RepoLaunch: Automating Build and Management of Code Repositories across Languages and Platforms

Kenan Li*†, Rongzhi Li*, Linghao Zhang*, Qirui Jin*†, Liao Zhu*,
Xiaosong Huang, Geng Zhang, Yikai Zhang, Shilin He, Chengxing Xie,
Xin Zhang, Zijian Jin, Bowen Li, Chaoyun Zhang, Yu Kang, Yufan Huang,
Elsie Nallipogu, Saravan Rajmohan, Qingwei Lin, Dongmei Zhang

* Equal contribution † Corresponding authors

NeurIPS 2026 Main Track Poster
arxiv:2603.05026

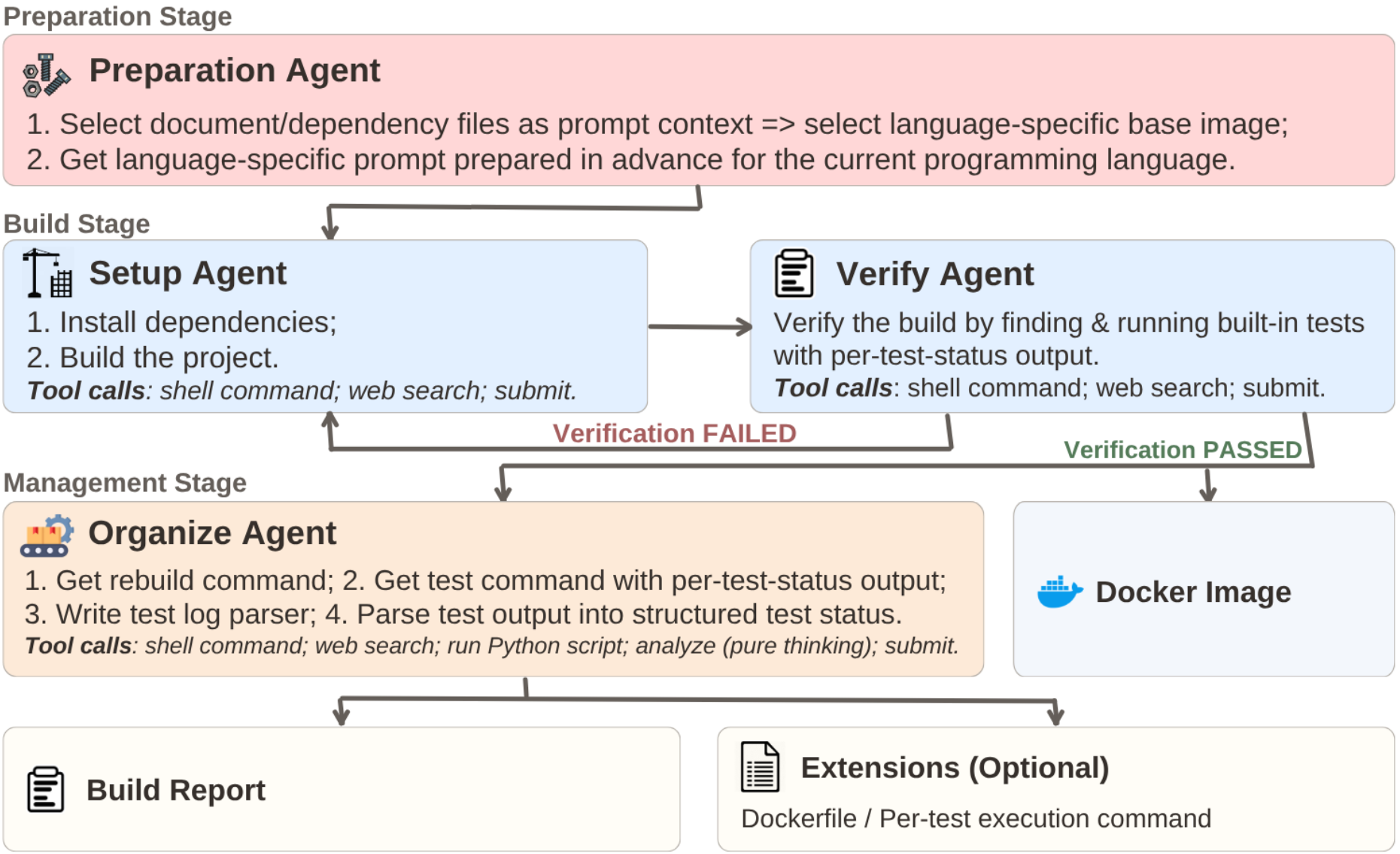
Motivation: Automatic Repo Build & Test

- While numerous prior works focused on agentic code repair, ***automatic repository build and test*** remains under-explored.
- While people have begun to use LM agents in their own projects to facilitate repo build and test empirically for some specific languages and a single platform, reusable ***repo build and test*** tools and dedicated ***failure analysis*** of the problem remain missing.
  - GitTaskBench (AAAI 2026): repo build and test failure takes **65%** of all the failures when coding agents implement a SWE task.
- Recently, the evaluation and training of coding agents require numerous ***executable environments*** with ***test status extractors***, which still require heavy manual crafting effort so far.

RepoLaunch: What it Resolves

- Install dependencies and build a code repository in a Docker image, with image layer information saved for Dockerfile re-construction.
- Get organized commands to rebuild the repository inside its container after repository source code changes.
- Get test commands to output per-testcase status, write a parser to parse test output into structured testcase-status mapping
  - and optionally find commands to run individual testcases.

Method of RepoLaunch: an Agentic Workflow



Ablation of RepoLaunch Modules: Does each really work?

Experiment	Build Success	Management Success	Verification Success	Avg. Build Steps	Avg. Manage. Steps
Linux					
search + prompt + image + str-replace-editor #	84.9% (129/152)	89.9% (116/129)	92.2% (107/116)	23.8	21.1
Baseline (search + prompt + image)	**78.9%** (120/152)	**89.2%** (107/120)	**91.6%** (98/107)	**25.0**	**21.8**
w/o web search	71.7% (109/152)	85.3% (93/109)	91.4% (85/93)	27.7	25.1
w/o language-specific prompt	72.3% (86/152)	80.2% (69/86)	84.1% (58/69)	26.1	24.7
w/o language-specific image*	67.1% (102/152)	88.2% (90/102)	91.1% (82/90)	31.1	22.5
Windows					
search + prompt + image + str-replace-editor #	68.0% (136/200)	72.8% (99/136)	80.8% (80/99)	36.7	29.8
Baseline (search + prompt + image)	**60.5%** (121/200)	**70.2%** (85/121)	**80.0%** (68/85)	**41.2**	**31.7**
w/o web search	51.5% (103/200)	66.0% (68/103)	80.9% (55/68)	48.6	39.8
w/o language-specific prompt	52.5% (105/200)	61.9% (65/105)	73.8% (48/65)	47.6	38.2
w/o language-specific image	46.0% (92/200)	70.7% (65/92)	80.0% (52/65)	58.6	32.9

* The baseline is the setting used in the following experiments. As RepoLaunch is used to create executable environments for SWE bug-fix tasks, the bugs in the repos need to be preserved, so file editor is by default disabled.

Success Rate and Cost Comparison with Prior Works: RepoLaunch achieves SOTA using a low cost

Build and *test extraction* success

On our repo build&test benchmark containing 152 instances across 8 languages in Appendix F.1. The two agents are driven by GPT-5 with Medium thinking effort.

Method	Languages Tested	Build	Test Extraction
RepoLaunch	**All the 8 languages**	**78.9%**	**81.7%**
SWE-agent	All the 8 languages	62.5%	50.5%

* Test extraction success rate is conditional on successful build. Test extraction success requires both correct test command (test success) and accurate test log parser (extraction of full test statuses).

Build success against prior methods

On the repo build&test benchmark containing 152 instances across 8 languages in Appendix F.1. Agentic methods below are driven by GPT-5 with Medium thinking effort.

Method	Type	Python	C/C++	All 8 languages
RepoLaunch	**Agentic**	**78.6%**	**81.3%**	**78.9%**
repo2run	Agentic	59.5%	—	—
CXXCrafter	Agentic	—	68.8%	—
SWE-agent	Agentic	52.4%	62.5%	62.5%
railpack	Rule-based	23.8%	31.3%	31.6%
pipreqs	Rule-based	40.5%	—	—

Comparison on *Python build* cost and success

Method	Source	API cost per task	Build success
RepoLaunch	**Our experiment**	**$0.65**	**79%**
repo2run	Our experiment	$0.70	60%
swe-rebench	arxiv:2505.20411,2602.23866	$0.10	20~30%
OpenSWE	arXiv:2603.13023	$19.1	~50%

Scalability of RepoLaunch:
RepoLaunch is robust across different repo sizes

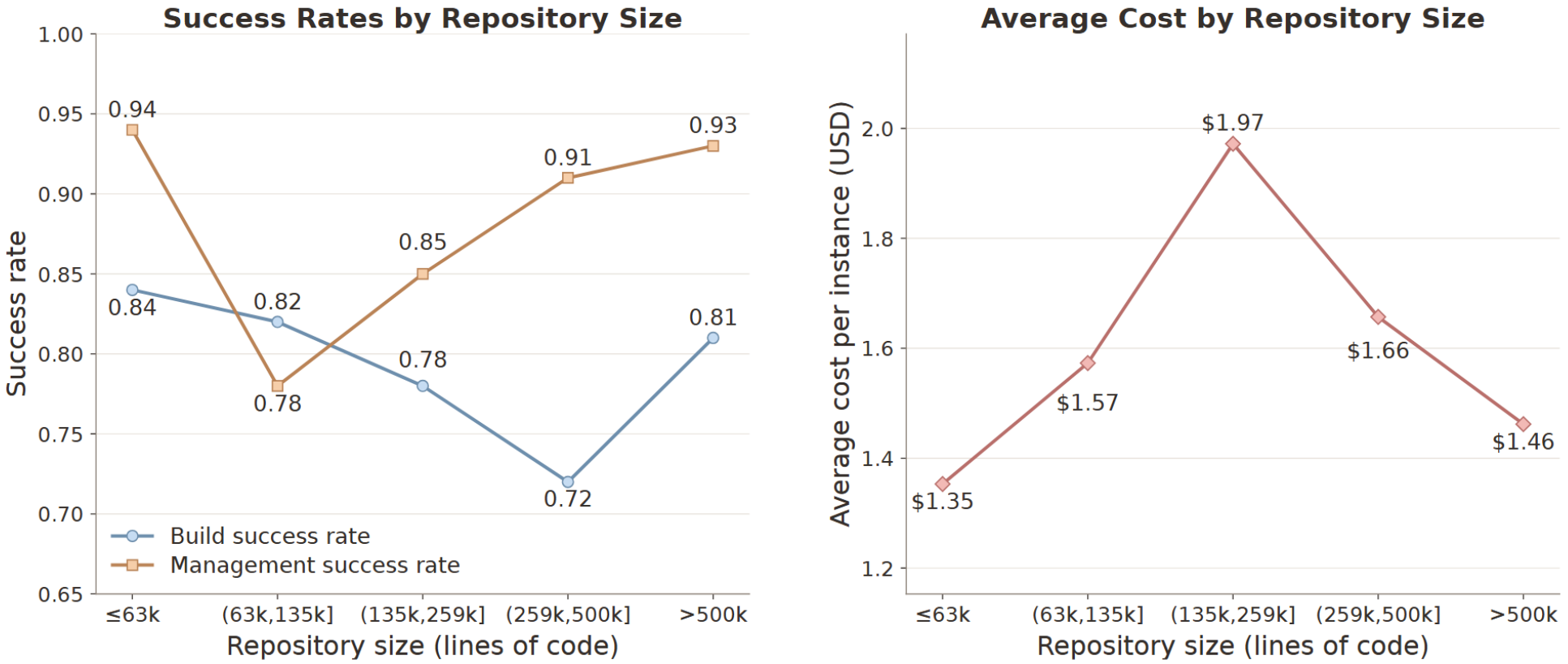
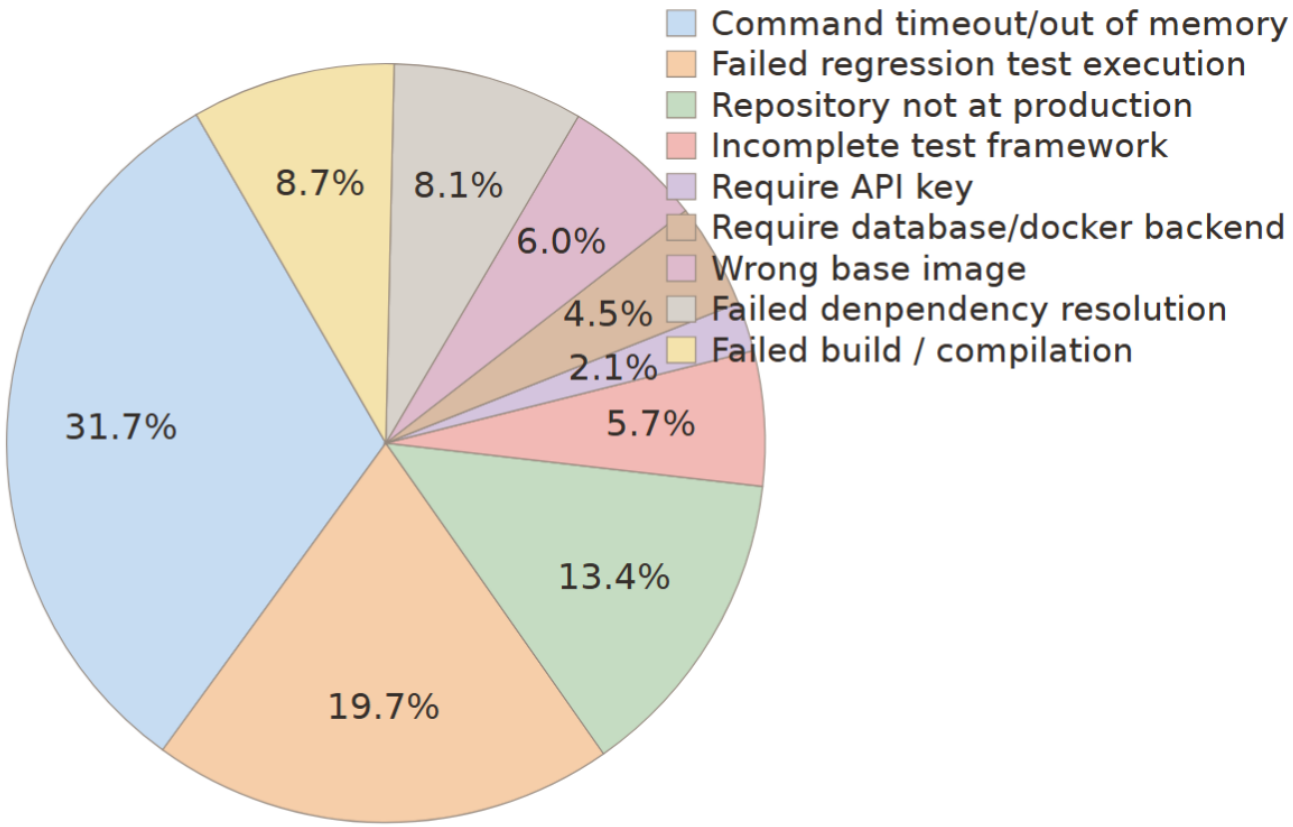
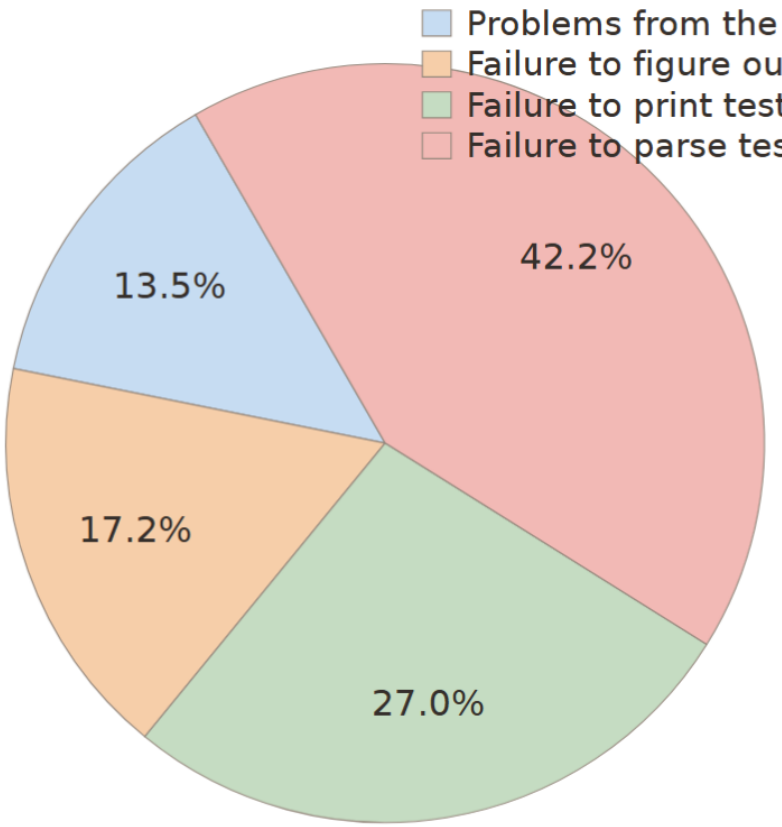


Figure 4: Scalability of RepoLaunch when repo size grows. Each of the five repo size ranges contains 75 repos.

Failure Patterns of RepoLaunch



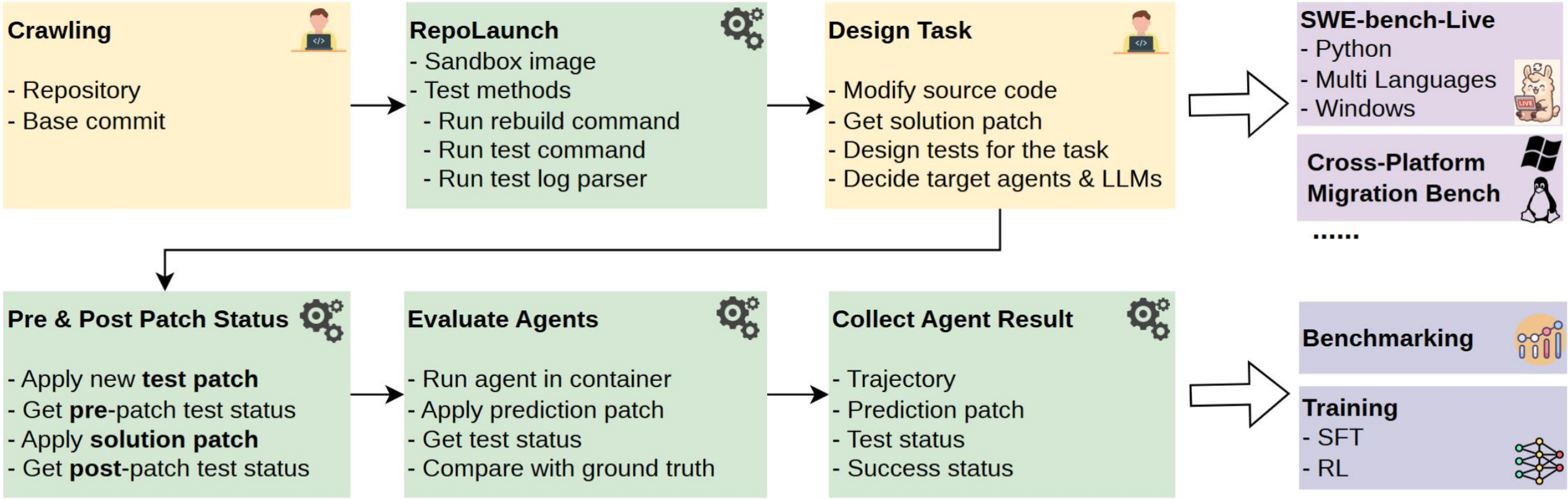
(a) Failure Pattern at Build Stage



(b) Pattern at Management Stage

Figure 3: The proportion of different failure patterns in the Build and the Management stage .

Application of RepoLaunch:
A fully automated SWE dataset creation pipeline driven by RepoLaunch



Application of RepoLaunch:
RepoLaunch creates ***SWE-bench-Live***

- We release ***SWE-bench-Live***, the **first** automatically updating, multi-language and multi-OS SWE task set for agentic benchmarking and training.
- Each task instance is a **GitHub issue-PR pair** crawled from GitHub.
- ***RepoLaunch*** contributes the **executable environment** and **test status extractor** of each task instance in our SWE-bench-Live.
- ***RepoLaunch*** eliminates heavy human effort in SWE dataset creation.
  With RepoLaunch, the only human intervention is task design.

RepoLaunch creates much more multilingual SWE tasks than prior *human-crafted* SWE benchmarks

RepoLaunch creates much more multilingual SWE tasks than prior works

Benchmark	Languages Covered	Tasks	Repos Covered
SWE-bench-Live/MultiLang + Windows (ours)	**9**	**1077+65**	**431**
prior SWE-bench/Multilingual	9	300	41
prior SWE-bench-Pro	4	642	11

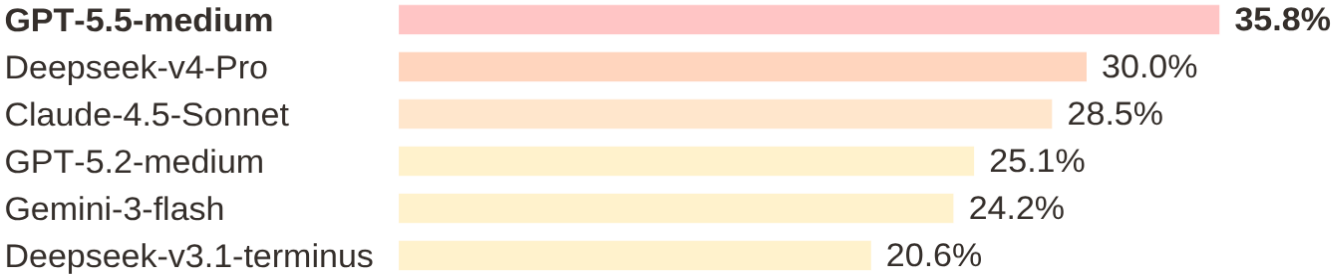
Benchmark creation success rates using RepoLaunch

Split (language or platform)	Build	Management	Number of Tasks
JS/TS	89.0%	92.3%	**219**
C/C++	82.4%	86.4%	**253**
C#	92.0%	95.3%	**136**
Java	78.0%	93.9%	**160**
Go	72.3%	94.9%	**138**
Rust	72.1%	81.0%	**171**
Windows	70.5%	80.1%	**065**

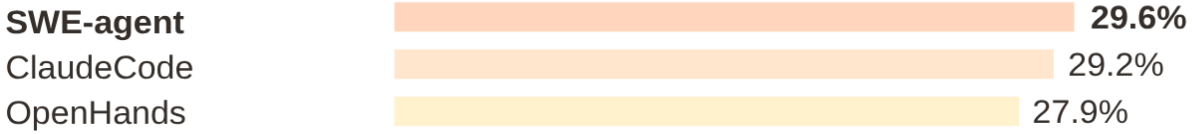
* Build/Management success rates are conditional on the preceding stage.
Task counts are retained instances after validation, which filters tasks without FAIL_TO_PASS test cases.

Benchmarking Result:
Benchmarking LM agents on our SWE-bench-Live/MultiLang and SWE-bench-Live/Windows

LM success rate averaged across all the agent harnesses and benchmark splits



Agent Harness success rate averaged across all the LMs and benchmark splits



Best model–harness combinations by benchmark split

Split	Best combination	Success
Java	**GPT-5.5 + SWE-agent**	**50.0%**
JS/TS	GPT-5.5 + SWE-agent	47.5%
C#	GPT-5.5/Deepseek-v4 + SWE-agent	44.9%
Go	Claude-4.5 + ClaudeCode	43.8%
C/C++	Claude-4.5 + OpenHands/SWE-agent	43.8%
Rust	Gemini-3 + SWE-agent	37.5%
Windows	Claude-4.5 + Win-Agent	30.0%

Insights:
1. LM performance difference is much more significant than harness performance difference;
2. Rust is the hardest language; Windows is the hardest OS;
3. Different LMs master different subsets of languages.

[SWE-bench-Live Leaderboard] (swe-bench-live.github.io)
20+ third-party contributors have contributed their LM agent success rates to our leaderboard!

Thank you!

RepoLaunch Source Code:
github.com/microsoft/RepoLaunch

SWE-bench-Live Dataset:
huggingface.co/collections/SWE-bench-Live/swe-bench-live

Correspondence:
Kenan Li kenan.li.24@ucl.ac.uk
Qirui Jin qiruijin@gatech.edu

Acknowledgements:
Project supervisors and resource providers:
Xin Zhang, Bowen Li, Chaoyun Zhang, Yu Kang

Project website: