# Large-Scale Terminal Agentic Trajectory Generation from Dockerized Environments

**Siwei Wu** [1 2 3 *] **Yizhi Li** [1 2 3 *] **Yuyang Song** [2 4 *] **Wei Zhang** [5] **Yang Wang** [1] **Riza Batista-Navarro** [1]
**Xian Yang** [1] **Mingjie Tang** [4] **Bryan Dai** [2] **Jian Yang** [5] **Chenghua Lin** [1]

# Why Terminal Agents?

Codex

Claude Code


1. CLI agents such as Codex and Claude Code have already transformed the way most programmers work and have achieved tremendous success in real-world applications. Therefore, it is crucial to evaluate the ability of different agents to solve a wide range of tasks in CLI/terminal environments.

2. TerminalBench was specifically proposed to evaluate the ability of different agents to solve complex tasks across various domains in terminal environments. Our goal is also to improve the capabilities of agentic models in terminal environments.

# Motivation

## A terminal training instance is only useful when it has:

**1**

**Executable environment**

a runnable Docker/container state

**2**

**Environment-aligned task**

instructions grounded in the actual repo

**3**

**Executable validation**

code that checks side effects and final state

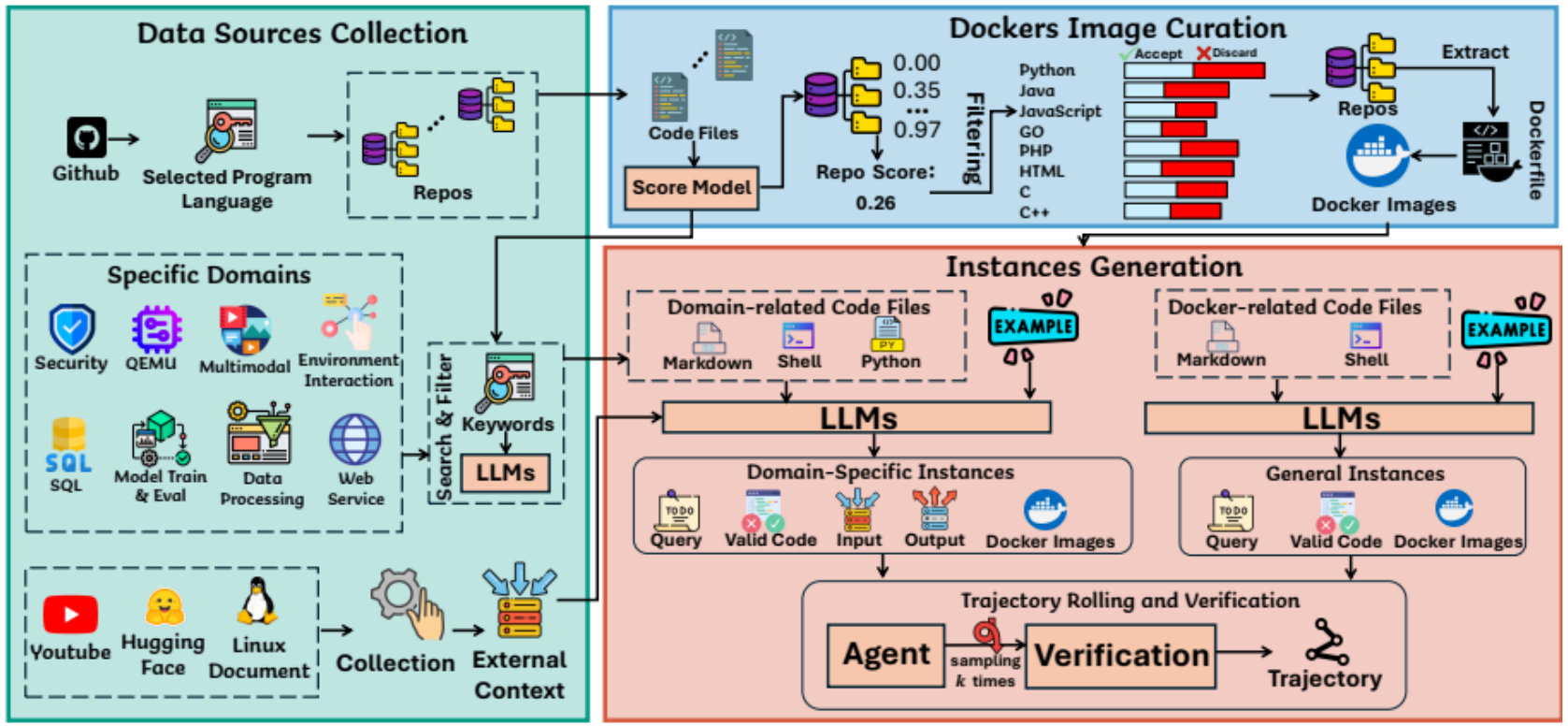
**4**

**Successful trajectory**

multi-turn agent interaction that passes validation

1. **most methods optimize only part of this equation.** Scale without strong execution grounding, or execution grounding without enough environment diversity.
2. **Terminal agents need scale, execution grounding, and verification together, and prior data construction has a trade-off:**
    - LLM-simulated or rule-based trajectories scale, but are weakly tied to runtime effects.
    - SWE-style executable environments are realistic, but scale and diversity are limited.
    - Terminal tasks especially depend on filesystem state, dependency resolution, and tool side effects.

# Framework of TerminalTraj




1. **Data Sources Collection:** We collect large-scale GitHub repositories, domain-specific files, and external resources to provide diverse signals for executable terminal tasks.
2. **Docker Image Curation:** We filter high-quality repositories with a scoring model and build them into reproducible Docker environments for agent interaction.
3. **Instance Generation:** We generate Docker-aligned task queries and executable validation code, then roll out agents to collect verified terminal trajectories

# Pipeline Overview

Start from open-source repositories, then filter for buildable execution environments.

**899K**
candidate GitHub repositories

**196K**
high-quality repositories

**32K**
Docker images constructed

**8**
programming languages

## Model-based repository scoring

- ScoreModel estimates code completeness and executability.
- Low-score repositories are discarded before expensive Docker builds.
- This avoids relying only on stars, commits, or hand-crafted heuristics.

*Table 1.* Statistics of Collected Repositories, Docker Images, and Trajectories Across Different Programming Langu

| Category | ALL | Programming Language | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | GO | C++ | C | HTML | Java | JavaScript | PHP | Python |
| Collected Repos | 899,741 | 184,989 | 24,161 | 26,341 | 42,209 | 60,253 | 129,044 | 102,857 | 329,887 |
| High-Quality Repos | 196,051 | 34,655 | 7,518 | 5,316 | 8,817 | 13,887 | 23,031 | 18,098 | 78,729 |
| Docker Images | 32,325 | 7,400 | 1,424 | 854 | 1,946 | 2,775 | 5,400 | 1,689 | 10,837 |

**Repository scoring makes environment construction both scalable and predictable.**

# Step 2: generate tasks and executable validation

LLMs reverse-engineer tasks from code, documentation, and scripts.

## Input signals

- Markdown files: intent and usage
- Shell scripts: executable logic
- Dockerfiles: dependencies and runtime context
- Domain context: images, data files, web links, checkpoints

## LLM output

- Task query: user-facing instruction
- Valid code: pytest-style executable checker
- Required inputs/outputs
- Docker image binding

```
def test_result_exists():
    assert
os.path.exists("output.json")

def test_output_schema():
    data =
json.load(open("output.json")
)
    assert "accuracy" in data
```

### Why validation code matters

**It checks the final environment state, not whether the model wrote the same commands as a reference solution.**

# Step 3: roll out agents and keep only verified trajectories

The agent interacts with the Docker environment through a real terminal loop.

```
Observation: pytest fails because dependency
is missing
Thought: install the missing package and
rerun tests
$ pip install pyyaml
$ python scripts/convert.py data.csv
output.json
$ pytest validation/test_task.py -q
2 passed
```

## Quality gate

- Run agent in the target Docker image.
- Record multi-turn stdout, stderr, commands, and self-corrections.
- Execute validation code after the agent stops.
- Only passing trajectories enter the final dataset.

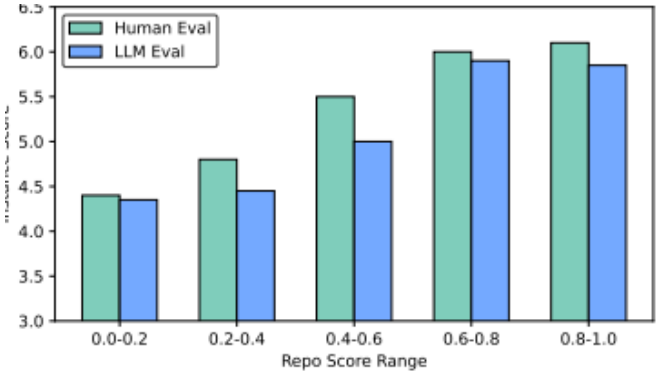
**Final dataset: 50,733 verified trajectories**

# Dataset Scale

The pipeline is large, difficult, and quality-controlled.

*Table 1.* Statistics of Collected Repositories, Docker Images, and Trajectories Across Different Programming Langu

| Category | ALL | Programming Language | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | GO | C++ | C | HTML | Java | JavaScript | PHP | Python |
| Collected Repos | 899,741 | 184,989 | 24,161 | 26,341 | 42,209 | 60,253 | 129,044 | 102,857 | 329,887 |
| High-Quality Repos | 196,051 | 34,655 | 7,518 | 5,316 | 8,817 | 13,887 | 23,031 | 18,098 | 78,729 |
| Docker Images | 32,325 | 7,400 | 1,424 | 854 | 1,946 | 2,775 | 5,400 | 1,689 | 10,837 |




## Scale

- 1,030,695 generated instances from 32,325 Docker images.
- 50,733 final trajectories after execution-based filtering.
- A low verified rate (~4.9%) reflects difficult tasks and strict verification.

## Quality signal

- Higher repository scores correlate with higher human and LLM instance quality.
- Repository-level quality carries over to task-level supervision.

# Main result: strong gains on TerminalBench

SFT on TerminalTraj consistently improves Qwen2.5-Coder backbones.

| | | | | |
|---|---|---|---|---|
| Qwen3-Coder-30B-A3B-Instruct | 30B | Terminus-2 | 23.80 | 14.60 |
| Qwen3-30B-A3B-Nex-N1 | 30B | OpenHands | 25.00 | 8.30† |
| Qwen3-32B | 32B | OpenHands | 11.25 | 3.40 |
| Qwen3-32B-Nex-N1 | 32B | OpenHands | 28.75 | 16.70† |
| Qwen2.5-Coder-7B-Instruct (**Backbone**) | 7B | Terminus-2 | 6.25 | 0.00 |
| Qwen2.5-Coder-14B-Instruct (**Backbone**) | 14B | Terminus-2 | 6.25 | 1.18 |
| Qwen2.5-Coder-32B-Instruct (**Backbone**) | 32B | Terminus-2 | 5.00 | 4.49 |
| **TerminalTraj-7B (Ours)** | 7B | Terminus-2 | 23.01 | 10.10 |
| **TerminalTraj-14B (Ours)** | 14B | Terminus-2 | 28.91 | 19.10 |
| **TerminalTraj-32B (Ours)** | 32B | Terminus-2 | **35.30** | **22.00** |

Focused excerpt from Table 2: Qwen2.5-Coder backbones vs. TerminalTraj models.

- 7B: 6.25 -> 23.01 on TB 1.0
- 14B: 6.25 -> 28.91 on TB 1.0
- 32B: 5.00 -> 35.30 on TB 1.0

## TerminalTraj-32B

**35.30%**

TerminalBench 1.0

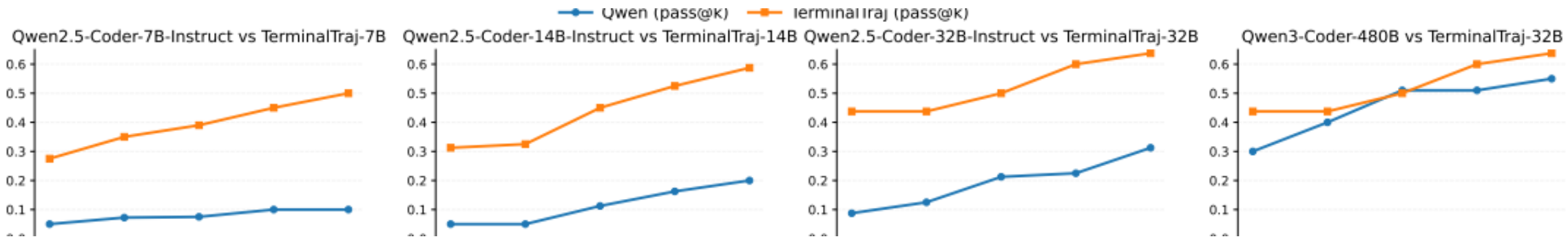
**22.00%**

TerminalBench 2.0

- +20 absolute points on TB 1.0 over the 32B backbone.
- +10 absolute points on TB 2.0 over the 32B backbone.
- Strongest model under 100B parameters in the comparison.
- Comparable to Qwen3-Coder-480B with far fewer parameters.

# TerminalTraj improves test-time scaling

Better training data helps models convert more samples into more solved tasks.




## Observation

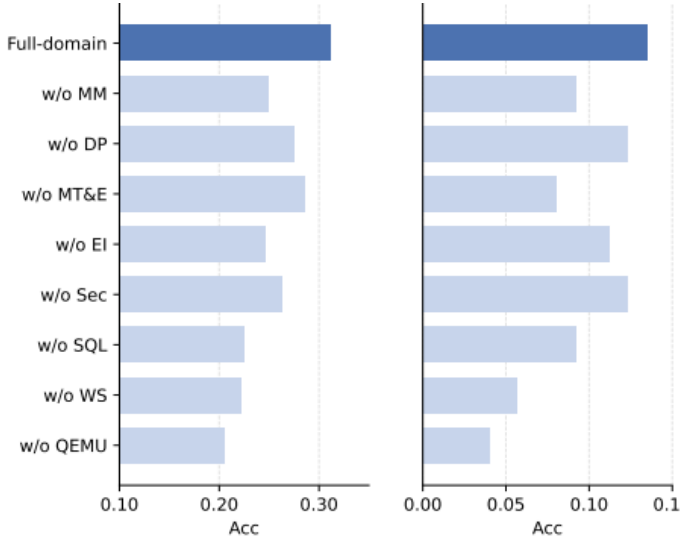
- Backbone models show weak or flat pass@k curves.
- TerminalTraj models keep improving as k increases from 1 to 16.
- TerminalTraj-32B reaches about 63% pass@16.

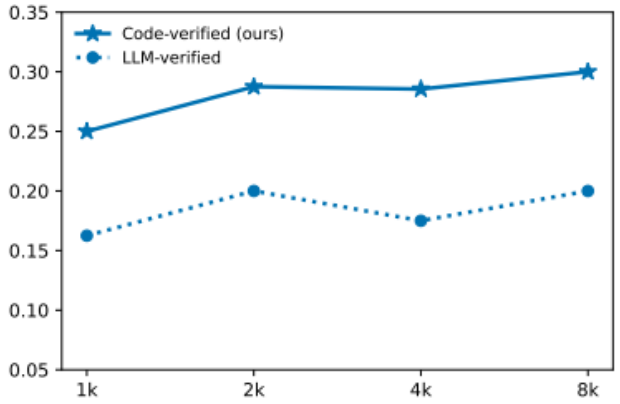
## Interpretation

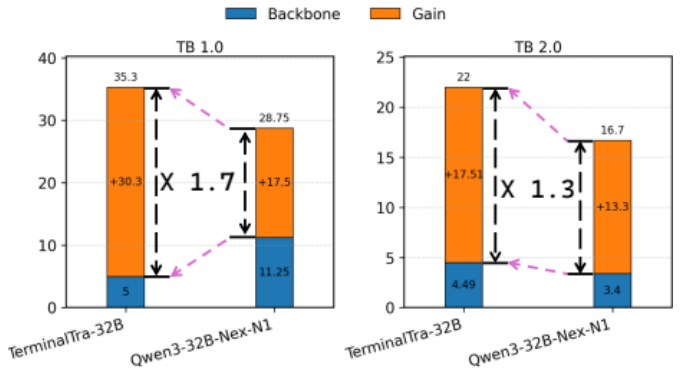
**Execution-grounded trajectories teach the model a richer distribution of valid terminal strategies, so additional inference compute is more useful.**

# Ablations: why does it work?

Diversity and executable verification both matter.




## Domain diversity

Removing any domain hurts; WS and QEMU are especially important.

## Code verification

Code-verified trajectories outperform LLM-verified trajectories at every data scale.

## Dataset quality

TerminalTraj achieves larger gains than Nex-N1 despite fewer trajectories and a weaker backbone.

# Thanks!