EvoConfig: Self-Evolving Multi-Agent Systems for Efficient Autonomous Environment Configuration

Anonymous ACL submission

Abstract

A reliable executable environment is the foundation for ensuring that large language models solve software engineering tasks. Due to the complex and tedious construction process, large-scale configuration is relatively inefficient. However, most methods always overlook fine-grained analysis of the actions performed by the agent, making it difficult to handle complex errors and resulting in configuration failures. To address this bottleneck, we propose EvoConfig, an efficient environment configuration framework that optimizes multi-agent collaboration to build correct runtime environments. EvoConfig features an expert diagnosis module for fine-grained post-execution analysis, and a self-evolving mechanism that lets expert agents self-feedback and dynamically adjust error-fixing priorities in real time. Empirically, EvoConfig matches the previous state-of-the-art Repo2Run on Repo2Run's 420 repositories, while delivering clear gains on harder cases: on the more challenging Envbench, EvoConfig achieves a 78.1% success rate, outperforming Repo2Run by 7.1%. Beyond end-to-end success, EvoConfig also demonstrates stronger debugging competence, achieving higher accuracy in error identification and producing more effective repair recommendations than existing methods [1].

1 Introduction

Large language models (LLMs) have made rapid progress in handling complex software engineering (SWE) tasks (He et al., 2025; Wang et al., 2025c; Xia et al., 2025; Wang et al., 2024a; Pandit et al., 2025; Yang et al., 2025b), leading to the emergence of a wide range of code agents such as SWE-Agent (Yang et al., 2024), OpenHands (Wang et al., 2024b), MetaGPT (Hong et al., 2023), Copilot (GitHub, 2021) and Cursor (Anysphere, 2023).

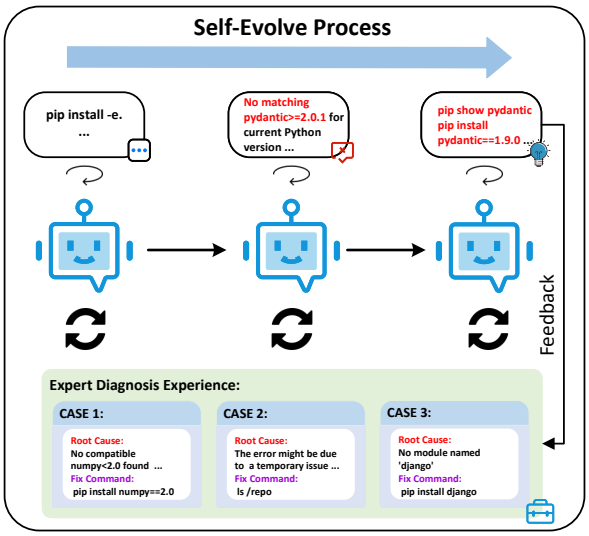

Figure 1: Self-Evolving Diagnostic Process.

As research increasingly shifts toward repository-level software engineering tasks, scalable execution and reliable validation become essential (Wang et al., 2025a; Xie et al., 2024; Liu et al., 2024; Rutherford et al., 2024; Krnjaic et al., 2024). Code agents are no longer required to only generate or modify code, but must also complete end-to-end workflows, including environment construction, testing, and validation, within real code runtime environments. However, a long-overlooked challenge is now becoming increasingly evident: automatically configuring executable environments. Environment setup still depends heavily on human expertise and can be difficult even for experienced developers. Yet a stable, runnable environment is a prerequisite for tackling complex software engineering tasks. *Therefore, enabling agents to reliably configure environments is critical to advancing code agents.*

In real-world repositories, an agent must autonomously complete dependency installation, version resolution, and test execution under challeng-

[1] We will open-source the code after the paper is published.

ing conditions, including unknown dependencies, incomplete documentation, and the coexistence of multiple build tools. Most existing methods formulate environment configuration as a sequential decision-making problem (Bouzenia and Pradel, 2025a; Vergopoulos et al., 2025; Hu et al., 2025; Zhang et al., 2025a): the agent observes the current execution outcome and heuristically proposes the next action. However, **these approaches often fail to explicitly address process-level errors that occur during configuration** (e.g., cascading dependency conflicts, toolchain mismatches, partial installations). Such errors can accumulate across steps and ultimately cause environment construction to fail. Some prior work (Milliken et al., 2025a; Vergopoulos et al., 2025) introduces repair strategies that are detached from the original configuration context. These methods typically rely on predefined, experience-based rules to produce static repair actions, **but they lack fine-grained diagnosis of the specific failure causes in the ongoing configuration process**. Consequently, agents are more prone to hallucinated fixes or repetitive trial-and-error behaviors, and may even fall into infinite loops when confronted with complex failures. In addition, the single-agent workflow exacerbates the problem: error-related information and noisy execution traces accumulate over time, which can mislead subsequent decisions and further reduce both success rate and efficiency.

To address these challenges, we propose EvoConfig, an efficient environment configuration framework based on self-evolving multi-agent collaboration. **Our core objective is to improve environment configuration success rates while simultaneously enhancing process-level error correction capabilities during the configuration process**. Specifically, a main agent is responsible for environment configuration, while expert agents act as diagnostic specialists that perform fine-grained analysis of execution results and autonomously determine whether repairs are required, ultimately providing structured and actionable guidance to the main agent. More importantly, we introduce an online self-evolving mechanism that enables expert agents to continuously learn from error correction cases and dynamically adjust their analytical focus and structured suggestions, thereby improving the agent's ability to resolve complex environment configuration failures. Notably, this self-evolving mechanism does not rely on external memory modules, avoiding additional reasoning overhead and token consumption. We evaluate EvoConfig on multiple real-world open-source repositories, and the results demonstrate that our approach not only improves environment configuration success rates but also significantly enhances process-level error correction during the configuration process.

In summary, our main contributions are summarized as follows:

- We are the first to propose the multi-agent collaborative framework EvoConfig for automated environment configuration, improving configuration success rates through optimized agent workflows.
- We propose an expert diagnostic module and introduce a self-evolving mechanism to adaptively enhance the process-level error correction capability of agents in the environment configuration process.
- We conduct extensive evaluations on multiple open-source benchmarks against advanced agent frameworks, demonstrating that EvoConfig achieves state-of-the-art performance in both environment configuration and process-level error correction.

2 Formulation

2.1 Task Definition

Given a real-world open-source GitHub repository R at a specified version, the system is provided with a clean initial execution environment E_0. The ultimate goal is to automatically construct a target execution environment E through a sequence of interactive commands, such that unit tests can be successfully executed in the resulting environment.

2.2 Iterative Configuration Process

In this work, we model environment configuration as an interactive decision-making process. Specifically, at interaction round t, the agent is in the current environment state E_t and selects a set of commands from the action space $\mathcal{A}$ for execution:

$$a_t = \{c_t^1, c_t^2, \ldots, c_t^{k_t}\}, \quad a_t \subseteq \mathcal{A}, \qquad (1)$$

where each c_t^i denotes an atomic executable command, and k_t is the number of commands issued at round t.

After executing the command set a_t, the system performs a state transition based on the current

environment state and the execution outcomes:

$$E_{t+1} = \delta(E_t, a_t). \tag{2}$$

This process is repeated for at most $t_{\max}$ interaction rounds, until the test cases are successfully executed or the number of interactions exceeds a predefined maximum threshold. In this work, we place particular emphasis on whether the commands generated at each round result in execution errors. Accordingly, the overall optimization objective is to improve the environment configuration success rate under a limited interaction budget, while simultaneously enhancing the agent's capability for process-level error correction.

3 Method

This section introduces EvoConfig, a self-evolving multi-agent framework for efficient environment configuration. Given a code repository, EvoConfig performs multiple rounds of interaction and decision-making while continuously repairing environment configuration issues, ultimately generating an executable Dockerfile to build a runnable environment. EvoConfig consists of three main components: an environment information extraction module, a main environment configuration module, and an self-evolving expert diagnosis module.

3.1 Environment Info Extraction Module

We introduce a lightweight environment information extraction module that provides the main agent with a small set of high-impact prior signals before interactive configuration begins. The module focuses on extracting stable structural cues that are directly relevant to environment configuration.

Formally, given a repository R, the module produces a prior summary:

$$P(R) = \{M, I, T\}, \tag{3}$$

where M, I, and T denote the dependency management strategy, project importability, and test structure, respectively.

Dependency Management Strategy. The dependency management strategy M is inferred from configuration files such as `poetry.lock`, `pyproject.toml`, and `requirements*.txt`, guiding early installation decisions.

Project Importability. Project importability I captures whether the project needs to be installed for tests to run, based on installation metadata, `src/` layouts, and package structure.

Test Structure Hypothesis. The test structure hypothesis T describes the presence and location of tests, the inferred test framework, and whether tests import project modules.

The prior summary $P(R)$ is injected into the initial prompt words of the main agent to guide the generation of the initial configuration strategy of the main agent with almost no increase in computational cost.

3.2 Main Environment Configuration Module

After the environment prior information is extracted, the system enters the core environment configuration stage. Unlike previous approaches, the main agent responsible for environment configuration focuses solely on action execution and sequence, without bearing the burden of long-term memory and semantic analysis of execution results.

Specifically, at interaction step t, the main agent performs ReAct (Yao et al., 2022) framework reasoning based on a limited context and generates an action output, which is parsed into a sequence of atomic commands and executed sequentially in the runtime environment. Each command returns standard output and an exit code as execution feedback for the current step. During this process, the main agent concentrates only on action generation, scheduling, and execution order, and does not directly interpret the semantics of execution results. Instead, the execution context is delegated to the expert diagnosis module for analysis. This design allows the main agent to advance execution in a streaming manner, avoiding the accumulation of large volumes of raw output across multiple interaction rounds and receiving only highly summarized analytical feedback. As a result, it effectively mitigates a key issue in traditional interactive systems, where incorporating large amounts of low-value output directly into the main reasoning context leads to memory inflation and interferes with subsequent decision-making.

In addition, to maintain reasoning quality while reducing overall overhead, the main agent adopts a strict context management strategy. Combined with the system's rollback mechanism, it preserves key command sequences from successful execution rounds as well as structured diagnostic summaries from the diagnosis module as experience, thereby improving the efficiency of action generation and scheduling during environment configuration.

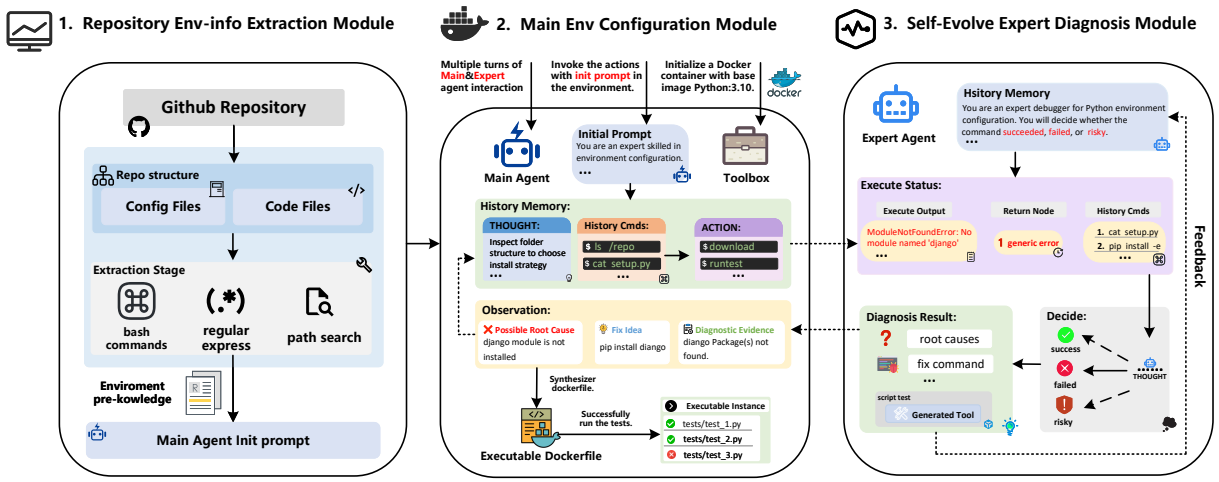

Figure 2: Workflow of EvoConfig. A **main configuration agent** performs interactive environment setup, while **self-evolving expert diagnostic agent** analyzes execution feedback and provide adaptive guidance. The validated command sequence is consolidated into a runnable Dockerfile.

3.3 Self-Evolving Expert Diagnosis Module

During environment configuration, accurate error diagnosis and repair are critical to ensuring successful system deployment. To enhance process-level error correction capability, we introduce an expert diagnosis module with a self-evolving mechanism, which explicitly decouples the standard output of execution actions from the primary configuration workflow and assigns it to an independent expert diagnostic agent.

The critical function of the expert agent is to assess the outcomes of execution actions and produce fine-grained analytical results. Specifically, based on the executed command, exit code, and standard output, the expert agent categorizes each action into one of three states—**success**, **failure**, or **potential risk**. According to the identified state, it generates corresponding repair commands or risk suggestions, and ultimately outputs a structured diagnostic report.

Notably, the expert agent is endowed with the capability of **on-the-live tool** creation and execution. At each decision step, the agent autonomously determines whether auxiliary tools are needed to support error judgment. Tool creation is subject to strict constraints: each tool must be a **single-line executable command** used solely for collecting diagnostic evidence rather than performing repairs. The outputs of these tools are treated as diagnostic evidence to strengthen error interpretation and are fed back to the main agent in a structured form.

Furthermore, we introduce the concept of a **self-evolving mechanism**. After each diagnostic cycle, the expert agent incrementally adjusts its internal rules based on feedback signals. These rules primarily govern repair suggestion generation, tool creation, and risk assessment. Through continuous evolution driven by historical experience, the expert agent progressively refines its decision-making process and becomes capable of handling increasingly complex configuration errors.

4 Experimental Setup

We evaluate EvoConfig from two complementary perspectives: environment build success and process-level error correction capability.

4.1 Environment Build Success Evaluation

Dataset and Baselines. We evaluate environment construction on 420 Repo2Run repositories (Hu et al., 2025) and 324 Python repositories from EnvBench (Eliseeva et al., 2025), excluding 5 EnvBench repositories larger than 200MB. All experiments follow the Repo2Run protocol and compare EvoConfig with **pipreqs** (bndr, 2016), **LLM Generator**, **SWE-agent** (Yang et al., 2024), and **Repo2Run** (Hu et al., 2025). We use `gpt-4o-2024-05-13`, `GPT-3.5-turbo`, and `GPT-4o-mini`, with a 2-hour time limit and up to 100 interaction rounds. Additional details are provided in Appendix A.

Evaluation Metrics. We use two metrics to evaluate environment construction. **DGSR** measures the percentage of attempts that generate a runnable Dockerfile that builds without errors, while **EBSR**

Method	Backbone	DGSR	# Successfully Generated Dockerfiles	EBSR	# Successfully Built Environments
pipreqs (bndr, 2016)	-	29.8%	125	6.0%	25
LLM generator (Hu et al., 2025)	GPT-4o	47.6%	200	22.1%	93
SWE-agent (Yang et al., 2024)	GPT-4o	26.9%	113	9.0%	38
Repo2Run (Hu et al., 2025)	GPT-4o	100%	420	86.0%	361
EvoConfig	GPT-4o	**100%**	**420**	**88.1%**	**370**

Table 1: Main results of different baselines in terms of Dockerfile generation and environment build success under the same backbone.

Method	Backbone	EBSR	# Successfully Built Environments
Repo2Run	GPT-3.5-turbo	71.0%	230
	GPT-4o-mini	40.0%	12
EvoConfig	GPT-3.5-turbo	**78.1%**	**253**
	GPT-4o-mini	**46.7%**	**14**

Table 2: Performance comparison under different backbone models on the 324 repositories from EnvBench. Results for `gpt-4o-mini` are obtained on a randomly sampled subset of 30 repositories.

measures the percentage of attempts that successfully build executable environments, requiring both a successful Dockerfile build and the ability to execute tests with `pytest`, regardless of test outcomes.

4.2 Process Error Correction Evaluation

Dataset and Baselines. For process-level error correction evaluation, we use the EnConda-Bench dataset (Kuang et al., 2025), which is designed to assess an agent's ability to diagnose errors and recover from failed configuration steps during interactive execution. We evaluate all 4,201 instances provided by EnConda-Bench and compare EvoConfig against representative baselines, including **SWE-Agent**, **OpenHands** (Wang et al., 2024b), **INSTALLAMATIC** (Milliken et al., 2025a), and **Repo2Run** (Hu et al., 2025), using `GPT-4.1` and `DeepSeek-V3` (Liu et al., 2023) as the underlying language models. More details about our selected baselines are provided in Appendix B.

Evaluation Metrics. We follow the evaluation protocol and metrics defined in EnConda-Bench, which measure an agent's capability from error perception to corrective execution. Specifically, the metrics include error classification precision and recall, error description accuracy and fix accuracy. Each agent interacts with the execution environment step by step, generates diagnostic feedback and repair actions upon failure, and is evaluated based on both the correctness of intermediate error handling and the final recovery outcome.

5 Result Analysis

5.1 Main Results

Environment Construction Success Analysis. The results of different baselines are presented in Table 1. Results of all baselines except EvoConfig are taken from the original Repo2Run benchmark to ensure a fair comparison.

We observe that EvoConfig achieves an environment building success rate that is comparable to, and slightly higher than, Repo2Run on the original set of 420 repositories. EvoConfig successfully builds executable environments for 370 repositories (EBSR 88.1%), compared to 361 repositories (EBSR 86.0%) built by Repo2Run. Given the already strong performance of Repo2Run, this improvement suggests that EvoConfig can recover a small but non-negligible fraction of failure cases that remain challenging for existing environment configuration agents. EvoConfig also maintains a DGSR of 100%, matching Repo2Run and confirming that robust rollback and verification mechanisms are preserved, while other baselines fail to consistently guarantee Dockerfile buildability.

Table 2 further presents environment building performance under different language model backbones. For `gpt-3.5-turbo`, EvoConfig improves EBSR from 71.0% to 78.1%, corresponding to 23 additional repositories successfully configured. We also present results using `gpt-4o-mini`, evaluated on a randomly sampled subset of 30 repositories due to computational constraints. This result indicates that the advantages of EvoConfig generalize across different model backbones.

Process-level Error Correction Analysis. We evaluate process-level error correction results on EnConda-Bench in Table 3. EvoConfig demon-

Method	Backbone	Perception			Feedback	Feedback and Action
		Error type			Error description	Fix suggestion
		Pre.	Rec.	F1	ACC.	ACC.
Code Agent						
SWE-Agent (Yang et al., 2024)	GPT-4.1	43.7	83.2	55.3	49.8	30.7
	DeepSeek-V3	41.2	70.3	51.9	44.5	27.8
OpenHands (Wang et al., 2024b)	GPT-4.1	42.5	72.0	53.2	46.0	29.1
	DeepSeek-V3	46.7	**93.6**	58.7	51.9	33.8
Environment Configuration Agent						
INSTALLAMATIC (Milliken et al., 2025a)	GPT-4.1	37.5	70.4	48.9	45.3	29.1
	DeepSeek-V3	40.7	76.8	53.2	49.3	32.5
Repo2Run (Hu et al., 2025)	GPT-4.1	44.2	72.3	54.8	48.5	38.6
	DeepSeek-V3	46.3	74.2	56.8	44.6	41.2
EvoConfig	GPT-4.1	49.2	75.4	59.7	**56.5**	39.4
	DeepSeek-V3	**52.3**	77.9	**62.6**	48.3	**45.9**

Table 3: Main results across different agents on EnConda-Bench.

strates consistently stronger performance across both error perception and repair-related metrics, indicating improved handling of configuration failures during interactive execution.

We observe that code agents such as SWE-Agent and OpenHands show improved error perception compared to generic agents, but their ability to translate diagnosis into effective repair actions remains limited. For instance, OpenHands with DeepSeek-V3 achieves an error type F1 score of 58.7 and an error description accuracy of 51.9, while its fix suggestion accuracy is only 33.8, indicating a clear gap between error understanding and action-level repair. Environment configuration agents further improve repair effectiveness: INSTALLAMATIC increases fix accuracy to 32.5, and Repo2Run reaches 41.2 under DeepSeek-V3, demonstrating the benefit of explicitly modeling environment interaction. EvoConfig consistently achieves the strongest performance across both backbones, reaching error type F1 scores of 59.7/62.6 and fix suggestion accuracies of 39.4/45.9 under GPT-4.1 and DeepSeek-V3, respectively. These results suggest that EvoConfig better aligns fine-grained error analysis with actionable repair guidance, highlighting the value of adaptive, expert-driven diagnosis in improving process-level error correction.

5.2 Ablation Study

We conduct ablation studies on a randomly sampled set of 100 repositories from EnvBench to examine the contributions of the environment information extraction module and the self-evolving expert diagnosis module, with additional details provided in Appendix C. As shown in Table 4, removing the self-evolving expert diagnosis module leads to a substantial drop in environment building success rate (EBSR) from 83.0% to 75.0%, while removing the environment information extraction module results in a smaller decrease to 82.0%. The runtime comparison in Figure 3, measured on 30 repositories successfully configured by all variants, further shows that EvoConfig consistently achieves lower average configuration time. In particular, disabling environment information extraction leads to longer execution trajectories, whereas removing expert diagnosis causes the most significant slowdown and higher variance, indicating repeated and inefficient repair attempts. These results suggest that environment information extraction mainly improves efficiency, while adaptive diagnosis is critical for robustness and success.

Method	EBSR	# Successfully Built Environments
w/o Environment Info Extraction	82.0%	82
w/o Self-Evolving Expert Diagnosis	75.0%	75
EvoConfig	83.0%	83

Table 4: Ablation results of EvoConfig in terms of environment build success.

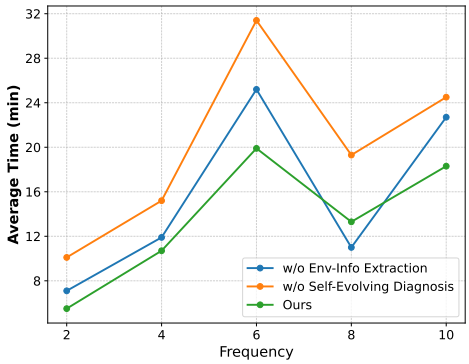

Figure 3: Runtime distribution of successful environment builds in the ablation study.

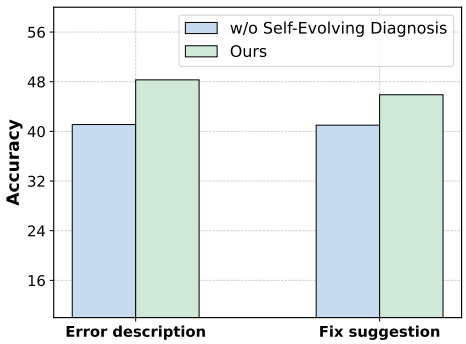

Figure 4: Ablation results of EvoConfig on process-level error correction.

Method	Times (min)	Tokens	Cost
Repo2Run	30.5	495268	$0.33
EvoConfig	**20.9**	**229531**	**$0.16**

Table 5: Effiency comparison of successful building.

Category	# Case (%)
Hardware Insufficiency	23 (32.4%)
Config Files Missing	20 (28.2%)
Dependency Installation Timeout	10 (14.1%)
Unit Tests Missing	5 (7.0%)
Runtest Timeout	13 (18.3%)

Table 6: Analysis of failure cases in EvoConfig.

To further evaluate the effectiveness of the self-evolving expert diagnosis module at the process level, we conduct a focused ablation study on EnConda-Bench using DeepSeek-V3, with results shown in Figure 4. The evaluation is performed on all instances from a randomly sampled set of 100 repositories, and more detailed analysis is provided in Appendix D. Removing the diagnosis module consistently degrades performance across all stages of error handling, with the error description accuracy decreasing from 48.3 to 44.1, and fix suggestion accuracy declining from 45.9 to 41.0. Notably, the performance gap is most pronounced in fix suggestion accuracy, indicating that without adaptive expert feedback, the agent struggles to translate error understanding into effective corrective actions. In contrast, EvoConfig maintains a more consistent perception feedback action performance, indicating that the self evolving diagnostic mechanism enhances error correction ability of agent at the process-level while ensuring environment configuration success rate.

6 Discussion

6.1 Efficiency and Cost Analysis

As shown in Table 5, experiments on the 324 EnvBench repositories show that EvoConfig substantially improves both aspects: it reduces the average configuration time per repository from 30.5 minutes to 20.9 minutes and incurs lower token-level and monetary cost under `gpt-3.5-turbo`. These gains are largely attributable to EvoConfig's multi-agent design, which separates execution control from error diagnosis and feedback interpretation, preventing long execution traces from repeatedly entering the main agent's context and thereby reducing redundant reasoning during configuration.

6.2 Failure Case Study

We analyze the failure cases of EvoConfig on the EnvBench benchmark, with the distribution summarized in Table 6. Most failures are caused by external execution constraints or repository-intrinsic issues rather than limitations of the agent itself. Hardware insufficiency is the most common failure source, accounting for 32.4% of failed cases, followed by missing or incomplete configuration information (28.2%), where repositories lack core files such as `pyproject.toml`, `setup.py`, or `requirements.txt`. A further portion of failures arises from execution timeouts during dependency installation or test execution, reflecting practical limits imposed by heavy dependencies and long-running tests.

7 Related Work

Executable environments as a prerequisite for training and evaluating SWE agents. Executable environments are a prerequisite for repository-level SWE agents, because both training signals and evaluation protocols assume that

projects can be built and their verification procedures can be executed reproducibly. Accordingly, environment configuration is deeply embedded in popular benchmarks and data pipelines: several widely-used settings rely on manual, repository-specific environment curation, such as SWE-bench (Jimenez et al., 2024), SWE-Flow (Zhang et al., 2025b), SWE-Gym (Pan et al., 2025), and R2E-Gym (Jain et al., 2025). Recent benchmark and data construction workflows increasingly incorporate automated or semi-automated environment synthesis as a critical stage, including SetupAgent (Vergopoulos et al., 2025), SWE-smith (Yang et al., 2025a), SWE-Factory (Guo et al., 2025), SWE-bench-Live (Zhang et al., 2025c), SWE-Compass (Xu et al., 2025), and SWE-Bench++ (Wang et al., 2025b). Collectively, these trends motivate environment synthesis as a first-class research problem that directly controls the scalability and reliability of executable SWE data.

Methods for automated environment setup. Automated setup methods broadly fall into *deterministic* and *agentic* families. Deterministic approaches implement setup as automated scripts or fixed pipelines, which execute standardized procedures across diverse repositories to maximize reproducibility and reduce per-repository manual effort; a representative example is R2E (Jain et al., 2024), which instantiates executable test environments via scripted setup procedures. EnvBench (Eliseeva et al., 2025) spans both families: it introduces a benchmark for automated environment setup, includes a deterministic shell-script baseline, and also evaluates LLM-based Bash agents under the same task definition and proxy-based verification signals. Template-guided container synthesis constrains Dockerfile structure while leaving repository-specific slots to be filled, improving robustness at scale in SWE-Bench++ (Wang et al., 2025b). Agentic approaches treat setup as interactive search-and-repair: an LLM agent retrieves commands from documentation and project artifacts, executes them in a sandbox, diagnoses failures from logs, and iteratively refines the plan, as done in SetupAgent (Vergopoulos et al., 2025) and in the RepoLaunch pipeline of SWE-bench-Live (Zhang et al., 2025c); SWE-Factory adopts multi-agent decomposition and environment reuse to amortize successful configurations (Guo et al., 2025). Related systems target key subroutines, including scalable export of runnable Docker environments in Repo2Run (Hu et al., 2025), installation under incomplete documentation in Installamatic (Milliken et al., 2025b), and test execution for arbitrary projects in ExecutionAgent (Bouzenia and Pradel, 2025b), while earlier dependency inference in DockerizeMe illustrates the limits of purely static signals (Horton and Parnin, 2019).

Benchmarks that evaluate environment setup ability. A complementary line of work elevates environment bootstrapping into a first-class benchmarked capability. EnvBench (Eliseeva et al., 2025) provides a large-scale benchmark for repository-specific setup across Python and JVM projects and introduces automatic proxy metrics such as missing-import and compilation checks to support scalable evaluation. SetupBench (Arora et al., 2025) formalizes bootstrapping from a bare Linux sandbox with deterministic one-line verification commands, enabling fine-grained analysis of failure modes such as incomplete toolchains and non-persistent modifications. Enconda-bench (Kuang et al., 2025) moves beyond end-to-end success by scoring *process-level* trajectories and diagnosing capabilities such as setup planning, error localization, and feedback-driven repair under realistically perturbed instructions. SWE-Compass (Xu et al., 2025) incorporates configuration and deployment tasks into a broader agentic coding evaluation suite, contextualizing setup as part of end-to-end agent behavior. Finally, Multi-Docker-Eval (Fu et al., 2025) expands evaluation to multi-language repositories and emphasizes both effectiveness and efficiency, including time and resource usage as well as resulting image size.

8 Conclusion

In this paper, we propose EvoConfig, a self-evolving multi-agent framework that decouples execution, diagnosis, and repair. By combining lightweight environment information extraction with adaptive expert diagnosis, EvoConfig improves configuration robustness and efficiency. Experiments on multiple benchmarks demonstrate strong environment building performance with reduced time and token cost, while process-level evaluations show improved error understanding and repair quality. EvoConfig focuses on enabling test execution, and extending it to reason about test outcomes remains future work.

Limitations

EvoConfig focuses on constructing executable environments and improving process-level error correction, but does not reason about test correctness. Our evaluation considers whether unit tests can be executed, rather than the proportion of tests that pass. In practice, test failures may arise from issues beyond environment configuration. While EvoConfig enables tests to run reliably, analyzing test outcomes and debugging failing tests remain outside its current scope and are left for future work.

Ethical Considerations

Potential Risks Although EvoConfig improves robustness and efficiency in automated environment configuration, several risks remain. The framework depends on execution feedback quality, and noisy or incomplete errors may still affect diagnosis. In addition, the self-evolving mechanism may require sufficient feedback to stabilize in early stages. Finally, EvoConfig focuses on building runnable environments and does not guarantee the correctness of test outcomes, which may limit its use in strict functional validation settings.

Ethical Statement This work focuses on automated environment configuration for open-source software repositories using large language model–based agents. All experiments are conducted on publicly available data and executed in isolated environments, without involving personal, sensitive, or private information.

LLMs Usage Statement Large language models were used to assist with language polishing and clarity improvement during the writing process; all technical content, experimental design, and conclusions were developed and verified by the authors. The proposed method aims to improve research reproducibility and scalability in software engineering and does not introduce new ethical risks beyond those of existing automated development tools.

References

Anysphere. 2023. Cursor: An ai-first code editor. https://www.cursor.com/. Official website, accessed 2026-01-05.

Avi Arora, Jinu Jang, and Roshanak Zilouchian Moghaddam. 2025. Setupbench: Assessing software engineering agents' ability to bootstrap development environments. *Preprint*, arXiv:2507.09063.

bndr. 2016. pipreqs: Generate pip requirements.txt file based on imports of any project. https://github.com/bndr/pipreqs. GitHub repository, accessed 2026-01-05.

Islem Bouzenia and Michael Pradel. 2025a. You name it, i run it: An llm agent to execute tests of arbitrary projects. *Proceedings of the ACM on Software Engineering*, 2(ISSTA):1054–1076.

Islem Bouzenia and Michael Pradel. 2025b. You name it, I run it: An LLM agent to execute tests of arbitrary projects. *Proc. ACM Softw. Eng.*, 2(ISSTA):1054–1076.

Aleksandra Eliseeva, Alexander Kovrigin, Ilia Kholkin, Egor Bogomolov, and Yaroslav Zharov. 2025. Envbench: A benchmark for automated environment setup. In *ICLR 2025 Third Workshop on Deep Learning for Code*.

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.

GitHub. 2021. Github copilot: Your ai pair programmer. https://github.com/features/copilot. Online product page, accessed 2026-01-05.

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

Junda He, Christoph Treude, and David Lo. 2025. Llm-based multi-agent systems for software engineering: Literature review, vision, and the road ahead. *ACM Transactions on Software Engineering and Methodology*, 34(5):1–30.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, and 1 others. 2023. Metagpt: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Eric Horton and Chris Parnin. 2019. Dockerizeme: automatic inference of environment dependencies for python code snippets. In *Proceedings of the 41st International Conference on Software Engineering, ICSE 2019, Montreal, QC, Canada, May 25-31, 2019*, pages 328–338. IEEE / ACM.

Ruida Hu, Chao Peng, XinchenWang, Junjielong Xu, and Cuiyun Gao. 2025. Repo2run: Automated building executable environment for code repository at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Naman Jain, Manish Shetty, Tianjun Zhang, King Han, Koushik Sen, and Ion Stoica. 2024. R2e: Turning any github repository into a programming agent test environment. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zhang, Liang Zheng, Koushik Sen, and Ion Stoica. 2025. R2e-gym: Procedural environment generation and hybrid verifiers for scaling open-weights SWE agents. In *Second Conference on Language Modeling*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

Aleksandar Krnjaic, Raul D Steleac, Jonathan D Thomas, Georgios Papoudakis, Lukas Schäfer, Andrew Wing Keung To, Kuan-Ho Lao, Murat Cubuktepe, Matthew Haley, Peter Börsting, and 1 others. 2024. Scalable multi-agent reinforcement learning for warehouse logistics with robotic and human coworkers. In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 677–684. IEEE.

Jiayi Kuang, Yinghui Li, Xin Zhang, Yangning Li, Di Yin, Xing Sun, Ying Shen, and Philip S. Yu. 2025. Process-level trajectory evaluation for environment configuration in software engineering agents. *Preprint*, arXiv:2510.25694.

Dingbang Liu, Fenghui Ren, Jun Yan, Guoxin Su, Wen Gu, and Shohei Kato. 2024. Scaling up multi-agent reinforcement learning: An extensive survey on scalability issues. *IEEE Access*, 12:94610–94631.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2023. Repobench: Benchmarking repository-level code auto-completion systems. *arXiv preprint arXiv:2306.03091*.

Louis Milliken, Sungmin Kang, and Shin Yoo. 2025a. Beyond pip install: Evaluating llm agents for the automated installation of python projects. In *2025 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 1–11. IEEE.

Louis Milliken, Sungmin Kang, and Shin Yoo. 2025b. Beyond pip install: Evaluating LLM agents for the automated installation of python projects. In *IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2025, Montreal, QC, Canada, March 4-7, 2025*, pages 1–11. IEEE.

Jiayi Pan, Xingyao Wang, Graham Neubig, Navdeep Jaitly, Heng Ji, Alane Suhr, and Yizhe Zhang. 2025. Training software engineering agents and verifiers with SWE-gym. In *Forty-second International Conference on Machine Learning*.

Shrey Pandit, Xuan-Phi Nguyen, Yifei Ming, Austin Xu, Jiayu Wang, Caiming Xiong, and Shafiq Joty. 2025. Synthesizing agentic data for web agents with progressive difficulty enhancement mechanisms. *arXiv preprint arXiv:2510.13913*.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Garðar Ingvarsson Juto, Timon Willi, Ravi Hammond, Akbir Khan, Christian Schroeder de Witt, and 1 others. 2024. Jaxmarl: Multi-agent rl environments and algorithms in jax. *Advances in Neural Information Processing Systems*, 37:50925–50951.

Konstantinos Vergopoulos, Mark Niklas Mueller, and Martin Vechev. 2025. Automated benchmark generation for repository-level coding tasks. In *Forty-second International Conference on Machine Learning*.

Junhao Wang, Daoguang Zan, Shulin Xin, Siyao Liu, Yurong Wu, and Kai Shen. 2025a. Swe-mirror: Scaling issue-resolving datasets by mirroring issues across repositories. *arXiv 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. 2025b. Swe-bench++: A framework for the scalable generation of software engineering benchmarks from open-source repositories. *Preprint*, arXiv:2512.17419.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, and 1 others. 2024a. Opendevin: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 3.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, and 1 others. 2024b. Openhands: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*.

Yanlin Wang, Wanjun Zhong, Yanxian Huang, Ensheng Shi, Min Yang, Jiachi Chen, Hui Li, Yuchi Ma, Qianxiang Wang, and Zibin Zheng. 2025c. Agents in software engineering: Survey, landscape, and vision. *Automated Software Engineering*, 32(2):70.

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? *arXiv preprint arXiv:2511.13646*.

Yiqing Xie, Alex Xie, Divyanshu Sheth, Pengfei Liu, Daniel Fried, and Carolyn Rose. 2024. Codebench-gen: Creating scalable execution-based code generation benchmarks. *arXiv preprint arXiv:2404.00566*.

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. 2025. Swe-compass: Towards unified evaluation of agentic coding abilities for large language models. *Preprint*, arXiv:2511.05459.

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

John Yang, Kilian Lieret, Carlos E Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. 2025a. SWE-smith: Scaling data for software engineering agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Zonghan Yang, Shengjie Wang, Kelin Fu, Wenyang He, Weimin Xiong, Yibo Liu, Yibo Miao, Bofei Gao, Yejie Wang, Yingwei Ma, and 1 others. 2025b. Kimi-dev: Agentless training as skill prior for swe-agents. *arXiv preprint arXiv:2509.23045*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*.

Jiayi Zhang, Yiran Peng, Fanqi Kong, Yang Cheng, Yifan Wu, Zhaoyang Yu, Jinyu Xiang, Jianhao Ruan, Jinlin Wang, Maojia Song, and 1 others. 2025a. Autoenv: Automated environments for measuring cross-environment agent learning. *arXiv preprint arXiv:2511.19304*.

Lei Zhang, Jiaxi Yang, Min Yang, Jian Yang, Mouxiang Chen, Jiajun Zhang, Zeyu Cui, Binyuan Hui, and Junyang Lin. 2025b. Synthesizing software engineering data in a test-driven manner. In *Forty-second International Conference on Machine Learning*.

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. 2025c. SWE-bench goes live! In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

A Environment Build Success Evaluation

A.1 Baselines

We evaluate environment configuration performance using the following representative baselines:

- **pipreqs**: A static dependency analysis tool that inspects Python import statements to infer required packages and generates a `requirements.txt` file, which is then used to construct a Dockerfile.
- **LLM Generator**: A direct LLM-based approach that parses repository `README` files and generates executable Dockerfiles without iterative interaction.
- **SWE-agent**: An LLM-based agent with a custom agent–computer interface that supports file inspection, editing, and command execution; its framework is retained while prompts are adapted for environment configuration.
- **Repo2Run**: A strong agent-based baseline specifically designed for iterative environment configuration through interaction with the execution environment.

Method	EBSR	Average Times (min)
w/o Environment Info Extraction	82.0%	22.5
w/o Self-Evolving Expert Diagnosis	75.0%	27.2
EvoConfig	83.0%	20.5

Table 7: Ablation results of EvoConfig in terms of environment build success and time cost.

B Process Error Correction Evaluation

To evaluate process-level error correction, we compare EvoConfig with several representative agent-based baselines that support iterative interaction with the execution environment.

- **SWE-agent**: An LLM-based agent originally designed for automated bug fixing, which supports file inspection, code editing, and command execution through a custom agent–computer interface. We adapt its prompts for process-level environment error correction.
- **OpenHands**: A general-purpose autonomous agent framework for software engineering tasks, used here as a generic baseline to assess its ability to correct environment errors through multi-step interaction.
- **INSTALLAMATIC**: An LLM-driven system that focuses on generating installation and setup commands for resolving dependency-related environment issues, without explicit long-horizon agent planning.

Method	Backbone	Perception			Feedback	Feedback and Action
		Error Type			Error Description	Fix Suggestion
		Pre.	**Rec.**	**F1**	**ACC.**	**ACC.**
w/o Self-Evolving Expert Diagnosis	DeepSeek-V3	43.1	76.2	55.1	44.1	41.0
EvoConfig	DeepSeek-V3	**52.3**	**77.9**	**62.6**	**48.3**	**45.9**

Table 8: Complete ablation experiment results of EvoConfig on process-level error correction.

- **Repo2Run**: A specialized agent-based system for repository environment configuration.

C Time Efficiency Analysis

We analyze environment building success rates (EBSR) and time efficiency of different variants, as reported in Table 7. EvoConfig achieves both the highest success rate (83.0%) and the lowest average configuration time (20.5 minutes), indicating that its improved performance does not come at the cost of increased runtime. In contrast, removing the environment information extraction module slightly reduces the success rate to 82.0% while increasing the average configuration time to 22.5 minutes, suggesting that limited environment awareness leads to inefficient trial-and-error and correspondingly longer execution trajectories.

Specifically, the success rate drops substantially to 75.0%, accompanied by a significant increase in average configuration time to 27.2 minutes. This observation indicates that static diagnosis strategies not only reduce the likelihood of successful environment construction but also result in repeated and inefficient repair attempts, thereby prolonging the overall configuration process. Together, these results demonstrate that while environment information extraction mainly contributes to execution efficiency, adaptive expert diagnosis is crucial for achieving both high configuration success rates and low configuration time.

D Effect of Self-Evolving Expert Diagnosis

We evaluate the contribution of the self-evolving expert diagnosis module by comparing the full EvoConfig framework with a variant that removes this component while using the same backbone (DeepSeek-V3). As shown in Table 8, removing self-evolving diagnosis results in a consistent performance degradation across all evaluation stages, indicating its critical role in the overall system.

From the perception perspective, the absence of self-evolving diagnosis leads to a noticeable drop in error type recognition performance, with the F1 score decreasing from 62.6 to 55.1, mainly due to reduced precision. This suggests that static expert behavior limits the agent's ability to accurately identify error patterns. Moreover, the accuracy of error description generation also declines from 48.3% to 44.1%, reflecting less precise feedback when adaptive diagnosis is disabled.

The impact is further reflected in the action stage, where fix suggestion accuracy drops from 45.9% to 41.0%. Since effective repair actions rely on accurate error understanding, these results demonstrate that self-evolving expert diagnosis is an essential component for maintaining coherent perception–feedback–action alignment in large-scale environment configuration.