NEURAL INFORMATION PROCESSING SYSTEMS

哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

ByteDance

Repo2Run: Automated Building Executable Environment for Code Repository at Scale

★ **NeurIPS 2025 Spotlight**

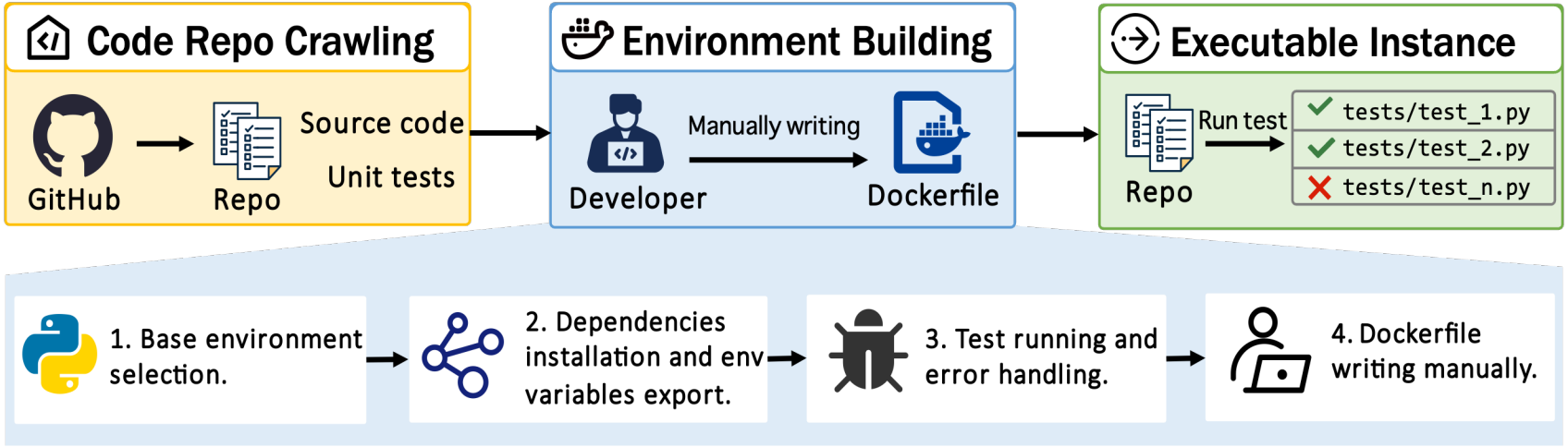
Ruida Hu[1], Chao peng[2*], Xinchen Wang[1], Junjielong Xu[2], Cuiyun Gao[1*]

[1]Harbin Institute of Technology, Shenzhen [2]ByteDance, Beijing

**Corresponding Authors*

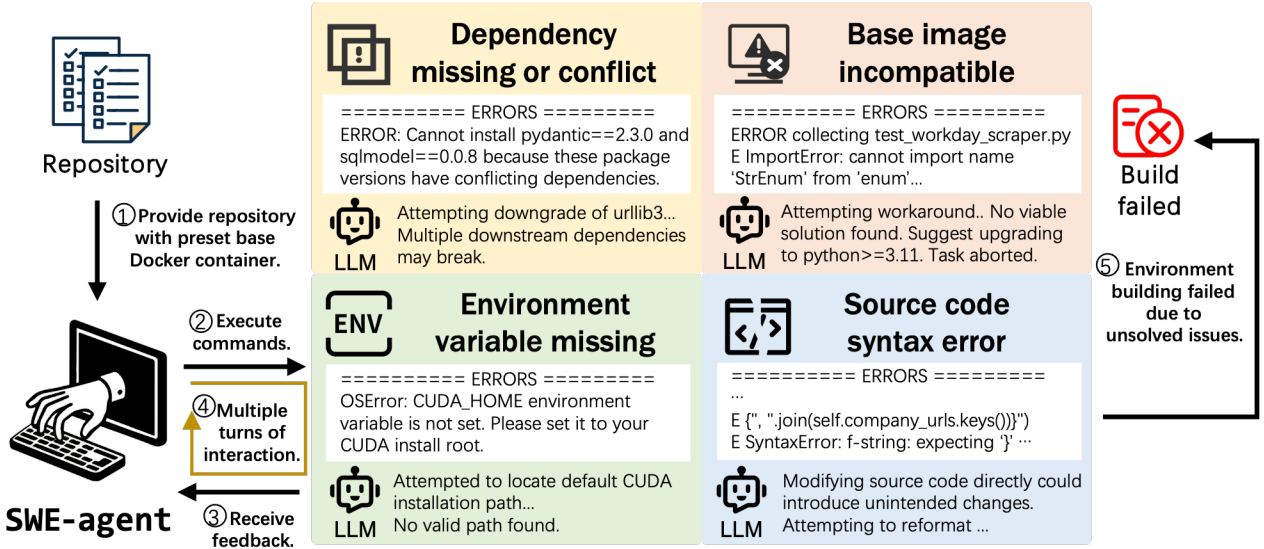
Presenter: Ruida Hu

Background & Challenges



Building executable environments is a vital step for testing AI coding agents, but it remains a significant manual bottleneck for developers. Automating this process is essential for enabling scalable, reproducible research and unlocking the next wave of AI-driven software engineering.

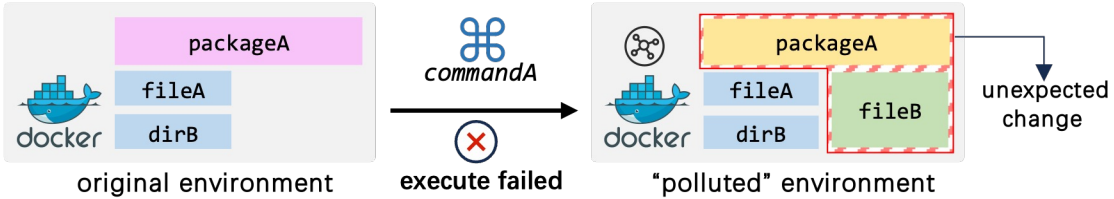
Background & Challenges



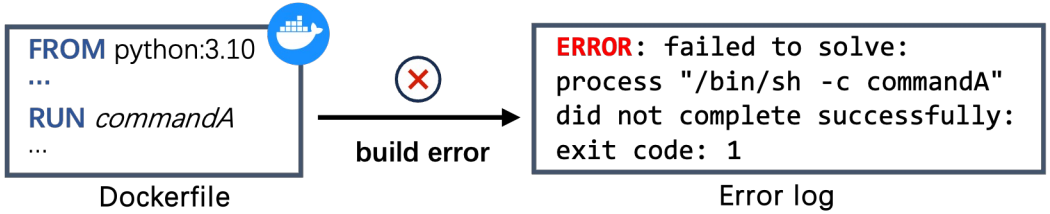
Challenge 1: Hard to explore a valid trajectory that can successfully build the environment.

Existing LLM agents struggle to build complex software environments due to ineffective error handling. We address this by designing eight specific actions within a dedicated build system, successfully guiding the agent through unresolved issues to completion.

Background & Challenges



(a) A failed execution of "*commandA*" leads to "pollution" of the environment.

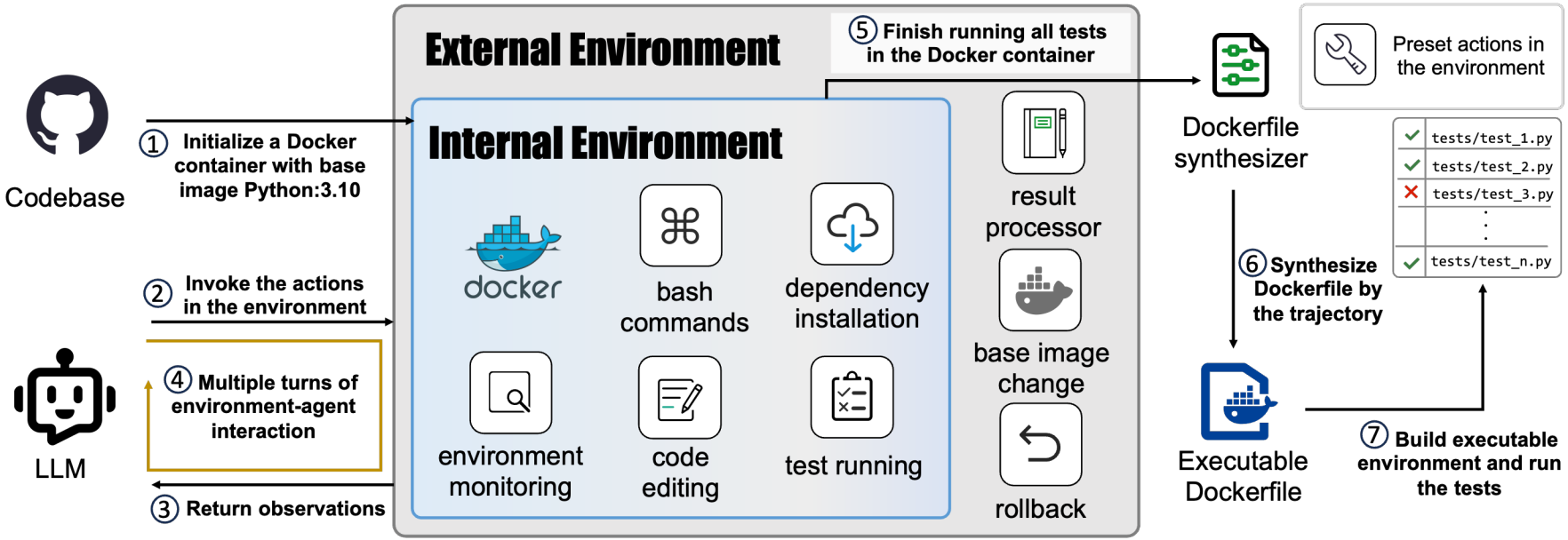


(b) Building failure caused by adding "*commandA*" to the Dockerfile.

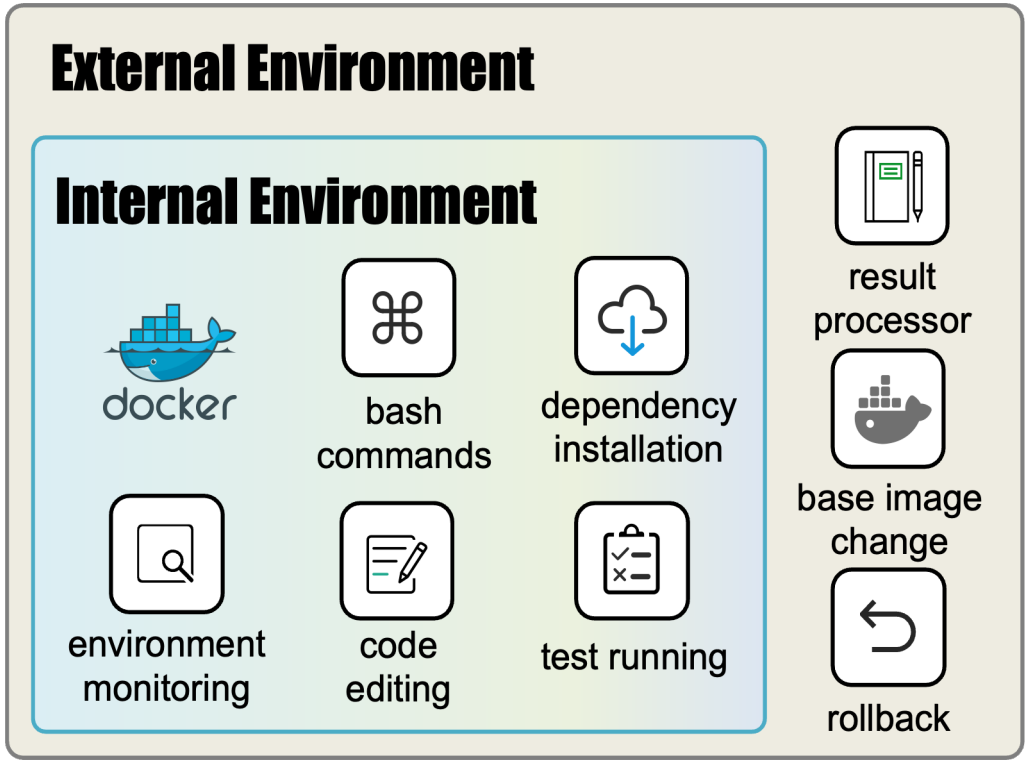
Challenge 2: Failing to synthesize a runnable Dockerfile without execution failure.

Incorrect commands can "pollute" the build environment, leading to synthesis failures when creating a runnable Dockerfile. To prevent this, we introduce an adaptive rollback action that reverts the environment to a clean state after a failed command, ensuring a consistent and error-free synthesis process.

Approach

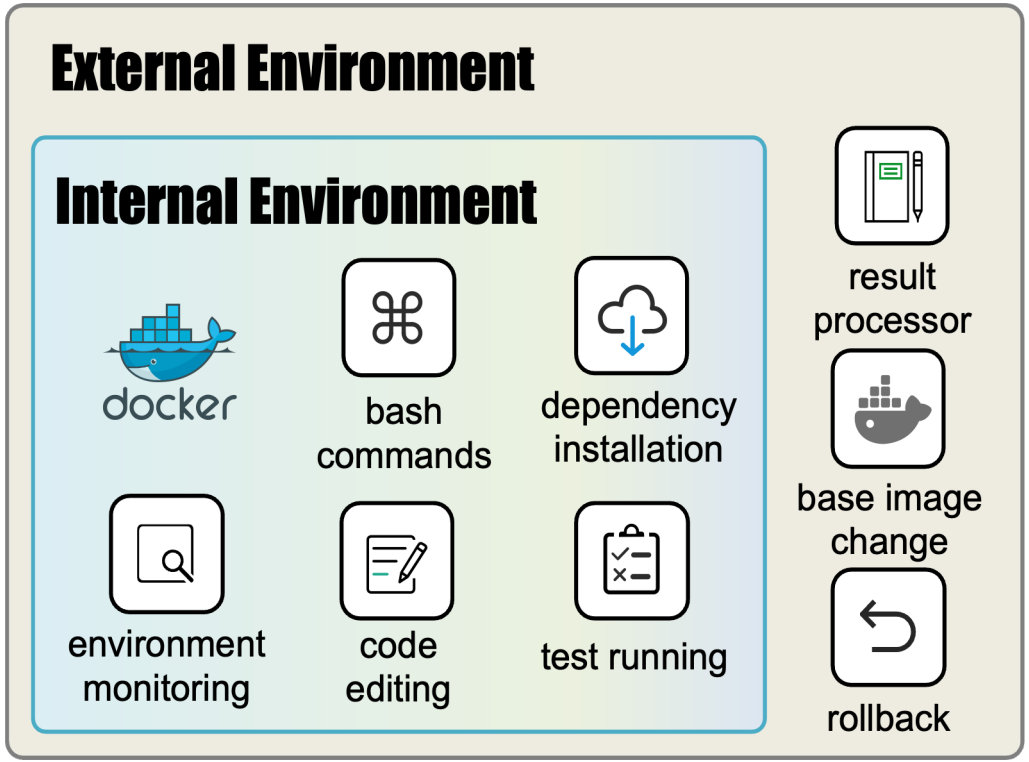

Codebase
① Initialize a Docker container with base image Python:3.10
② Invoke the actions in the environment
LLM
④ Multiple turns of environment-agent interaction
③ Return observations
External Environment
Internal Environment
docker
bash commands
dependency installation
environment monitoring
code editing
test running
⑤ Finish running all tests in the Docker container
result processor
base image change
rollback
Dockerfile synthesizer
Preset actions in the environment
tests/test_1.py
tests/test_2.py
tests/test_3.py
tests/test_n.py
⑥ Synthesize Dockerfile by the trajectory
Executable Dockerfile
⑦ Build executable environment and run the tests

Approach: build phase



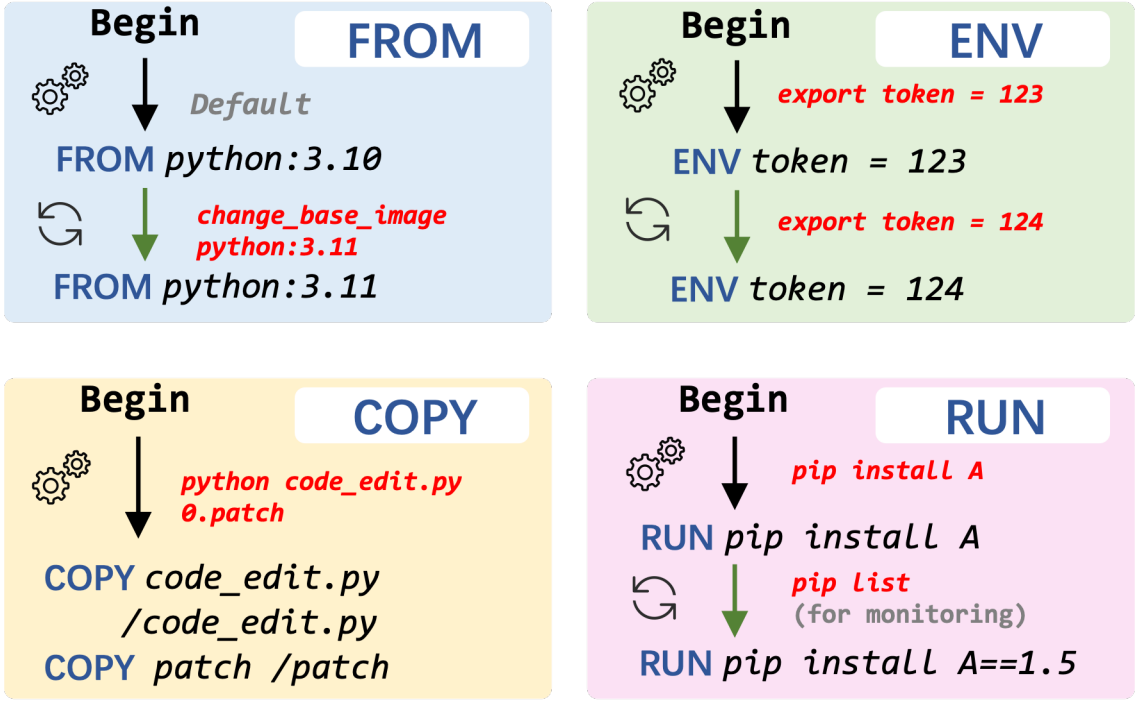
- **Result processor**: truncates long command outputs, presenting only the most critical information (the beginning and end of logs) to prevent overwhelming the LLM agent.
- **Base image change**: allows the agent to switch to a more suitable base image mid-process, which then resets the build environment and clears all prior commands to start fresh.
- **Rollback**: automatically restores the environment to a pre-saved snapshot after a command fails, preventing environment "pollution" and ensuring the final Dockerfile is runnable.

Approach: build phase



- **Environment Monitoring**: Inspects the environment's state, files, and dependencies.
- **Dependency Installation**: Installs and manages necessary software packages while handling conflicts.
- **Test Running**: Verifies the build's success by executing tests and provides feedback on failures.
- **Code Editing**: Modifies code to fix issues but is restricted from altering test files.
- **Bash Commands**: Executes any general-purpose bash command for maximum flexibility.

Approach: record phase



The **Dockerfile synthesizer** converts the successful sequence of build commands into a complete Dockerfile by translating them into *FROM*, *ENV*, *COPY*, and *RUN* statements to ensure an executable environment.

Experiment

We evaluate the effectiveness of Repo2Run on 420 Python code repositories. As the popular option, we select gpt-4o-2024-05-13 for all experiments, with the temperature uniformly set to 0.2.

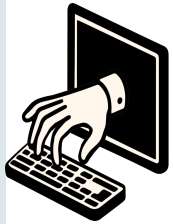
Benchmark

To validate our tool, we created a new benchmark by selecting high-quality Python repositories from GitHub that were created in 2024 with over 100 stars and a `test` or `tests` directory.

Evaluation Metrics

Dockerfile Generation Success Rate (DGSR): This metric measures the percentage of times a method successfully generates a Dockerfile that can be built into an image without any errors.
Environment Building Success Rate (EBSR): This metric measures the percentage of times the built Docker image successfully creates an environment where the project's tests can be executed with `pytest`, regardless of whether the tests pass or fail.

Experiment



SWE-agent

Baselines

pipreqs: An automated, non-LLM tool that generates a `requirements.txt` file by scanning the project's import statements, which we then use to create a basic Dockerfile.

LLM generator: A straightforward approach where we prompt an LLM to generate a Dockerfile directly by reading the instructions found in the repository's `README` file.

SWE-agent: An existing LLM agent designed for bug fixing, which we adapt for environment building by modifying its prompts while keeping its core framework for file interaction and command execution.

Results

Metric	DGSR	# Successfully Generated Dockerfiles	EBSR	# Successfully Built Environments
pipreqs	29.8%	125	6.0%	25
LLM generator	47.6%	200	22.1%	93
SWE-agent	26.9%	113	9.0%	38
Repo2Run	**100%**	**420**	**86.0%**	**361**

Repo2Run's Superior Performance: Repo2Run dramatically outperforms all baselines with an 86.0% environment building success rate, and its rollback feature ensures a 100% Dockerfile generation success rate.

Limitations of Baseline Methods: Baseline methods like `pipreqs` and `SWE-agent` frequently failed due to package conflicts or the inability to handle errors, proving that general-purpose approaches are insufficient.

Results

Type	AI/ML	SE	NLP	DV	Security	Others	All
# Success	221	70	41	10	7	12	361
# Total	267	73	47	11	8	14	420
EBSR	82.8%	95.9%	87.2%	90.9%	87.5%	85.7%	86.0%

Consistent Performance & Real-World Validity: Repo2Run maintains high success rates (over 80%) across diverse software domains, validating its effectiveness on a benchmark that accurately reflects real-world projects.

Results

Metric	DGSR	# Successfully Generated Dockerfiles	EBSR	# Successfully Built Environments
w/o dual-environment	92.4%	388	41.7%	175
w/o rollback	96.9%	407	83.6%	351
w/o Dockerfile synthesizer	19.5%	82	13.8%	58
Repo2Run	**100%**	**420**	**86.0%**	**361**

Removing the Dual-Environment Architecture: Taking away the specialized actions (like rollback) and leaving only basic bash commands caused a sharp 44.3% drop in the environment building success rate (EBSR), as the agent struggled to navigate and solve complex errors.

Removing the Dockerfile Synthesizer: Relying on the LLM to directly write the Dockerfile, instead of using our synthesizer, resulted in a catastrophic 80.5% drop in successfully generating a buildable file, demonstrating that a specialized tool is essential to translate action history into a reliable Dockerfile.

哈爾濱工業大學(深圳)
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

ByteDance

Conclusion

We introduce Repo2Run, the first LLM-based agent that automates the creation of executable environments and Dockerfiles for Python repositories. By leveraging a novel dual-environment architecture and a dedicated Dockerfile synthesizer, Repo2Run achieves an impressive 86.0% success rate on a benchmark of 420 real-world projects, establishing a strong foundation for scaling up reproducible software environments.

preprint

code

ByteDance SE Lab

WeChat