Wu, Z. Zhuang, X. Zhou, R. Shu, X. Han, C. Y. Chong, Y. Wang, and J. Liu, “Lolbench: Evaluating coding agents with long-horizon proposals on large software systems,” 2026. [Online]. Available: <https://arxiv.org/abs/2609.37143>
- [7] C. S. Xia, Y. Deng, S. Dunn, and L. Zhang, “Agentless: Demystifying llm-based software engineering agents,” 2024. [Online]. Available: <https://arxiv.org/abs/2407.01489>
- [8] J. Kwok, S. Li, P. Atreya, Y. Liu, Y. Jiang, C. Finn, M. Pavone, I. Stoica, and A. Mirhoseini, “Llm-as-a-verifier: A general-purpose verification framework,” 2026. [Online]. Available: <https://arxiv.org/abs/2607.05391>
- [9] T. R. Team, P. Gao, Z. Tian, X. Meng, X. Wang, R. Hu, Y. Xiao, Y. Liu, Z. Zhang, J. Chen, C. Gao, Y. Lin, Y. Xiong, C. Peng, and X. Liu, “Trae agent: An llm-based agent for software engineering with test-time scaling,” 2025. [Online]. Available: <https://arxiv.org/abs/2507.23370>
- [10] M. Raghavendra, A. Gunjal, B. Liu, and Y. He, “Agentic rubrics as contextual verifiers for swe agents,” 2026. [Online]. Available: <https://arxiv.org/abs/2601.04171>
- [11] S. Gandhi, J. Tsay, J. Ganhotra, K. Kate, and Y. Rizk, “When agents go astray: Course-correcting swe agents with prms,” 2025. [Online]. Available: <https://arxiv.org/abs/2509.02360>
- [12] R. Nanda, C. Maddila, S. Jha, E. M. Khan, M. Paltenghi, and S. Chandra, “Wink: Recovering from misbehaviors in coding agents,” 2026. [Online]. Available: <https://arxiv.org/abs/2602.17037>
- [13] Y. Qi, Z. Yin, X. Shi, H. Peng, S. Lu, Y. Liu, R. Xuan, Y. Liu, Z. Hu, X. Wang, L. Hou, B. Xu, and J. Li, “Trajdebug: Tracing error lifecycle to identify critical failures in long-horizon agent trajectories,” 2026. [Online]. Available: <https://arxiv.org/abs/2608.06346>
- [14] D. Zheng, R. Ye, Y. Wang, Y. Ye, H. Zhang, E. Shi, X. Liu, Y. Ma, J. Yu, and Z. Zheng, “Swe-prime: Fewer trajectories, better performance,” 2026. [Online]. Available: <https://arxiv.org/abs/2608.27449>
- [15] J. He, J. Jia, C. Liu, C. Xue, Y. Song, X. Yang, and D. Sun, “Proccrlbench: Evaluating process-level defects and control preservation in llm coding agents,” 2026. [Online]. Available: <https://arxiv.org/abs/2605.20251>
- [16] C. Li, Y. Xu, Z. Wang, S. H. Tan, Tse-Hsun, and Chen, “Independent patch verification for coding agents with a bidirectional reconstruct-and-verify framework,” 2026. [Online]. Available: <https://arxiv.org/abs/2608.08950>
- [17] S. Liu, S. Dehghan, J. Y. Kim, J. Ganhotra, M. Hirzel, and R. Jabbarvand, “Online monitoring and corrective steering of programming agents,” 2026. [Online]. Available: <https://arxiv.org/abs/2608.06701>
- [18] P. Sahoo, G. Mittal, X. Li, S. Ma, B. Steenhoeck, P. Lin, and Y. Hu, “Agentlens: Revealing the lucky pass problem in swe-agent evaluation,” 2026. [Online]. Available: <https://arxiv.org/abs/2605.12925>
- [19] J. Pan, X. Wang, G. Neubig, N. Jaitly, H. Ji, A. Suhr, and Y. Zhang, “Training software engineering agents and verifiers with swe-gym,” 2025. [Online]. Available: <https://arxiv.org/abs/2412.21139>
- [20] N. Jain, J. Singh, M. Shetty, L. Zheng, K. Sen, and I. Stoica, “R2e-gym: Procedural environments and hybrid verifiers for scaling open-weights swe agents,” 2025. [Online]. Available: <https://arxiv.org/abs/2504.07164>
- [21] M. Luo, N. Jain, J. Singh, S. Tan, A. Patel, Q. Wu, A. Ariyak, C. Cai, T. Venkat, S. Zhu, B. Athiwaratkun, M. Roongta, C. Zhang, L. E. Li, R. A. Popa, K. Sen, and I. Stoica, “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-22281902c1468193aabb9a8c59bbe33>, 2025, notion Blog.
- [22] K. Shum, B. Hui, J. Chen, L. Zhang, X. W. J. Yang, Y. Huang, J. Lin, and J. He, “Swe-rm: Execution-free feedback for software engineering agents,” 2025. [Online]. Available: <https://arxiv.org/abs/2512.21919>
- [23] W. Zeng, Y. Shi, X. Gu, C. Hu, C. Wang, Y. Cui, H. Zhou, M. Qi, J. Wangni, Z. Yu, S. Gao, K. Cai, and S. He, “Dockerless: Environment-free program verifier for coding agents,” 2026. [Online]. Available: <https://arxiv.org/abs/2606.28436>
- [24] H. Han, J. Xie, X. Ma, W. Zhu, Z. Zhang, Z. Long, H. Chen, and Q. Ye, “Swe-trace: Optimizing long-horizon swe agents through rubric process reward models and heuristic test-time scaling,” 2026. [Online]. Available: <https://arxiv.org/abs/2604.14820>
- [25] X. Wang, (2025, 4) Sota on swe-bench verified with inference-time scaling and critic model. OpenHands Blog. Blog post. [Online]. Available: <https://www.openhands.dev/blog/sota-on-swe-bench-verified-with-inference-time-scaling-and-critic-model>
- [26] K. Zainullina, A. Golubev, M. Trofimova, S. Polezhaev, I. Badertdinov, D. Litvintseva, S. Karasik, F. Fisin, S. Skvortsov, M. Nekrashevich, A. Shevtsov, and B. Yangel, “Guided search strategies in non-serializable environments with applications to software engineering agents,” 2025. [Online]. Available: <https://arxiv.org/abs/2505.13652>
- [27] X. Wang, V. Chen, H. Ji, and G. Neubig, “A rubric-supervised critic from sparse real-world outcomes,” 2026. [Online]. Available: <https://arxiv.org/abs/2603.03800>
- [28] L. Tao, B. Peng, H. Wang, H. Cheng, W. Yao, Q. Wu, T. Ge, S. Li, and J. Gao, “Execritic: Learn to test, test to improve for coding agents,” 2026. [Online]. Available: <https://arxiv.org/abs/2609.09133>
- [29] R. Wang, J. Chen, S. Wang, C. Tao, S. Yang, Y. Jiang, K.-H. Yap, L. Shang, X. Li, and H. Bai, “Swe-review: Closing the loop on issue resolution with agentic code review,” 2026. [Online]. Available: <https://arxiv.org/abs/2607.06065>

- [30] S. Gandhi, Y. Xie, A. Naik, R. Zhu, and C. Rose, “Steer, don’t solve: Training small critic models for large code agents,” 2026. [Online]. Available: <https://arxiv.org/abs/2606.21811>
- [31] S. Rahman, Y. Kim, M. Parmar, A. A. Heydari, G. Liu, S. A. Lee, W. Zhang, A. Hosseini, A. A. Metwally, Y. Yang, B. Mirzasoleiman, X. Liu, P. Izmailov, S. Gabriel, M. Malhotra, S. Patel, D. McDuff, and H. Palangi, “Locating hidden failures makes long-horizon agents more reliable,” 2026. [Online]. Available: <https://arxiv.org/abs/2609.17930>
- [32] A. Ariyak, J. Zhang, J. Wang, S. Zhu, F. Bianchi, S. Srivastava, A. Panda, S. Bharti, C. Xu, J. Heo, X. S. Wu, J. Zou, P. Liang, L. Song, C. Zhang, B. Athiwaratkun, Z. Zhou, and Q. Wu, “Coderforge-preview: Sota open dataset for training efficient agents,” Feb. 2026, project core leads: Alpay Ariyak; Zhongzhu Zhou; Qingyang Wu. [Online]. Available: <https://www.together.ai/blog/coderforge-preview>
- [33] SWE-Critix GitHub, <https://github.com/SWE-Critix/SWE-Critix>, 2026.
- [34] SWE-Critix Hugging Face, <https://huggingface.co/SWE-Critix>, 2026.
- [35] X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, Y. Song, B. Li, J. Singh, H. H. Tran, F. Li, R. Ma, M. Zheng, B. Qian, Y. Shao, N. Muennighoff, Y. Zhang, B. Hui, J. Lin, R. Brennan, H. Peng, H. Ji, and G. Neubig, “Openhands: An open platform for ai software developers as generalist agents,” 2025. [Online]. Available: <https://arxiv.org/abs/2407.16741>
- [36] Q. Team, “Qwen3 technical report,” 2025. [Online]. Available: <https://arxiv.org/abs/2505.09388>
- [37] I. Badertdinov, A. Golubev, M. Nekrashevich, A. Shevtsov, S. Karasik, A. Andriushchenko, M. Trofimova, D. Litvintseva, and B. Yangel, “Swe-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents,” 2025. [Online]. Available: <https://arxiv.org/abs/2505.20411>
- [38] J. Yang, K. Lieret, C. E. Jimenez, A. Wettig, K. Khandpur, Y. Zhang, B. Hui, O. Press, L. Schmidt, and D. Yang, “Swe-smith: Scaling data for software engineering agents,” 2025. [Online]. Available: <https://arxiv.org/abs/2504.21798>
- [39] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou, “Chain-of-thought prompting elicits reasoning in large language models,” 2023. [Online]. Available: <https://arxiv.org/abs/2201.11903>
- [40] M. Y. Guan, M. Joglekar, E. Wallace, S. Jain, B. Barak, A. Helyar, R. Dias, A. Vallone, H. Ren, J. Wei, H. W. Chung, S. Toyer, J. Heidecke, A. Beutel, and A. Glaese, “Deliberative alignment: Reasoning enables safer language models,” 2025. [Online]. Available: <https://arxiv.org/abs/2412.16339>
- [41] X. Chen, G. Li, Z. Wang, B. Jin, C. Qian, Y. Wang, H. Wang, Y. Zhang, D. Zhang, T. Zhang, H. Tong, and H. Ji, “Rm-r1: Reward modeling as reasoning,” 2026. [Online]. Available: <https://arxiv.org/abs/2505.02387>
- [42] A. Wettig, A. Gupta, S. Malik, and D. Chen, “Qrating: Selecting high-quality data for training language models,” 2024. [Online]. Available: <https://arxiv.org/abs/2402.09739>
- [43] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, “Training language models to follow instructions with human feedback,” 2022. [Online]. Available: <https://arxiv.org/abs/2203.02155>
- [44] MindSpeed-LLM, <https://gitcode.com/Ascend/MindSpeed-LLM>, 2026.
- [45] DeepSeek-AI, “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” 2025. [Online]. Available: <https://arxiv.org/abs/2501.12948>
- [46] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo, “Deepseekmath: Pushing the limits of mathematical reasoning in open language models,” 2024. [Online]. Available: <https://arxiv.org/abs/2402.03300>
- [47] N. Lambert, “Reinforcement learning from human feedback,” 2025. [Online]. Available: <https://arxiv.org/abs/2504.12501>
- [48] G. Sheng, C. Zhang, Z. Ye, X. Wu, W. Zhang, R. Zhang, Y. Peng, H. Lin, and C. Wu, “Hybridflow: A flexible and efficient rlhf framework,” in *Proceedings of the Twentieth European Conference on Computer Systems*, ser. EuroSys ’25. ACM, Mar. 2025, p. 1279–1297. [Online]. Available: <http://dx.doi.org/10.1145/3689031.3696075>
- [49] vLLM Ascend, <https://github.com/vllm-project/vllm-ascend>, 2026.
- [50] W. U. Ahmad, N. Ludwig, S. Majumdar, and B. Ginsburg, “Open-swe-traces: Advancing dual-mode multilingual distillation for software engineering agents,” 2026. [Online]. Available: <https://arxiv.org/abs/2606.16038>
- [51] DeepSeek-AI, “Deepseek-v4.1-flash: Pushing the limits of kv cache compression,” 2026.
- [52] GLM-5-Team, “Glm-5: from vibe coding to agentic engineering,” 2026. [Online]. Available: <https://arxiv.org/abs/2602.15763>
- [53] GPT-6 Astra, <https://openai.com/index/gpt-6-astra/>, 2026.