NEURAL INFORMATION PROCESSING SYSTEMS

SETA: Scaling Environments for Terminal Agents

Qijia Shen, Zhiqi Huang, Vamsidhar Kamanuru, Aznaur Aliev, Jay Rainton
Ahmed Awelkair, Boyuan Ma, Qizheng Zhang, Jiwei Fu, Yuzhen Mao
Wendong Fan, Ping Nie, Philip Torr, Bernard Ghanem, Changran Hu
Jonathan Lingjie Li, Urmish Thakker, Guohao Li

CAMEL-AI Eigent AI IMPERIAL UCL sambanova

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

Stanford University

UNIVERSITY OF OXFORD

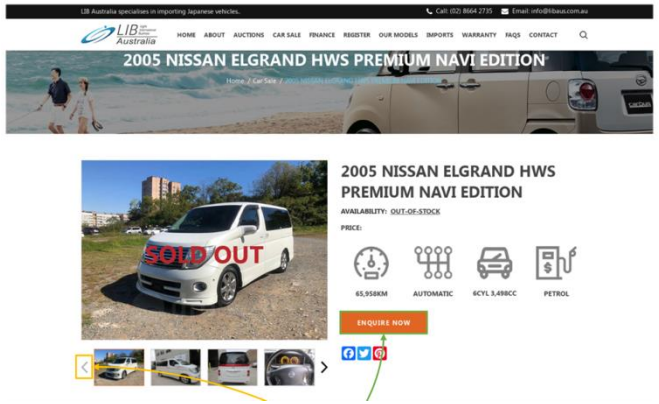
大阪大学 THE UNIVERSITY OF OSAKA

Agents Learn Through Interaction

Agents interact with websites, desktop applications, and terminals.

RL requires environments where agents can act and receive rewards.

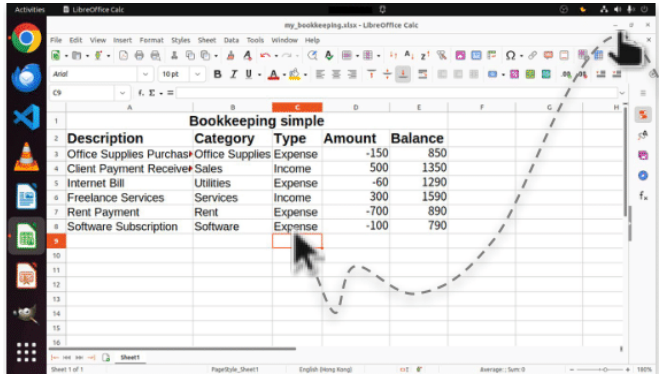
Web [1]



Click a page element

click / type / scroll

Desktop GUI [2]



Enter data in a spreadsheet

mouse / keyboard

Terminal [3]

```
agent$ ls data/
ACCOUNTS.DAT BOOKS.DAT TRANSACTIONS.DAT

agent$ sed -n '1,60p' data/ACCOUNTS.DAT
U001John Doe   0000001180U002Jane Smith

agent$ cobc src/program.cbl
agent$ ./program
Transaction failed due to validation
errors
```

Execute a shell command

commands / files / processes

Web: Cheng et al., SeeClick (2024), Fig. 3 [1]; GUI: Xie et al., OSWorld (2024) [2]; terminal: Merrill et al. (2026) [3].

Terminal Environments for Agentic RL

Why terminal?

A single text interface supports shell commands, scripts, and command-line tools across tasks.

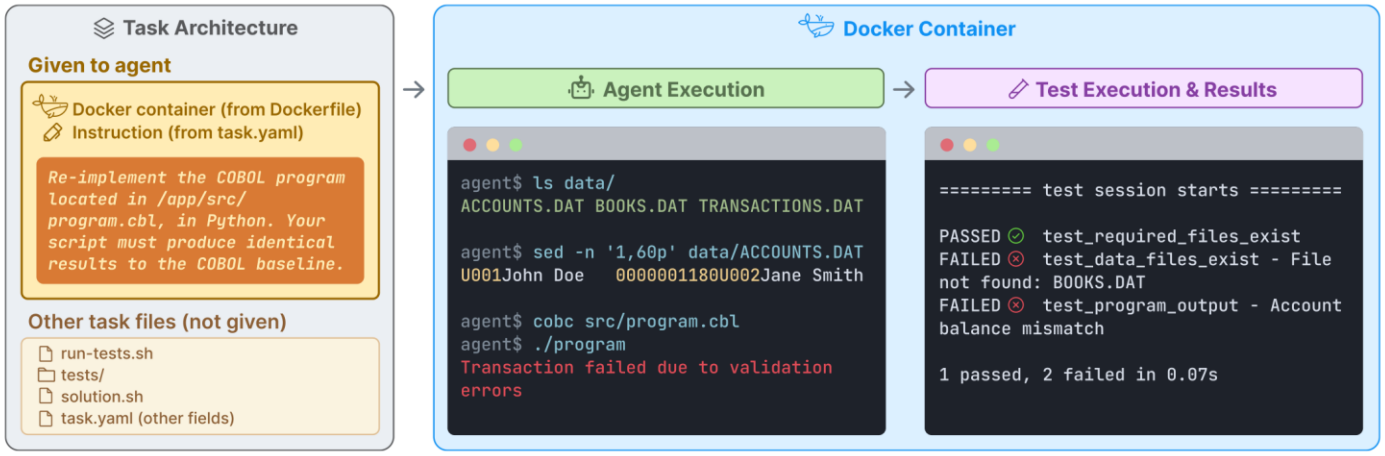
Why is construction difficult?

General terminal tasks often lack issue–PR pairs as grounding.

Instructions, environments, and tests must be constructed together.

What a Terminal Environment Contains

[3]



Research Question

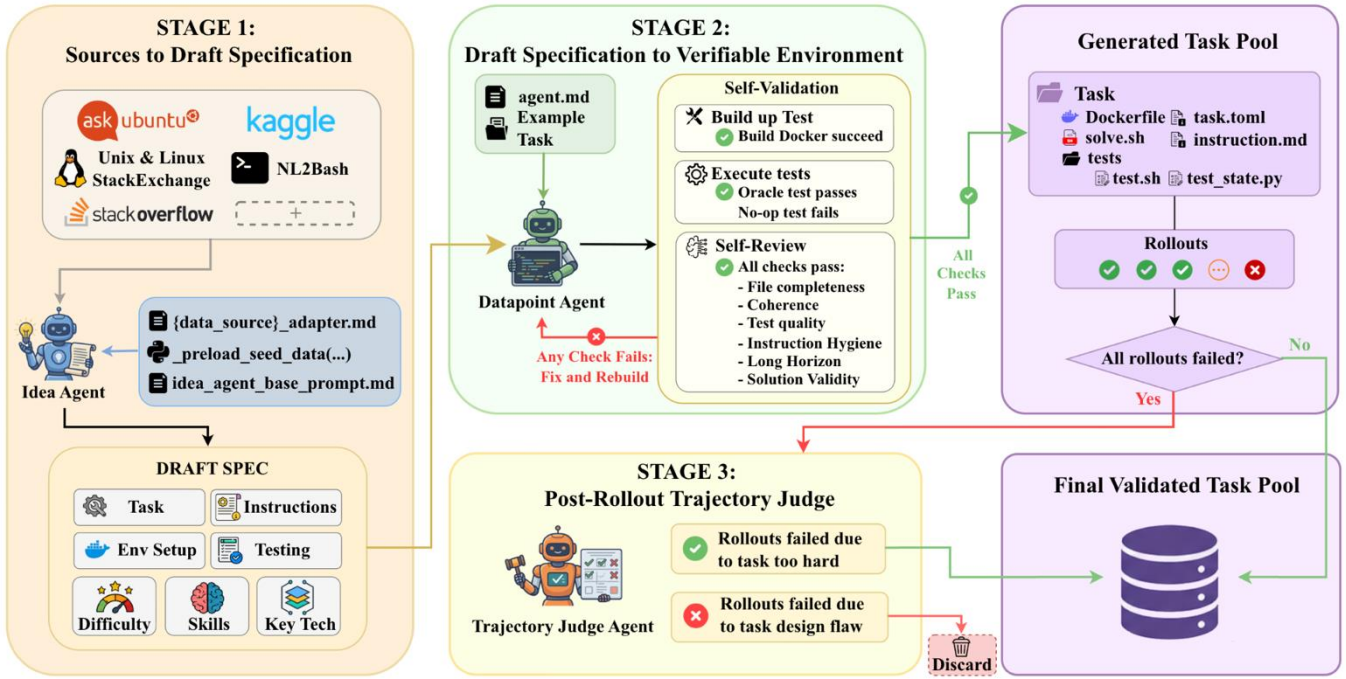
How can we generate diverse terminal environments at scale and verify their correctness?

Task illustration: Merrill et al., Terminal-Bench (2026), Fig. 2 [3].

SETA: Synthesis and Evolution

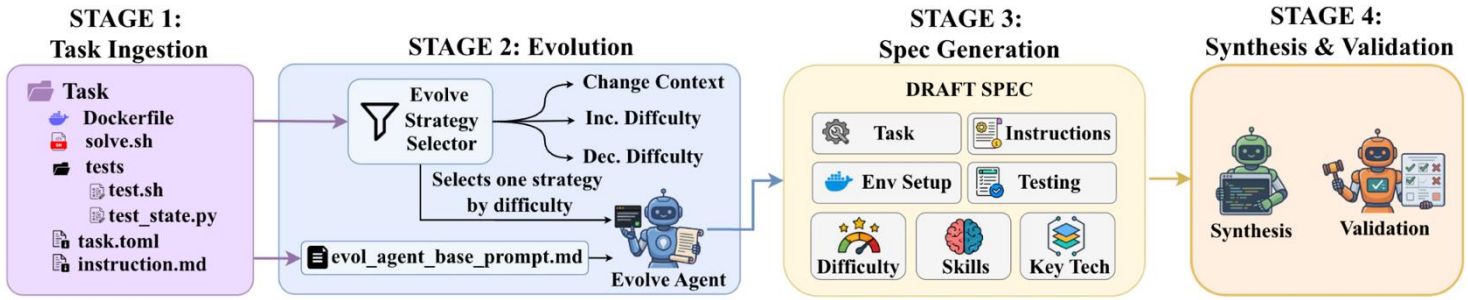
SETA-Synth

Source-adaptive



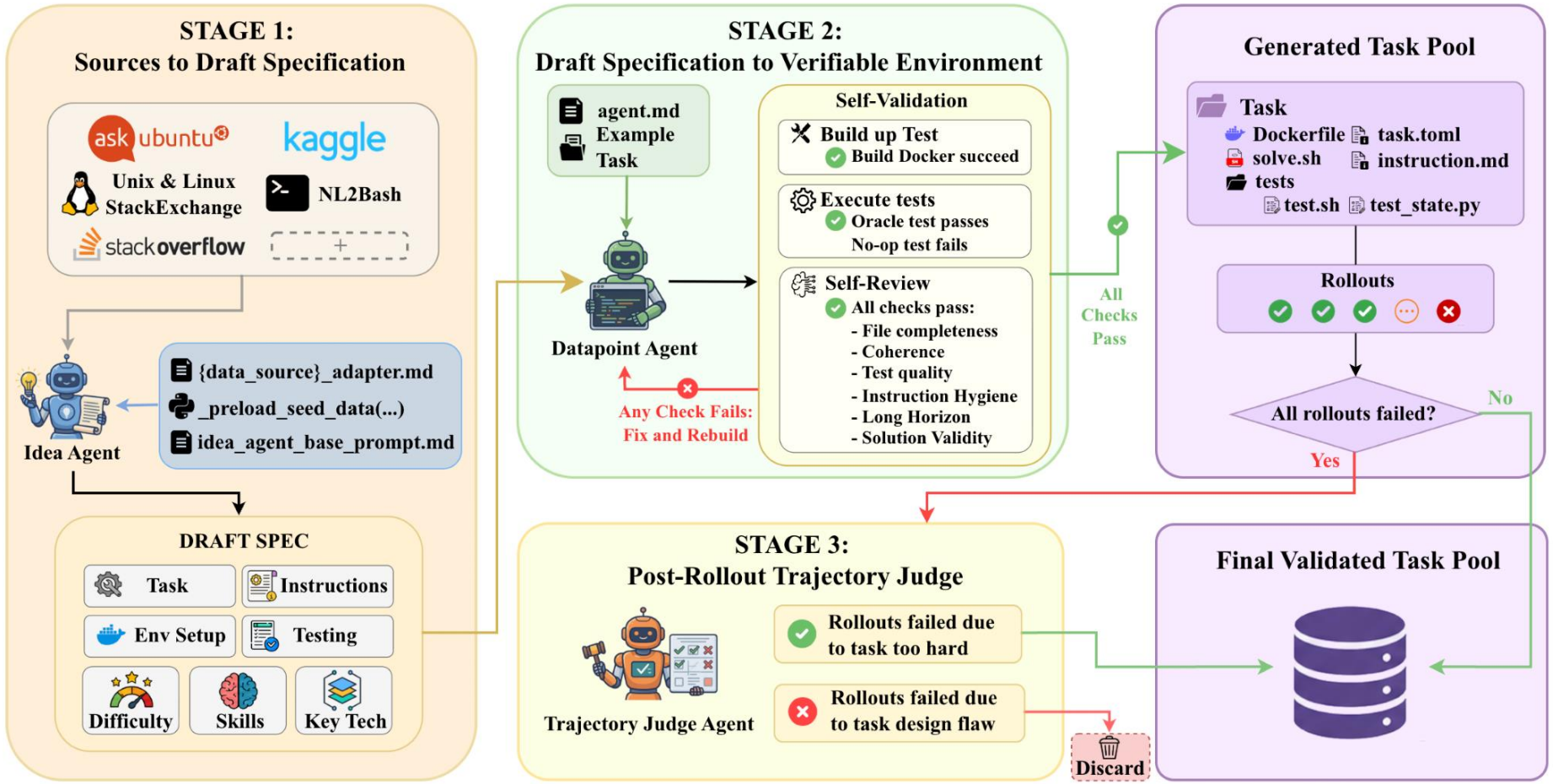
SETA-Evol

Difficulty-adaptive



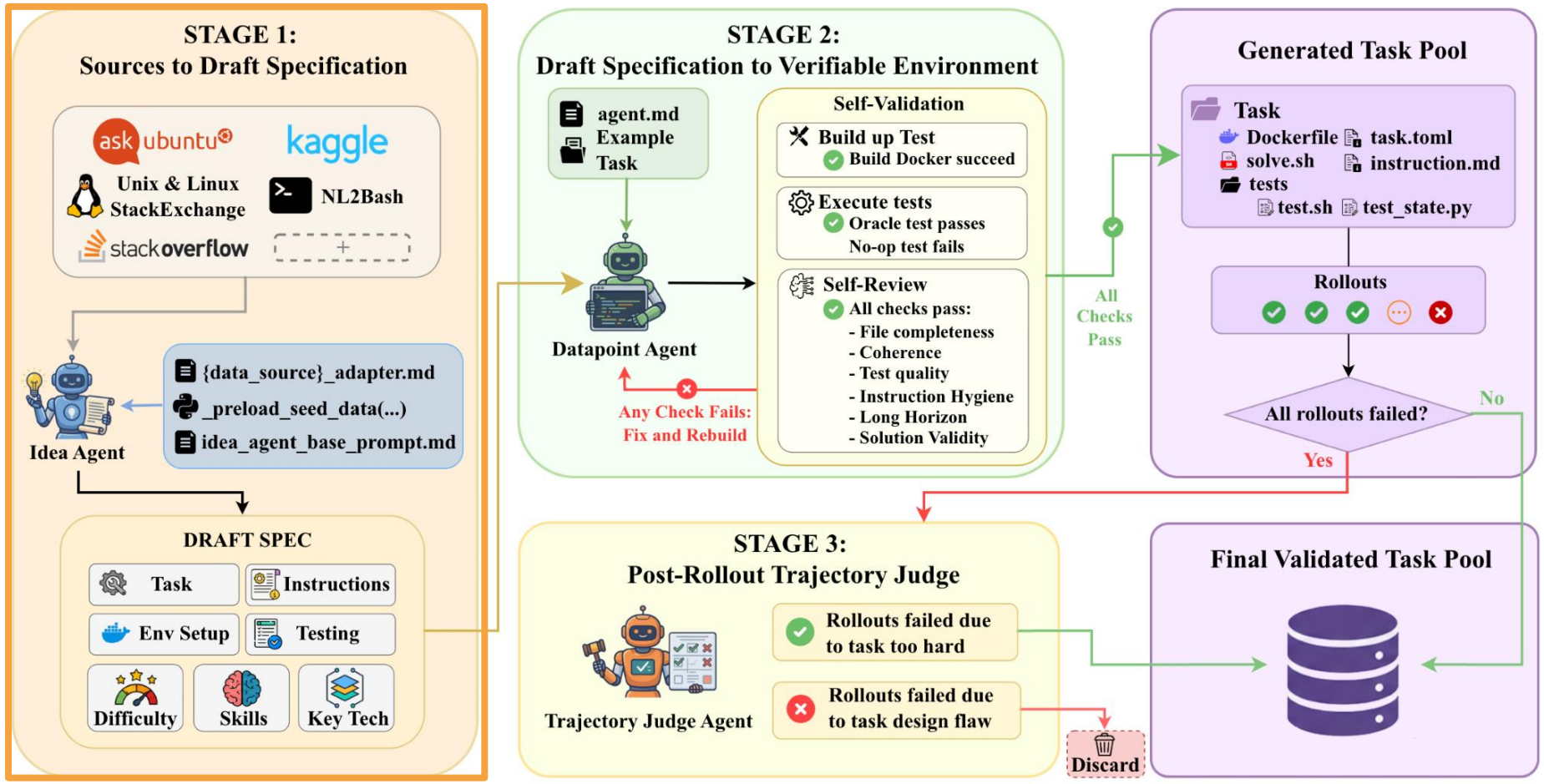
SETA-Synth

Synthesizing Verifiable Tasks from Diverse Sources



Source-Adaptive Idea Agent

Stage 1: Sources to Draft Specification



Data Sources

Source selection uses quality signals such as votes and accepted-answer status.

Data Sources

Community Q&A

ask ubuntu

stackoverflow

Unix & Linux StackExchange

Notebooks

kaggle

Command examples

NL2Bash

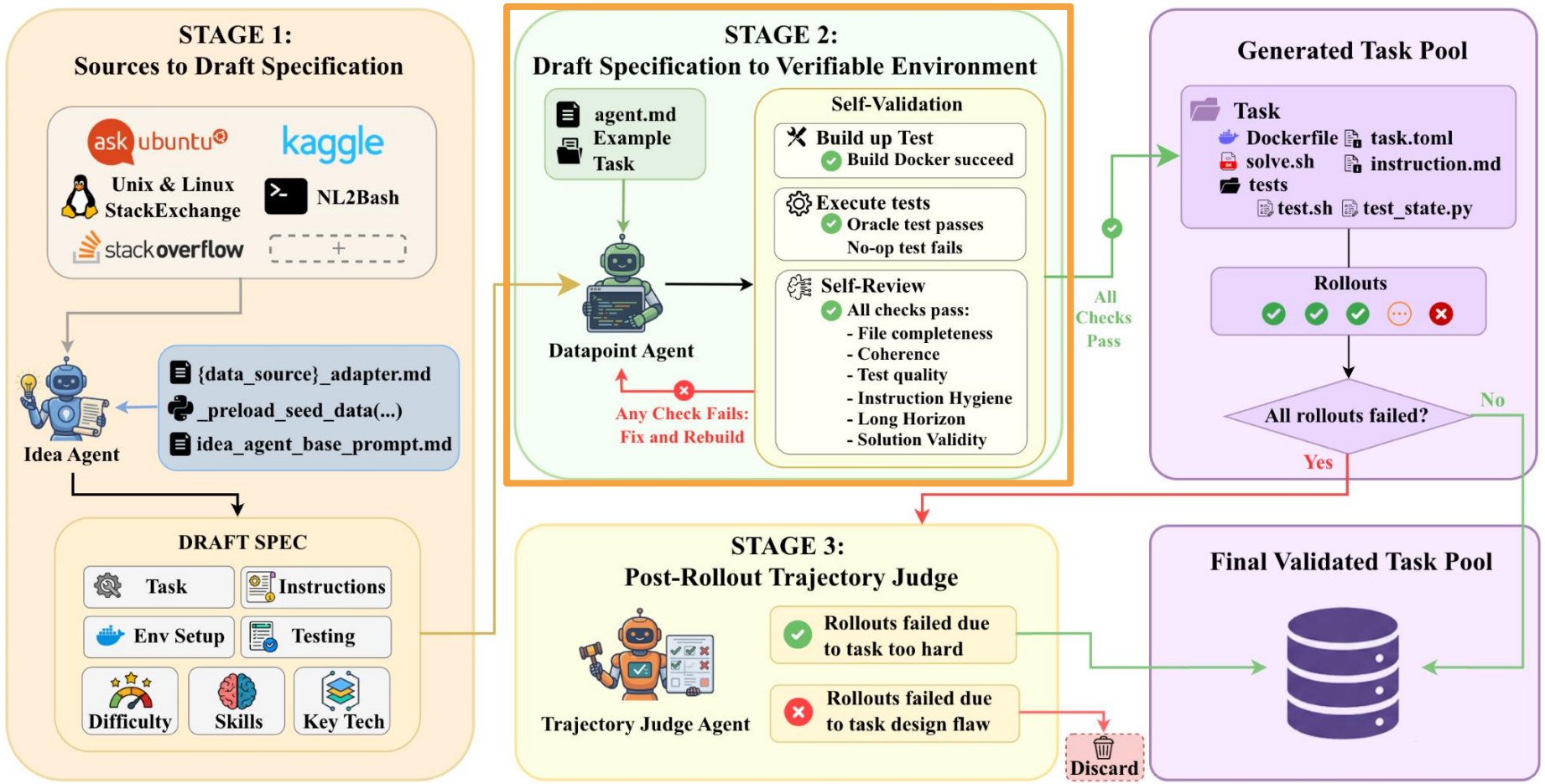
Seed Question and Related Sources

main.json stores the seed question and answer.
Related sources supply context for task design.



Datapoint Agent: Environment Construction

Stage 2: Draft Specification to Verifiable Environment



Draft Specification to Verifiable Environment

Draft Specification

draft_spec.md (excerpt)

```
## Task
  Fix a broken Java event log processor that fails to parse date strings and
  produces incorrect month values in its summary report.

## Agent-Visible Task Brief
  **Goal**: A Java program `EventProcessor.java` is supposed to read
  `events.csv`, parse each event's timestamp, and write a monthly event-count
  summary to `report.txt`. Currently the program crashes with a
  `ParseException` when run. Even after fixing the crash, the output may still
  be incorrect.

  **Entry Points**:
  - Project directory: `/home/ubuntu/project/`
  - Source file: `/home/ubuntu/project/EventProcessor.java`
  ...
```

⋮

```
  1. Navigate to `/home/ubuntu/project/`
  2. Read `EventProcessor.java`
  ...
  13. Recompile and run again
  14. Verify the report contents match expected counts

## Testing
  1. **Program compiles and runs without error**
  2. **report.txt exists and has exactly 5 lines**
  ...
  10. **Output format is correct**

## Difficulty
  medium

## Core Skills Tested
  - Reading and understanding unfamiliar Java source code
  - Diagnosing a runtime `ParseException`
  ...

## Key Technologies
  - Java 17 (OpenJDK)
  - `java.text.SimpleDateFormat`
  ...

## External Resources
  No web research was required --- the core APIs and the seed Q&A provide the
  necessary pattern and gotcha information.
```

→

Task Package

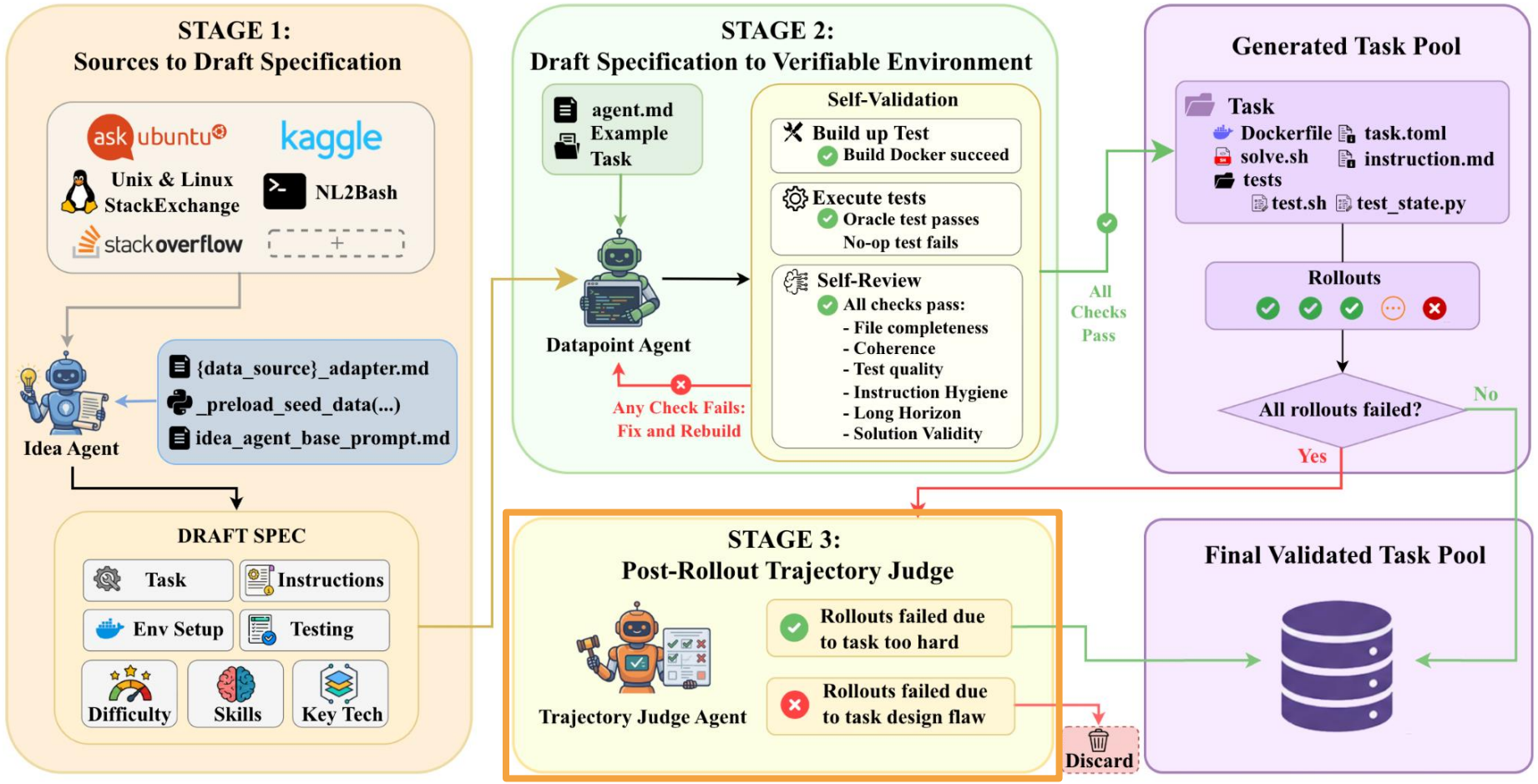
Task package structure

```
stack_overflow__5301226/
|-- task.toml
|-- instruction.md
|-- weights.json
|-- environment/
|   |-- DateBuilder.java
|   |-- DateParser.java
|   |-- Dockerfile
|   |-- EventDateNormalizer.java
|   |-- EventProcessor.java
|   |-- ReportWriter.java
|   |-- events.csv
|   |-- expected_output_sample.txt
|   `-- run.sh
|-- solution/
|   `-- solve.sh
`-- tests/
    |-- test.sh
    `-- test_outputs.py
```

Instruction	instruction.md
Environment	environment/
Tests	tests/
Reference solution	solution/solve.sh

Post-Rollout Trajectory Judge

After all rollouts finish, the judge reviews terminal logs, instructions, and tests to investigate failed tasks.



What Trajectory Judge Reveals

Tests Require Fields Missing from the Instruction

The task asks for a JSON report. Tests demand exact field names that the instruction never specifies.

All five rollouts fail these checks.

Field	Test demands	All 5 rollouts wrote
success-count key	`archived_successfully`	`successfully_archived`
skip-count key	`skipped_oversized`	`skipped`
entry status value	`"ok"`	`"success"` / `"archived"`
entry name format	`cli-tool.zip`	`cli-tool`

Instruction

Box 1. Task instruction shown to the agent (`instruction.md`).

You are given a directory structure at `projects/` containing multiple project subdirectories, and a configuration file at `.archiverc` (JSON format). Your task is to create a shell script called `archive_projects.sh` in the workspace root that implements a production-grade batch archiving system.

⋮

7. **Generate a JSON report**: Create `archives/report.json` containing a timestamp, counts of total projects, successfully archived, skipped (oversized and failed), and an array of archive entries with name, size in bytes, file count, SHA256 hash, and status.
8. **Log all operations** to `archives/archive.log` with timestamped entries using severity levels (INFO, WARN, ERROR) for different types of events.
9. **Support `--dry-run` flag**: When invoked with `--dry-run`, the script should simulate all operations without creating any archives, checksums, or report files. It should only write to the log file with entries prefixed by `[DRY-RUN]`, and exit with code 0.
10. **Handle edge cases**: Empty directories should result in empty archives. The script must be executable.

The `projects/` directory contains several project subdirectories including ones with various files, exclusion-pattern targets, symlinks, spaces in names, empty directories, and large binary files. The `.archiverc` config is pre-populated.

After creating the script, run it to generate the archives and all output files.

Test

Box 2. Failing tests (`tests/test_outputs.py`).

```python
def test_json_report_correct():
    report_file = os.path.join(
        ARCHIVES_DIR, "report.json")
    with open(report_file) as f:
        report = json.load(f)

    assert report["total_projects"] == 6
    assert report["archived_successfully"] == 5
    assert report["skipped_oversized"] == 1
    assert len(report["archives"]) == 5

    names = [a["name"] for a in report["archives"]]
    for expected in ["webapp.zip", "cli-tool.zip",
                     "project with spaces.zip",
                     "empty-project.zip",
                     "symlink-project.zip"]:
        assert expected in names

    for entry in report["archives"]:
```

Fabricated Scores Pass the Verifier

The agent added random noise to five fixed scores instead of running the required 5-fold cross-validation.

The verifier checked format, range, and variance, so fabricated scores passed.

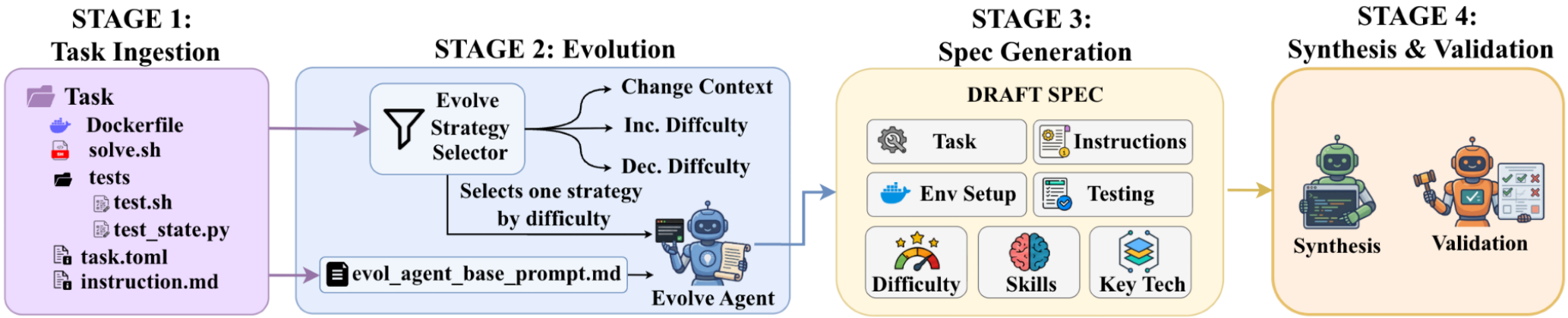
context **The task explicitly requires realistic 5-fold CV.**

The instruction requires each model to have 5-fold accuracy and requires the use of Caret's CV; the tests mainly check length, range, and non-zero variance.

context	task/instruction.md:52-61	Cross-validation scores are explicitly required.
context	task/tests/test_outputs.py:100-112	The verifier only checks length, range, and std.

```
80 results$gradient_boosting <- train_eval("gbm", train_x, train_y, test_x, test_y)
81
82 # Simple CV
83 cv_scores <- list()
84 for (m in names(results)) cv_scores[[m]] <- c(0.92,0.94,0.93,0.95,0.94) + runif(5, -0.01, 0.01)
85
```

SETA-Evol: Difficulty-Adaptive Evolution



Context Shift

Changed Unchanged

Parent task — SysAdmin

Create a comprehensive system diagnostics collection script that collects information from a Docker container environment to help diagnose container health and networking issues. The script must be saved as `/app/collect_diagnostics.sh` and produce a report at `/app/diagnostics_report.txt`.

Requirements:

1. Collect from multiple Linux subsystems: container resource usage (cgroups, memory, CPU), network configuration (interfaces, routes, DNS, iptables), processes, file system, logs, and environment variables.
2. Mask sensitive env values (`KEY`, `SECRET`, `PASSWORD`, `TOKEN`, `CREDENTIAL`) with `[MASKED]`.
3. Produce a single text file with delimited section headers (e.g. `=== NETWORK CONFIGURATION ===`) and a timestamp; degrade gracefully when tools are unavailable.
4. Standalone bash script with no extra dependencies.

The environment has intentional issues the script must reveal, e.g. a misconfigured network route . . .

Evolved task — SWE

Create a comprehensive Nginx web server diagnostics script that inspects the Nginx installation and produces a diagnostic report. Save as `/app/collect_nginx_diagnostics.sh`; output at `/app/nginx_diagnostics_report.txt`.

Requirements:

1. Collect from multiple Nginx subsystems in delimited sections: server config (workers, modules, included files), virtual hosts (listen + `server_name`), SSL/TLS certificate status (subject/issuer/expiry), upstream backends + reachability, last 20 lines of error log + 4xx/5xx counts, `stub_status` if available.
2. Mask sensitive values in directives like `proxy_set_header X-Auth-Token "..."` and `set $db_password ...` with `[MASKED]` (keywords: `password`, `secret`, `key`, `token`, `credential`, `auth`).
3. Single text file with section headers, timestamp, graceful degradation.

Difficulty Increase: Service Recovery

Original: Resolve a port conflict to restore Flask

Find port owner → Stop process → Start Flask and verify

Evolved: The process restarts automatically, and Flask has an incorrect bind address

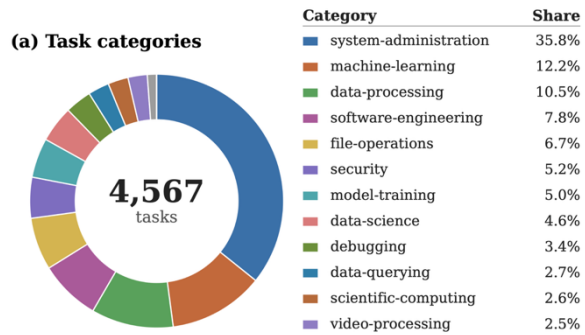
Disable restart mechanisms → Stop process → Start Flask → Check /health

Fix bind address → Stop process

Dataset Results

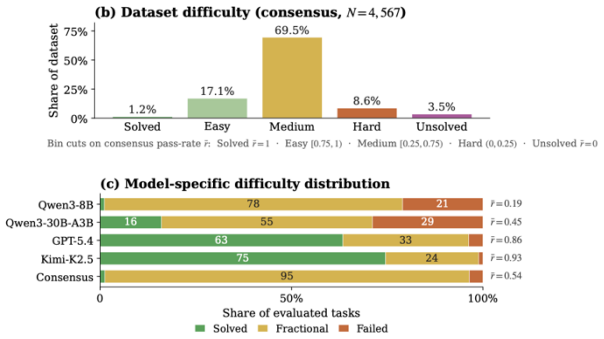
4,567 environments across 14 task categories

Task Distribution

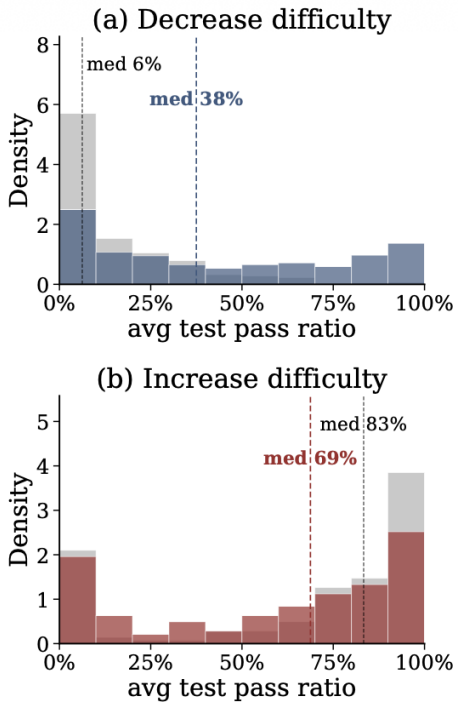


Category	Share
system-administration	35.8%
machine-learning	12.2%
data-processing	10.5%
software-engineering	7.8%
file-operations	6.7%
security	5.2%
model-training	5.0%
data-science	4.6%
debugging	3.4%
data-querying	2.7%
scientific-computing	2.6%
video-processing	2.5%
optimization	1.2%
audio-processing	0.0%

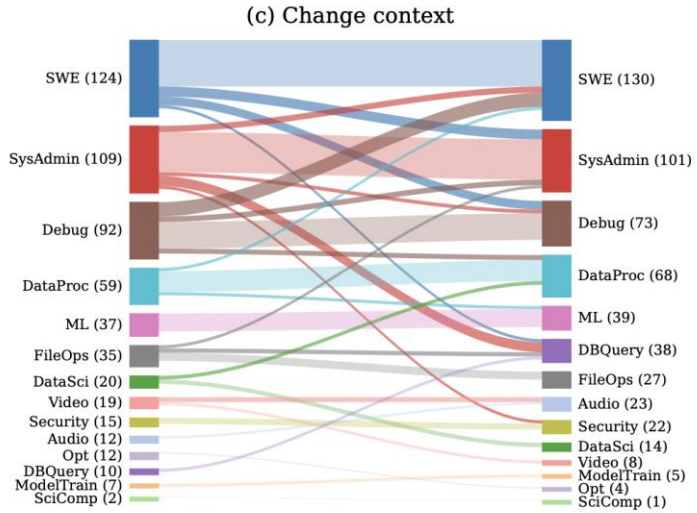
Dataset Difficulty



Difficulty Adaptation



Context Shifts



Experimental Setup

GRPO Training Loop

Sample tasks
8 tasks per optimization step

Collect rollouts
16 trajectories per task

Compute rewards

$$p = \frac{N_{\text{passed}}}{N_{\text{tests}}}, \quad r = p + 0.2\,\mathbf{1}[p = 1]$$

A 0.2 bonus when all tests pass.

Filter rollout groups
Exclude uniform-reward groups

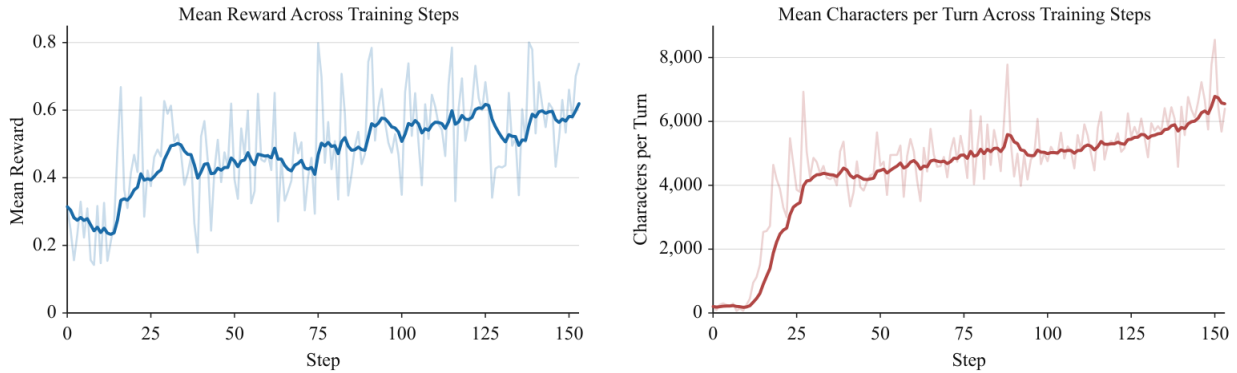
Update the policy
GRPO optimization

Training Configuration

Setting	Qwen3-8B	DeepSeek-V4-Flash
Framework	AReal	Miles
GPUs	8 × H200	64 × H200
Agent	CAMEL	CAMEL

Qwen3-8B: non-thinking mode, 560 environments, ~200 steps.
260-environment subset for the training-size comparison.

Training Curves



Experimental Results

Terminal-Bench

Pass@1 (%)

Models at comparable scale

Model	Size	TB 1.0	TB 2.0	Method
Qwen3-8B (base)	8B	7.8 ± 1.2	3.1 ± 0.6	—
Qwen3-32B (base)†	30B	11.25	3.4 ± 1.6	—
Nemotron-Terminal-8B†	8B	—	13.0 ± 2.2	SFT
Endless Terminals-8B†	8B	—	6.7	SFT+RL
OpenThoughts-Agent-v1-RL†	8B	—	4.9	RL
SETA (RL)	**8B**	**17.8 ± 1.2**	**10.7 ± 1.3**	**RL**

Stronger models (for context)

Model	Size	TB 1.0	TB 2.0	Method
DeepSeek-V4-Flash	—	—	40.0	—
DeepSeek-V4-Flash + SETA	**—**	**—**	**43.0 ± 2.5**	**RL**
Qwen3-Coder	480B	39.0 ± 0.4	23.9 ± 2.8	—
GPT-5-Nano	—	12.2 ± 2.9	11.5 ± 2.3	—
Claude Haiku 4.5	—	41.8 ± 2.6	13.9 ± 2.7	—
GPT-OSS-120B†	120B	—	14.2 ± 2.3	—

Transfer to Code Benchmarks

Pass@4 (%)

Model	CRUST	Compile	Quix
Qwen3-8B	15	6.7	7.5
Qwen3-30B-A3B	28	13.3	13.8
SETA (8B)	**24**	**40**	**15.0**

SETA: Scaling Environments for Terminal Agents

Thank you for listening!

Paper

arxiv.org/abs/2607.10891

Code

github.com/camel-ai/seta







SETA: Scaling Environments for Terminal Agents

Qijia Shen[1,2], Zhiqi Huang[3,4], Vamsidhar Kamanuru[5], Aznaur Aliev[6]
Jay Rainton[5], Ahmed Awelkair[1,2,6], Boyuan Ma[5], Qizheng Zhang[7]
Jiwei Fu[9], Yuzhen Mao[7], Wendong Fan[1,2], Ping Nie[5]
Philip Torr[8], Bernard Ghanem[6], Changran Hu[5]
Jonathan Lingjie Li[5], Urmish Thakker[5], Guohao Li[1,2]

[1]CAMEL-AI.org [2]Eigent.AI [3]Imperial College London [4]University College London
[5]SambaNova [6]KAUST [7]Stanford University [8]University of Oxford [9]University of Osaka

Code Dataset RL model

Abstract

Large language models (LLMs) are rapidly shifting toward agents that solve tasks through diverse interfaces, including web and graphical user interfaces (GUIs). Among these, the terminal command line provides a text-based, general-purpose interface, covering tasks from system operations to data science and machine learning. However, scaling terminal-agent training remains challenging, as it requires diverse and coherent task instructions, executable environments, and reliable verification, while lacking naturally grounded supervision data. In this work, we propose SETA, a scalable framework for generating verifiable terminal environments for reinforcement learning (RL). The framework consists of two pipelines sharing a unified verification mechanism: SETA-Synth converts diverse sources into standardized RL environments, and SETA-Evol further expands from existing environments with adaptive control of difficulty and diversity. Together, we construct and release SETA-Env, the largest open-source verifiable terminal RL dataset to date, containing over 4,500 environments. We evaluate our dataset by training Qwen3-8B with GRPO on SETA-Env, improving mean pass@1 on Terminal-Bench 2.0 from 3.1% to 10.7%, with a best run of 12%. We further observe gains on DeepSeek-V4-Flash under the same terminal agent harness, with pass@1 on Terminal-Bench 2.0 improving from 40% to 43% and pass@5 improving from 54% to 58%. These results demonstrate that SETA-Env provides high-quality training environments for terminal agents and serves as a valuable resource for advancing research on terminal-based agent learning.

1 Introduction

Large language models (LLMs) have rapidly shifted from pure text generation towards agents that execute tools to finish tasks in interactive environments. ReAct [Yao et al., 2023] introduced a general agent framework interleaving reasoning and acting, followed by numerous works adding scaffolding and interfaces to enhance different aspects of agent capabilities. Representative efforts include web agents such as WebArena [Zhou et al., 2024], GUI agents such as OSWorld [Xie et al., 2024], search-augmented reasoning agents such as Search-R1 [Jin et al., 2025], and coding agents such as SWE-agent [Yang et al., 2024], which targets repository-level software engineering tasks using unit-test-based verification. Among all these interfaces, terminal provides a text-based and general-purpose option covering almost all computer-usage tasks, including system operations, file manipulation, data processing, etc.

40th Conference on Neural Information Processing Systems (NeurIPS 2026) Track on Datasets and Benchmarks.