CS 329Z: Engineering AI Agents

Data Selection & Quality

Diyi Yang, Michael Ryan, John Yang

Announcements

- HW2 released!
  - Task due 11/23
  - Peer Reviews due 12/2
- Please sign up for oral exams!! Registration already over due.
- Midpoint demo + report due this Friday (11/6)

Recap: Lecture 10

1. **Agents need more than text.** Tasks, environments, trajectories.
2. **Humans are realistic but costly.** Demos are hard to scale.
3. **Mined traces are cheap.** Code, tutorials, and videos convert.
4. **Synthesis scales cheaply.** Models write steps and build tasks.
5. **Simulators give endless practice.** Their errors get learned too.
6. **Data teaches by accident.** Reward hacks, harness overfit.

Roadmap

Data Selection & Quality

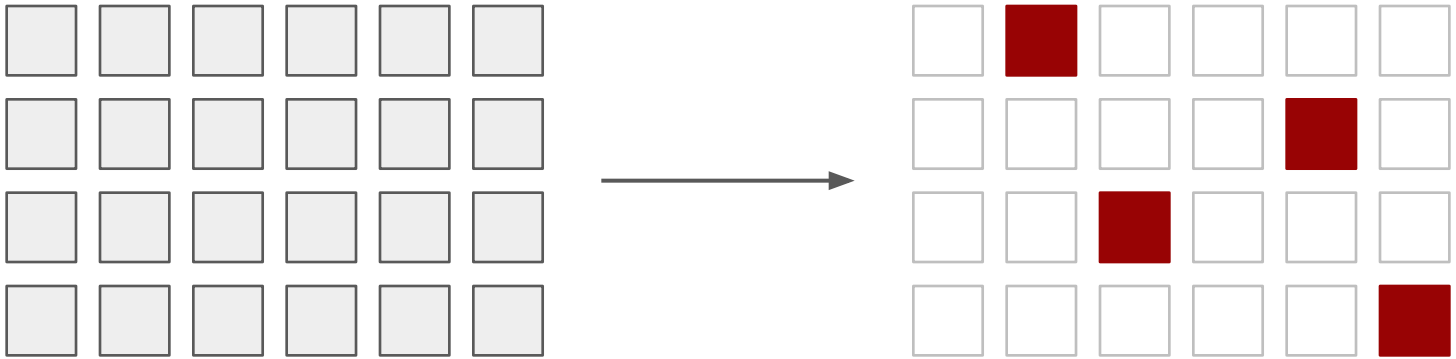
- Motivation [10 min]
- Which tasks are worth training on? [25 min]
- Which trajectories are worth imitating? [15 min]
- More data, or better data? [25 min]
- Takeaways [5 min]

tl;dr: Making agent tasks and trajectories is now cheap; deciding what to keep is the hard part. In this lecture, we'll cover which tasks to train on, which trajectories to imitate, and whether more data or better data wins.

Motivation

Motivation

Making tasks and trajectories is now cheap. Deciding what to keep is hard.



Make tasks + trajectories
Now cheap

Decide what to keep
The hard part

Motivation

CWM generated 3M agent trajectories, then decided which ones to keep.

3M

agent trajectories from 10.2k Docker images of 3.15k repos

→

Dropped near-duplicates

Within each repo, kept trajectories have pairwise Jaccard similarity below 0.5.

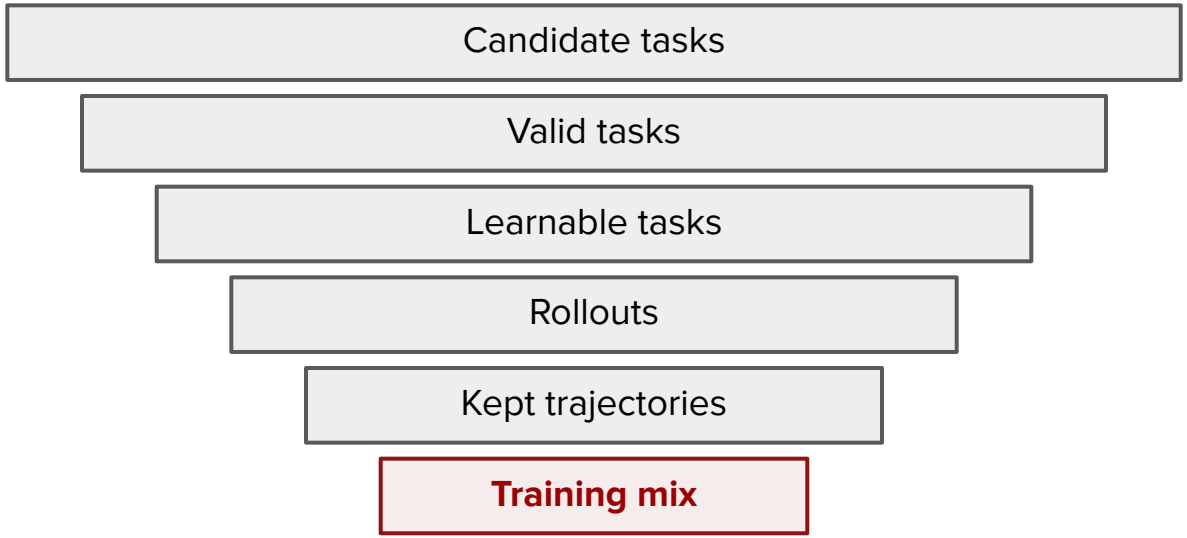
Kept failed attempts

The goal is a world model of code environments, so they don't filter by success.

Source: Meta FAIR CodeGen Team, CWM (2025)

Motivation

Every stage of the pipeline throws data away.



Section 2
Which tasks?

Section 3
Which trajectories?

Section 4
More or better data?

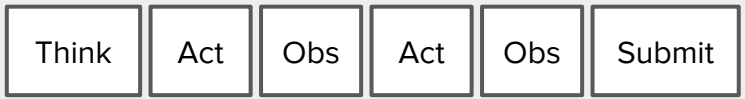
Motivation

We select two things: trajectories for SFT, environments for RL.

🎞️ Trajectory

Section 3

A recording of an agent solving a task (issue text + sandbox).



Then perform *supervised fine-tuning (SFT)* on traces. LM *imitates* behavior.

📦 Environment

Section 2

A task (issue text + sandbox) along with a mechanism for verifying success.



Then perform *reinforcement learning (RL)* against tasks. LM practices ReAct.

Motivation

Training data goes bad in three ways.



Wrong signal

Broken verifiers and hacked successes.

Sections 2–3



No signal

Tasks too easy or too hard to learn from.

Section 2



Redundant signal

More of the same.

Section 4

Motivation

Much evidence that given same base model, post-train data makes big difference.

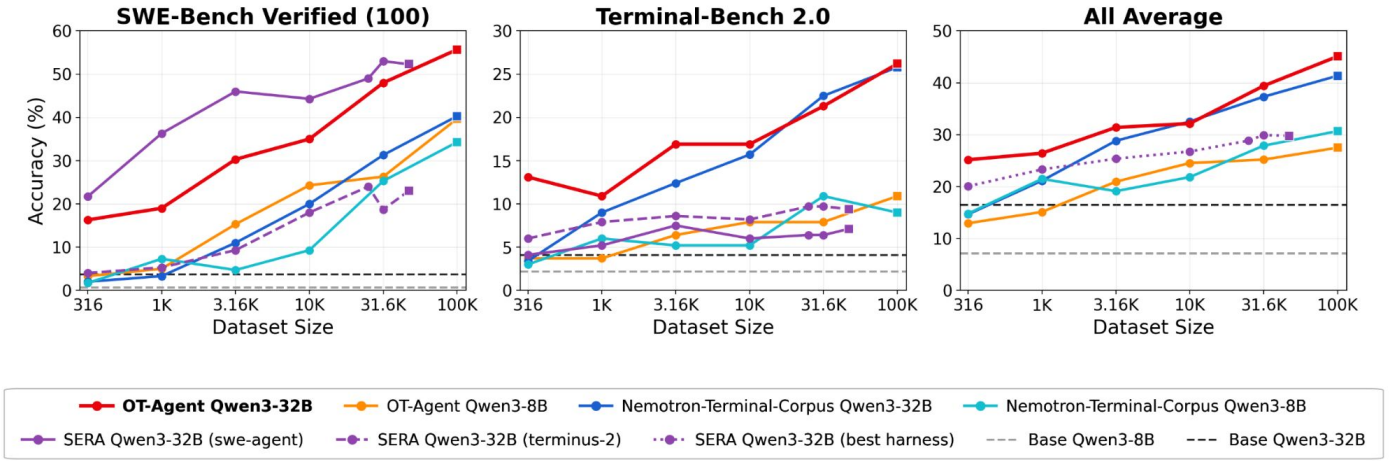


Figure 1: **The OpenThoughts-Agent-SFT dataset leads to SotA performance on Terminal-Bench 2.0 and an 100-subset of SWE-Bench Verified at large dataset scales.** The all-benchmark average is over the seven benchmarks reported in Table 1. Additional notes on SERA below.[1]

From OT-Agent paper (Raoof 2026), given same budget of SFT traces (x-axis), selecting the most effective subset leads to better performance across multiple benchmarks.

Motivation

The open tension: is less more, or is more more?

Less is more

LIMI: 78 hand-curated trajectories beat 10,000.

vs

More is more

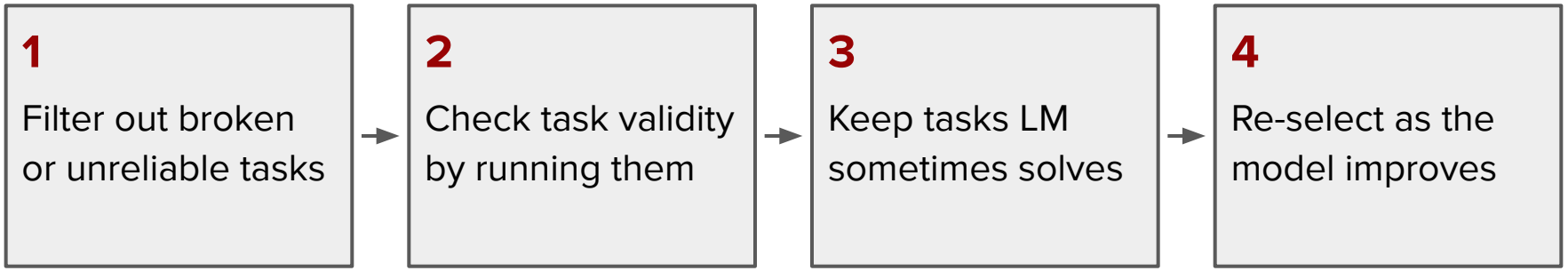
Orchard: at scale, data size matters more than how you pick it.

Section 4 looks at when each one holds.

Which tasks are worth training on?

Which tasks are worth training on?

Let’s use the mental model of thinking of the task selection process as a loop!



SWE-bench Verified

In sklearn-14520, the test requires an exact deprecation message that was only decided in PR discussion the agent never sees.

```
Copy param ignored in TfidfVectorizer
I was playing with vectorizers and I found this:

https://github.com/scikit-learn/scikit-learn/blob/ae16319626e2ca6ca0e54d4a5b83f73f817232aa/sklearn/feature_extraction/text.py#L1669

However that parameter is not used later in the method.

Here `copy=False` is used:

https://github.com/scikit-learn/scikit-learn/blob/ae16319626e2ca6ca0e54d4a5b83f73f817232aa/sklearn/feature_extraction/text.py#L1692

Is there anything I am missing?
```

```python
def test_tfidf_vectorizer_deprecationwarning():
    msg = ("'copy' param is unused and has been deprecated since "
           "version 0.22. Backward compatibility for 'copy' will "
           "be removed in 0.24.")
    with pytest.warns(DeprecationWarning, match=msg):
        tv = TfidfVectorizer()
        train_data = JUNK_FOOD_DOCS
        tv.fit(train_data)
        tv.transform(train_data, copy=True)
```


Source: OpenAI, Introducing SWE-bench Verified (2024)

SWE-bench Verified

93 Python developers annotated 1,699 random SWE-bench samples. Each sample was labeled by 3 annotators, keeping the highest-severity label.

Are the tasks well-specified? −

Evaluated models are expected to generate a patch given the *problem statement* and codebase. If the *problem statement* is poorly specified, it can be significantly harder, or in some cases impossible, to generate a patch that solves the problem.

We label the problem statement with these 4 possible labels:

- 0: The issue is well-specified and it is clear what is required for a successful solution.
- 1: There are some blanks to fill in about the issue, but there is a sensible interpretation of what is required for a successful solution.
- 2: The issue is vague and there is room for ambiguity. It is unclear what a successful solution would look like.
- 3: It is almost impossible to understand what you are being asked to do without further information.

How valid are the evaluation criteria? −

To evaluate a model's proposed solution, the model's generated patch is applied to the codebase and then the FAIL_TO_PASS and PASS_TO_PASS unit tests are executed. If the patch can be applied successfully and all of the tests pass, then the proposed solution is considered to have successfully resolved the issue.

We check if the FAIL_TO_PASS tests might fail even when a valid solution is provided, resulting in a false negative.

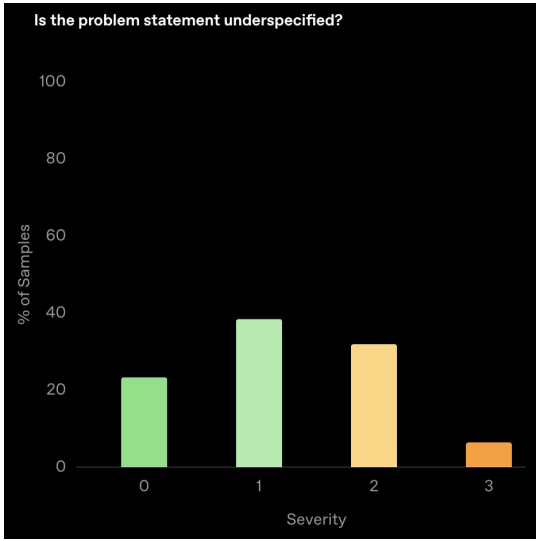
We label the tests with these 4 possible labels:

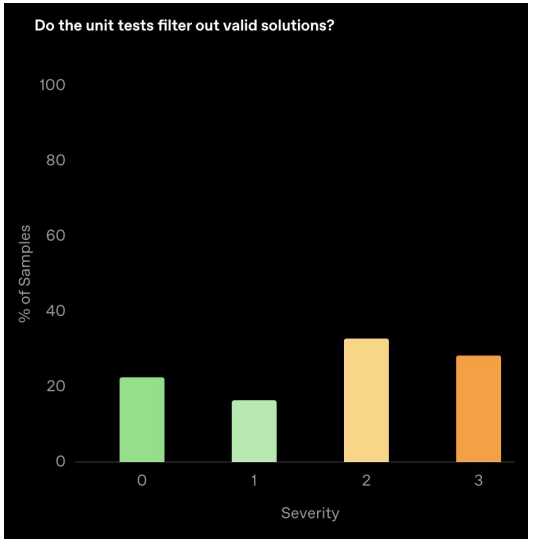
- 0: The tests perfectly cover all possible solutions.
- 1: The tests cover the majority of correct solutions, however some unusual solutions may be missed.
- 2: The tests work but some perfectly reasonable solutions may be missed by the tests.
- 3: The tests are too narrow/broad or they look for something different than what the issue is about.

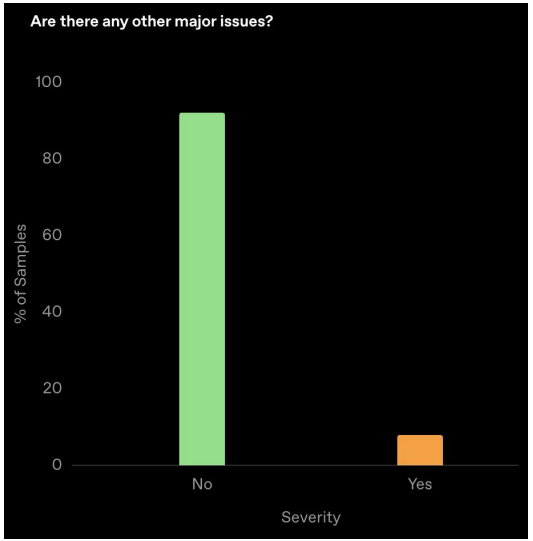
Source: OpenAI, Introducing SWE-bench Verified (2024)

SWE-bench Verified

38.3% flagged for underspecified problem statements, 61.1% for unfair unit tests; 68.3% filtered out overall.







Source: [OpenAI, Introducing SWE-bench Verified (2024)]

SWE-bench Verified

Broken tasks hide real ability. In training, they give wrong rewards.

16%

GPT-4o on original SWE-bench



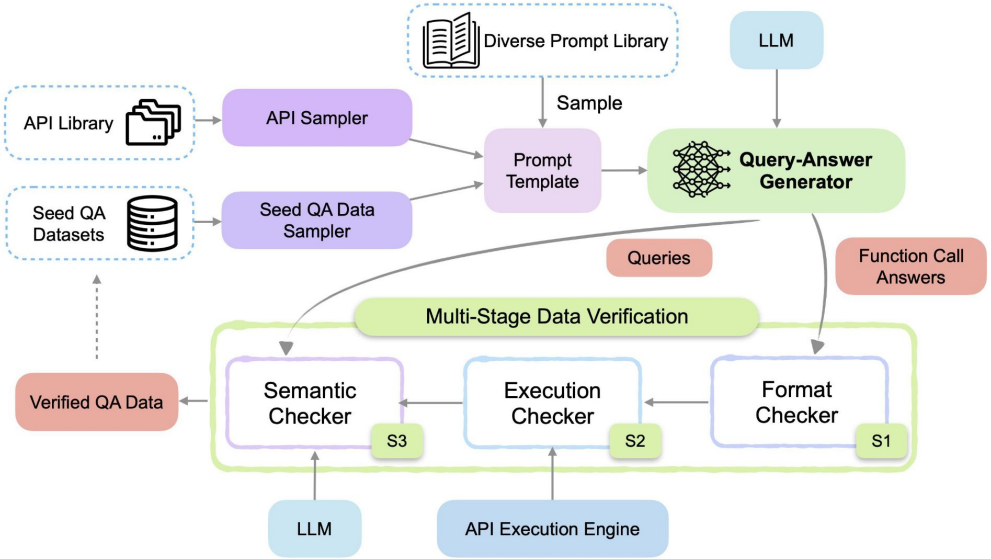
33.2%

GPT-4o on SWE-bench Verified

Source: OpenAI, Introducing SWE-bench Verified (2024)

APIGen

Human review doesn't scale, so APIGen automates it: a format check, an execution check, then an LLM judging whether results match the query's intent.



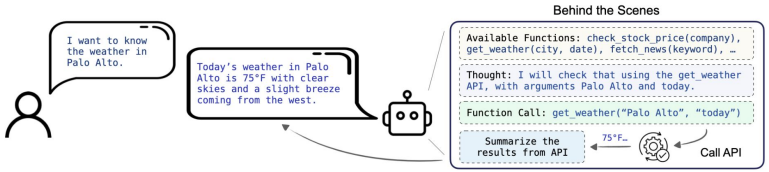
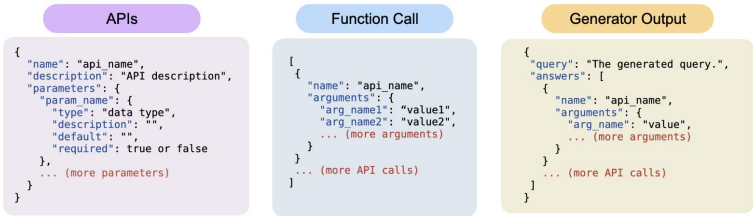


Figure 1: Workflow of an LLM-based function-calling agent.



Source: [Liu et al., APIGen (2024)]

CodeMidas

CodeMidas builds coding RL environments from source code alone: no issues, PRs, commits, or existing tests.

From MiMo model team.

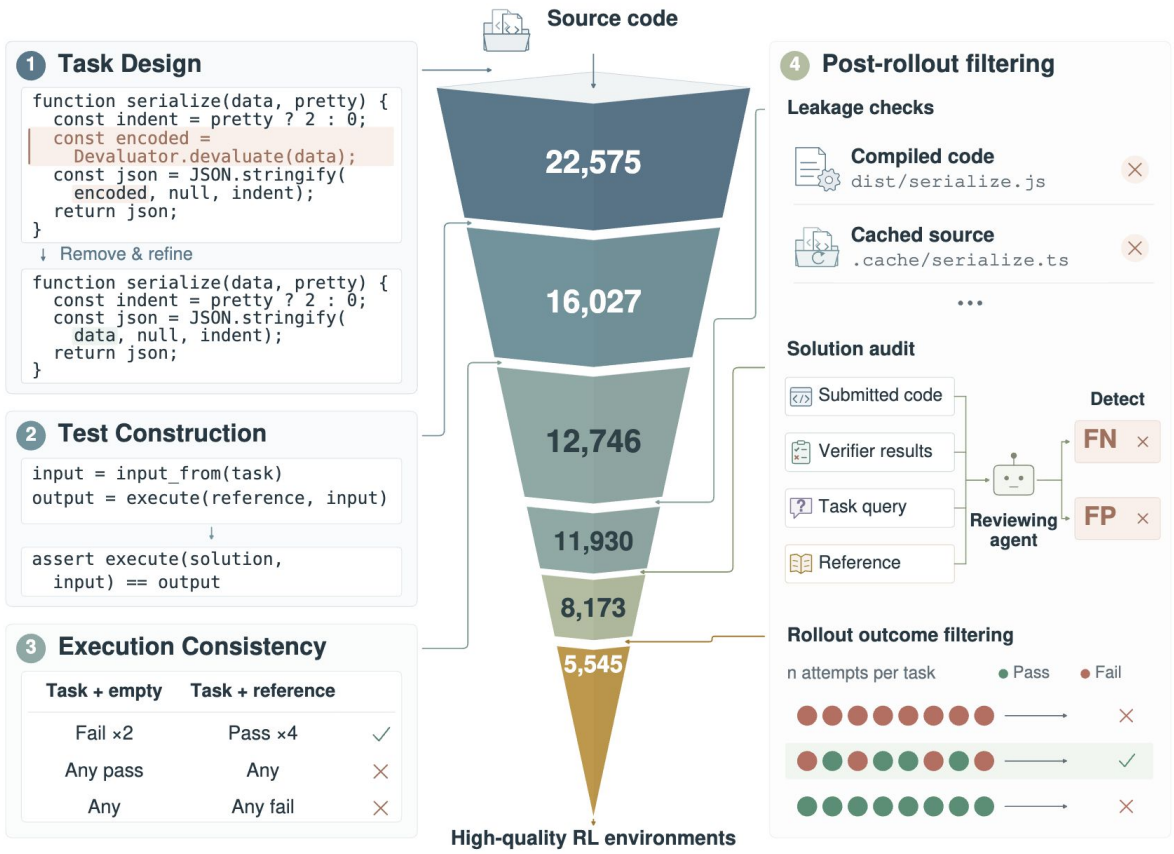


Figure 1 CodeMidas overview. Four modules cover task design, test construction, execution consistency, and post-rollout filtering. The pyramid shows retained task counts respectly.

Source: Ye et al., CodeMidas (2026)

CodeMidas

Tests should accept any correct solution.

Too strict: sklearn-14520

```
msg = ("'copy' param is unused and has"
       " been deprecated since version"
       " 0.22. Backward compatibility for"
       " 'copy' will be removed in 0.24.")
with pytest.warns(DeprecationWarning,
                  match=msg):
    tv.transform(train_data, copy=True)
```

The exact message came from PR discussion.

CodeMidas: only what the task states

```
# Task: raise ValueError on bad input.
# The message wording is not specified.

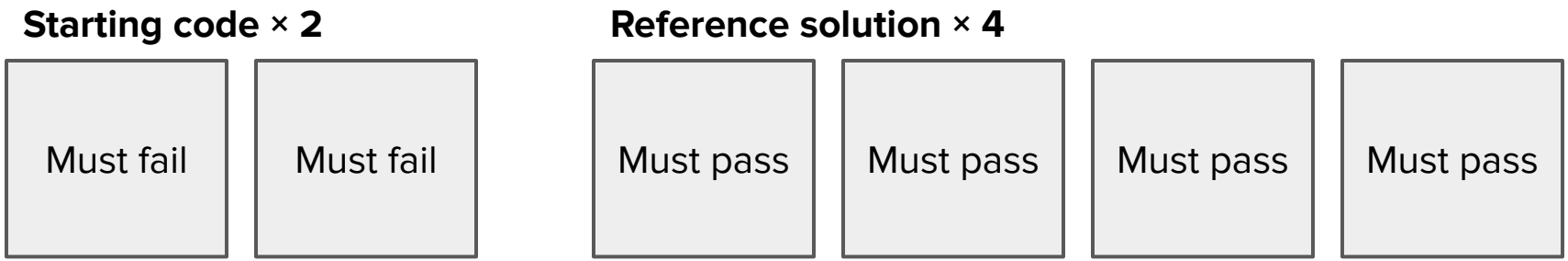
with pytest.raises(ValueError):
    parse_config(bad_input)
```

Illustrative test, following the paper's rule.

Source: Ye et al., CodeMidas (2026)

CodeMidas

Each task runs in six fresh containers.

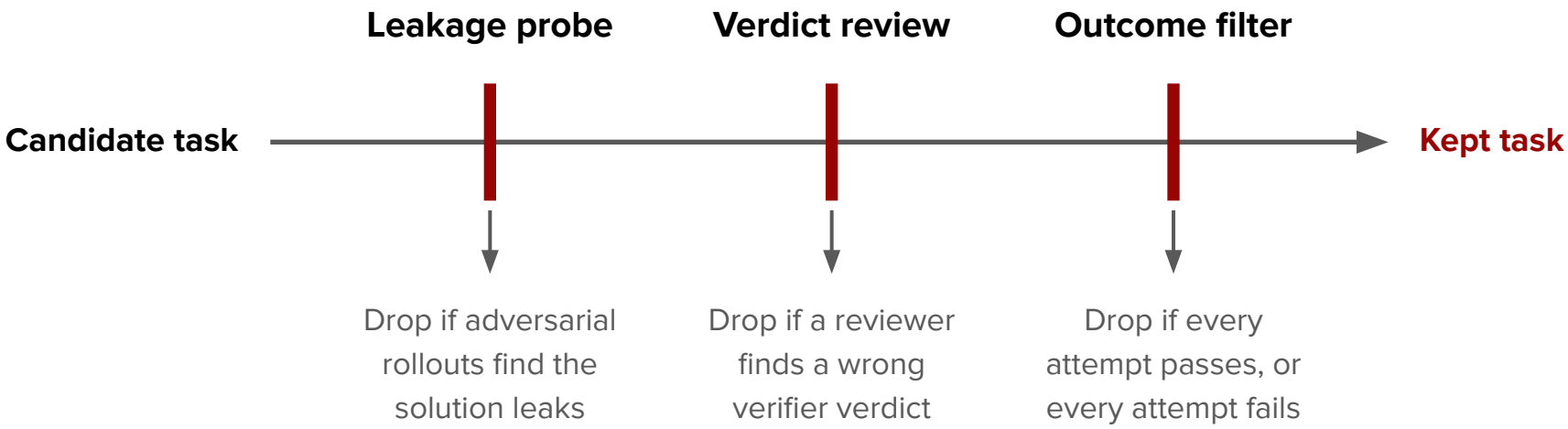


A task is kept only if all six runs come out as expected.

Source: Ye et al., CodeMidas (2026)

CodeMidas

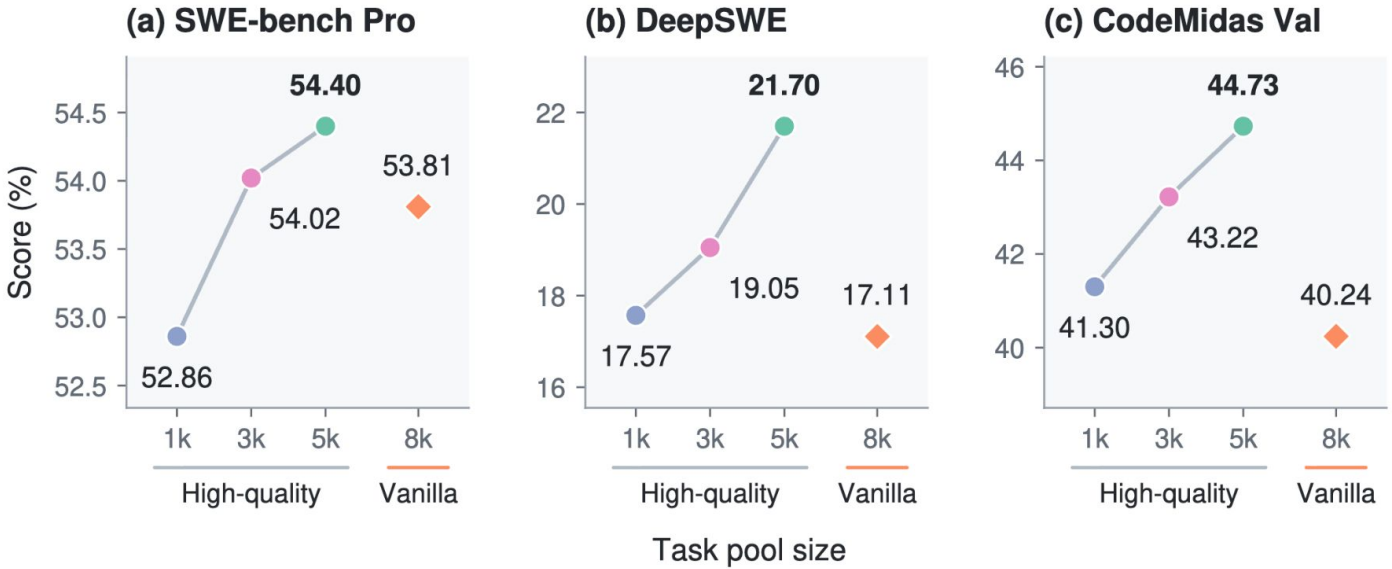
After rollouts, three more filters.



Source: Ye et al., CodeMidas (2026)

CodeMidas

Even the filtered 3k subset beats the unfiltered 8k sample on all three evaluations. Full 5k beats 8k by 0.59 / 4.59 / 4.49 points (SWE-bench Pro / DeepSWE / Internal).

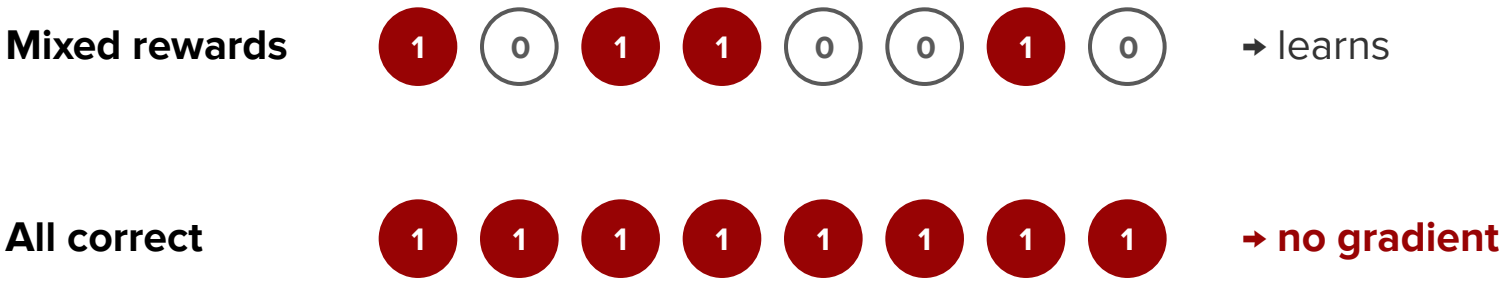


Source: Ye et al., CodeMidas (2026)

DAPO

If every rollout in a group gets the same reward, RL learns nothing.

advantage = (reward − group mean) / group std

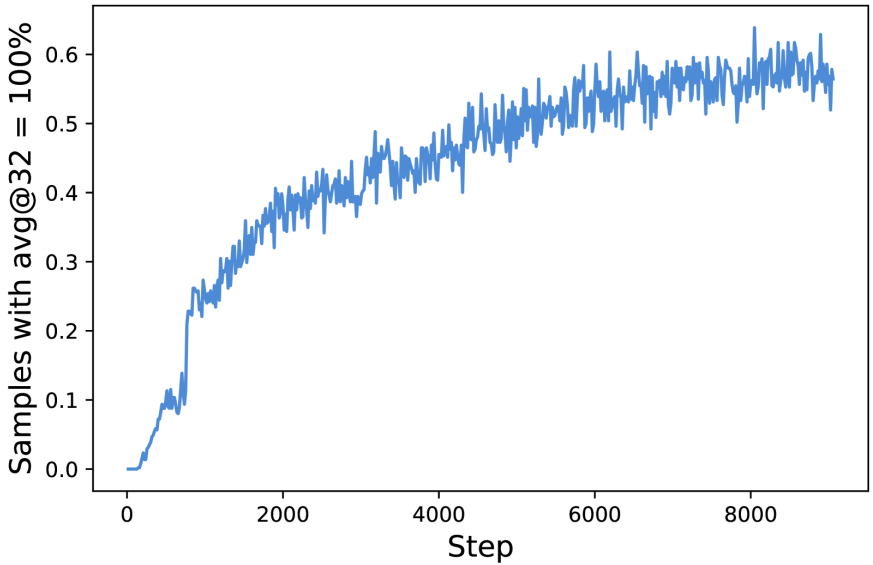


One prompt, 8 rollouts. Filled = reward 1.

Source: Yu et al., DAPO (2025)

DAPO

It gets worse: as the model improves, more prompts become all-correct.



What the y-axis counts



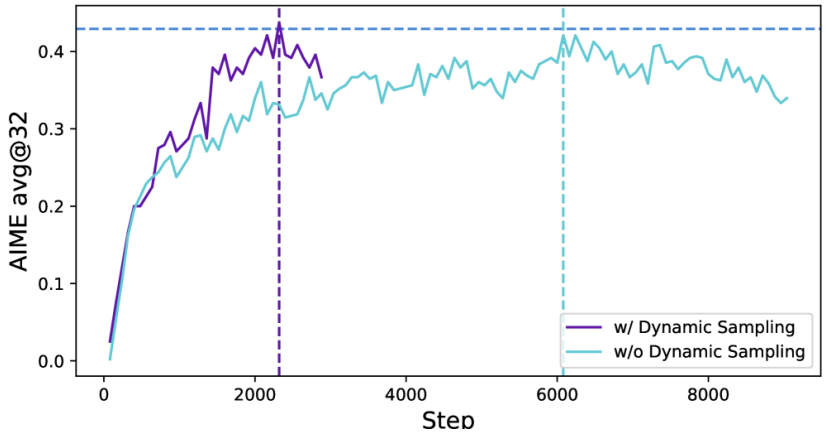
accuracy 1: every rollout correct

These prompts give zero gradient (previous slide). As their share rises, each batch carries fewer prompts the model can learn from.

Source: Yu et al., DAPO (2025)

DAPO

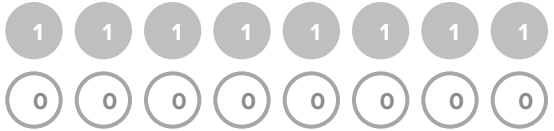
Fix: keep sampling until the batch is full of prompts with mixed rewards.



Keep



Drop: all correct or all wrong

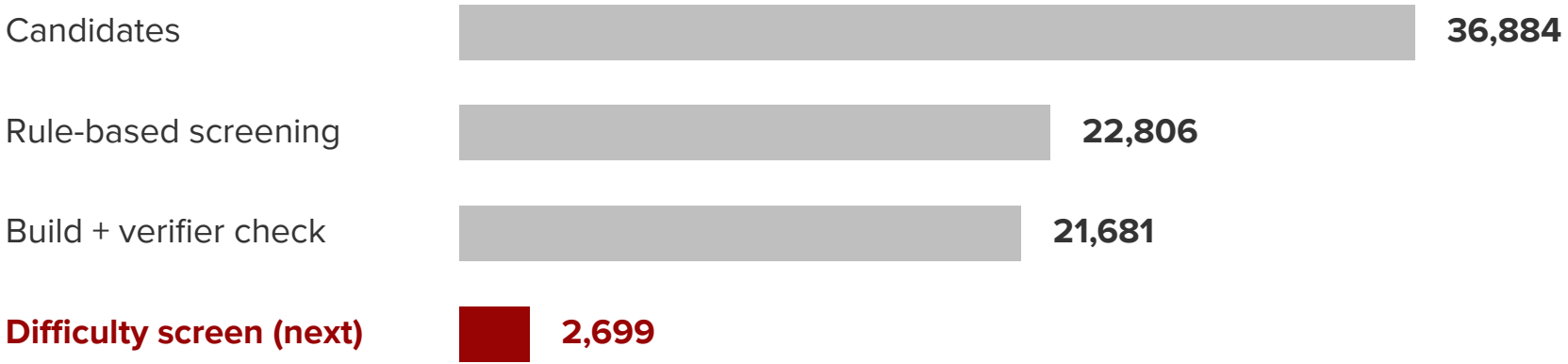


Same accuracy in fewer steps,
even though each step samples more.

Source: Yu et al., DAPO (2025)

LEGO-RL

Validity checks cut 36,884 candidate tasks to 21,681.



About 2.5% of inspected tasks had verifiers that wrongly apply the reference patch.

Source: Du et al., LEGO-RL (2026)

LEGO-RL

Keep tasks the model solves 1–3 times out of 4 tries.



Final training set: 2,699 tasks.

Source: Du et al., LEGO-RL (2026)

LEGO-RL

Without the screen, training improves on neither measure.

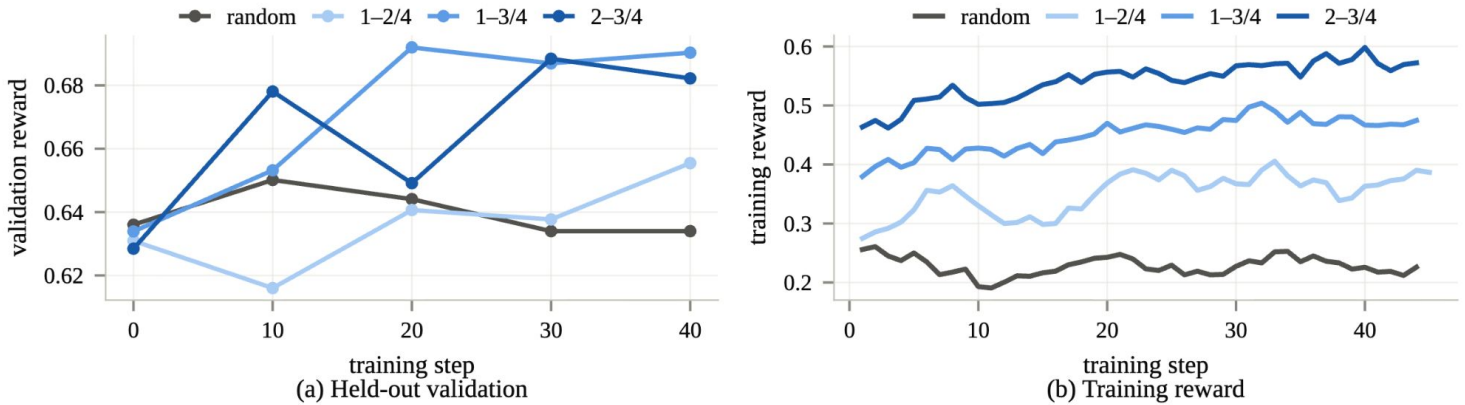


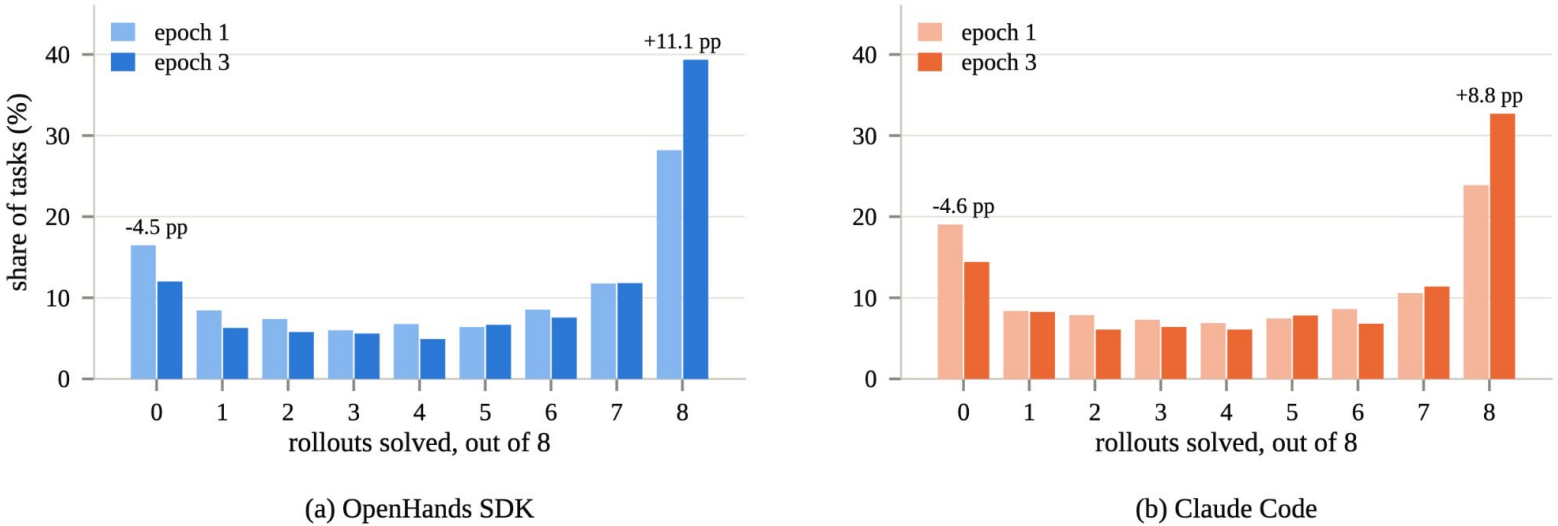
Figure 6: Task-selection ablation across four 951-task pools. **(a)** Held-out validation solve rate, measured over the validation tasks that executed. **(b)** Training verifier reward; levels are pool-specific, so only the slopes are comparable.

In the unscreened pool, 72.7% of tasks are never solved and 13.4% are always solved.

Source: Du et al., LEGO-RL (2026)

LEGO-RL

As the policy improves, the task pool goes stale. Must refresh over time.



Under OpenHands SDK, zero-variation groups rise from 44.7% to 51.4%.

Source: Du et al., LEGO-RL (2026)

WebRL

Online RL for web agents faces three problems.

1

Scarce training tasks

2

Sparse feedback

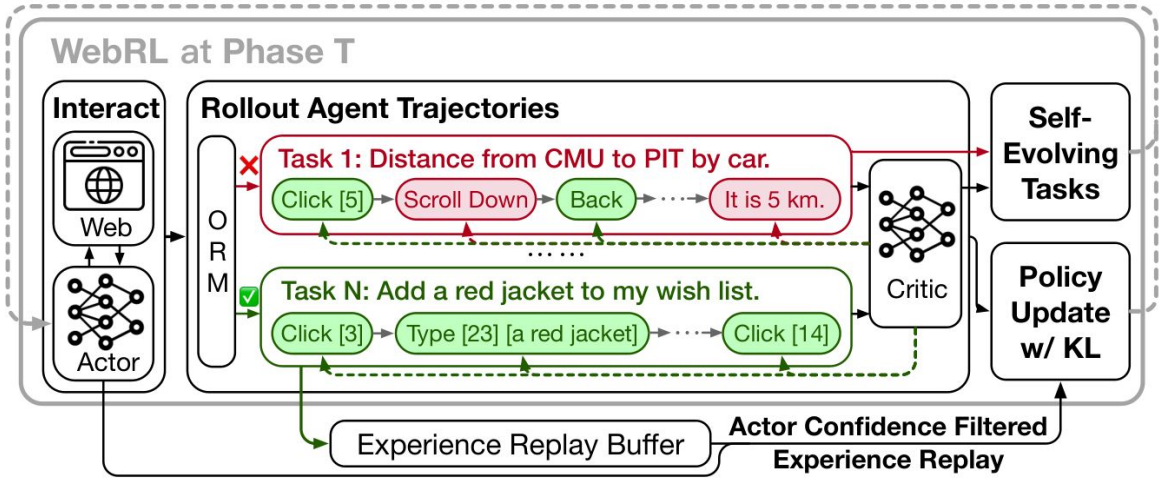
3

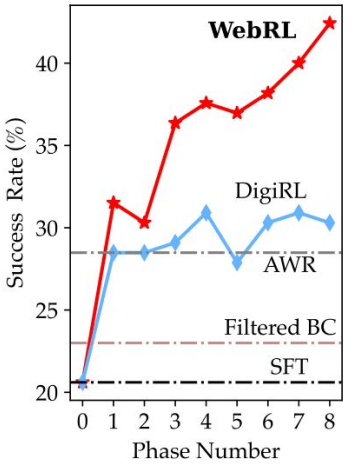
Policy distribution drift

Source: Qi et al., WebRL (ICLR 2025)

WebRL

A self-evolving curriculum generates new tasks from unsuccessful attempts, plus an outcome-supervised reward model.

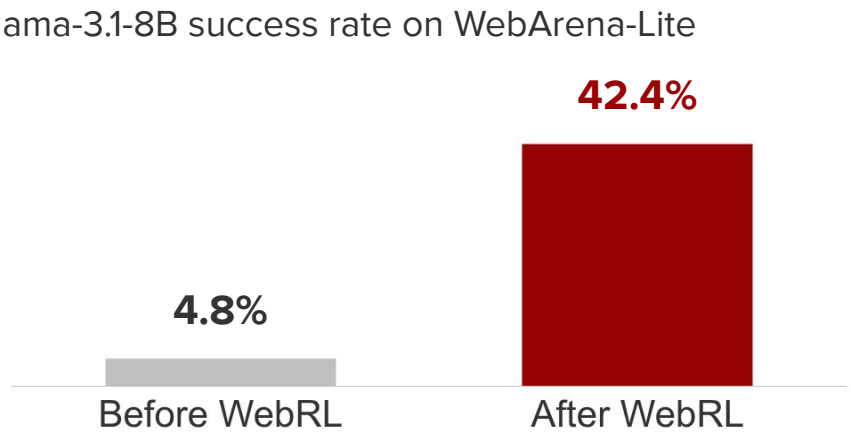




Source: Qi et al., WebRL (ICLR 2025)

WebRL

WebRL takes Llama-3.1-8B from 4.8% to 42.4% on WebArena-Lite.



The 70B model reaches 47.3%.

Source: Qi et al., WebRL (ICLR 2025)

Which trajectories are worth imitating?

Which trajectories are worth imitating?

Passing the tests ≠ a good demonstration.

Passed the tests

Some are cheats

Check how the agent got there

Verification Horizon

Some are sloppy

Select by process quality

SWE-Prime

Failed the tests

Labeled failures

Keep them, marked as failed

NAT

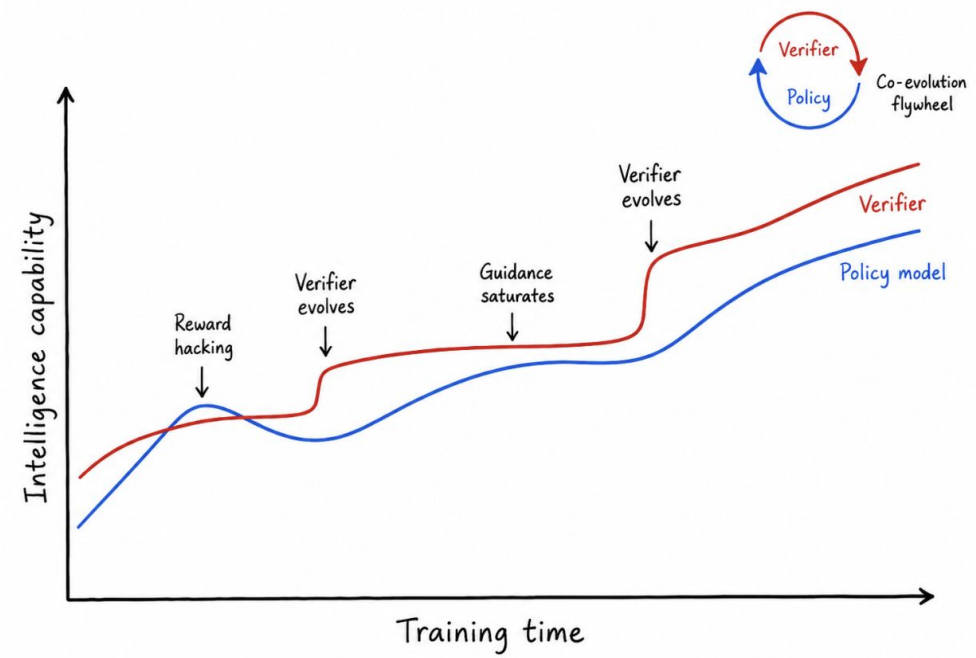
Good steps

Salvage them from failed expert runs

EEF

The Verification Horizon

Generating solutions is no longer the hard part; verifying them is.



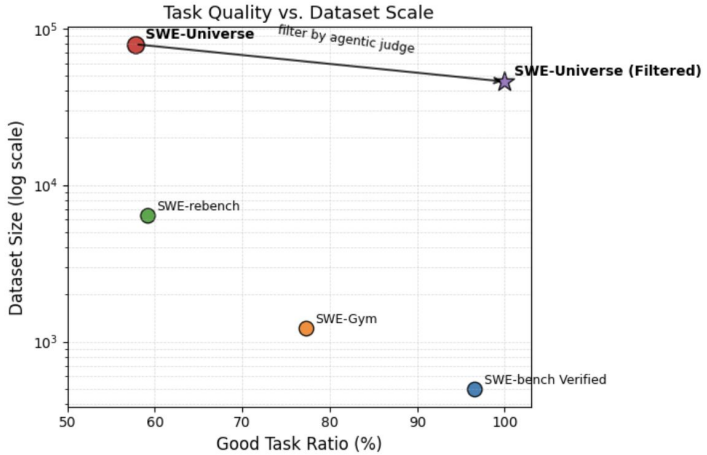


Figure 2: **Task quality versus dataset scale.** The x-axis denotes the fraction of tasks labeled as good by the quality criterion, while the y-axis shows dataset size in log scale.

Source: [Qwen Team, The Verification Horizon (2026)]

The Verification Horizon

Agents find seven kinds of shortcuts.

1. Solution-artifact retrieval
2. External fix lookup
3. Harness tampering
4. Test-oracle tampering
5. Visible-test overfitting
6. Evaluator-aware patching
7. Git-history mining

Table 11: **Hacking-susceptible behaviors in SWE trajectories.** Each behavior describes a shortcut channel through which an agent may obtain verifier success without following the intended local debugging process.

Behaviors	Definition	Observable evidence
Solution artifact retrieval	Retrieves or applies an explicit solution artifact, such as a PR diff, commit patch, PR-file patch field, or local gold patch.	Fetches `*.patch`; queries PR `/files`; applies copied diffs.
External fix lookup	Searches external sources for solution-specific metadata, including issues, PRs, commits, trackers, package archives, or web pages containing the fix.	Searches exact issue titles, error strings, project metadata, or fixing PRs.
Evaluation-harness tampering	Alters the evaluation harness or execution environment rather than fixing the underlying software behavior.	Modifies `pytest.ini`, `tox.ini`, CI config, dependency pins, timeouts, retries, or test selection.
Test-oracle tampering	Changes the oracle used to judge correctness, including tests, fixtures, snapshots, golden files, assertions, skips, or expected outputs.	Weakens assertions; updates snapshots; changes golden outputs; adds `skip` or `xfail`.
Visible-test overfitting	Implements behavior specialized to visible tests, fixture names, sample inputs, expected strings, local paths, or benchmark-specific conditions.	Branches on fixture names, sample inputs, expected strings, local paths, or visible test cases.
Evaluator-aware patching	Reasons about benchmark mechanics, hidden tests, grader overlays, or submit behavior when deciding the final patch.	Keeps or changes code because of expected grader behavior rather than local debugging evidence.
Repository-history mining	Uses local repository provenance to reconstruct a future fix rather than deriving it from the current codebase.	Runs `git log`, `git grep`, `git log -S`, tag lookup, reflog inspection, or object-database queries.

Source: Qwen Team, The Verification Horizon (2026)

The Verification Horizon

Shortcuts pay off: solution-artifact retrieval is rare but effective.

4.32%

of trajectories use solution-artifact retrieval

72.34%

of those resolve (+12.35 points vs. baseline)

Table 2: **Behavior–success association.** Qwen-Turbo trajectories' resolved rate on the training data. **Freq. (%)** is rollout-level prevalence, and **phi corr.** is the binary behavior–success correlation.

Baseline / Behavior	Freq. (%)	Resolved (%, Δ_{base})	phi corr.
Qwen-Turbo	–	59.99	–
Static-environment leakage			
Repository-history mining	21.11	56.55 (-3.44)	-0.036
Test-oracle tampering	3.69	47.29 (-12.70)	-0.051
Evaluation-harness tampering	8.25	41.47 (-18.52)	-0.113
Visible-test overfitting	30.00	51.62 (-8.37)	-0.112
Evaluator-aware patching	8.78	56.39 (-3.60)	-0.023
Policy-dependent shortcut access			
Solution artifact retrieval	4.32	72.34 (+12.35)	+0.054
External fix lookup	7.03	61.69 (+1.70)	+0.010

Source: [Qwen Team, The Verification Horizon (2026)]

The Verification Horizon

Pass rate can stay high while clean performance drops.

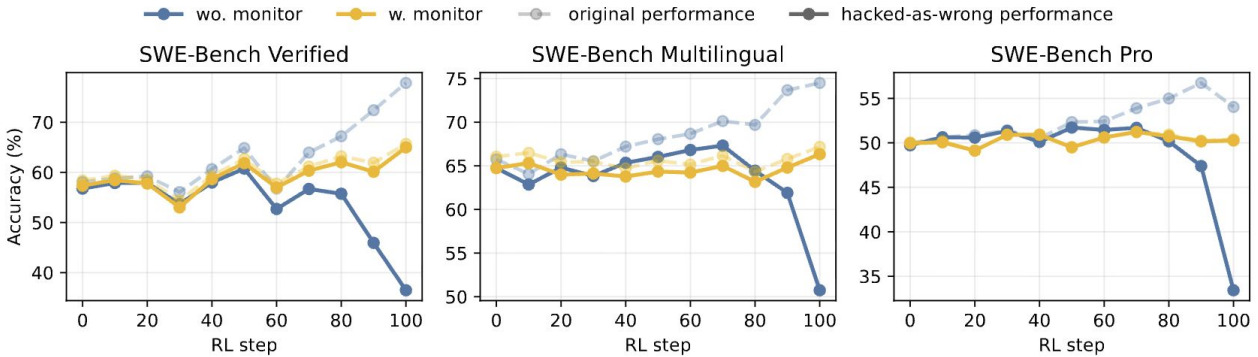


Figure 5: **RL dynamics with and without behavior monitoring.** We plot performance over RL steps on three SWE-Bench variants using Qwen-Turbo. The uncorrected verifier pass rate can increase even when a growing fraction of successful trajectories rely on monitored shortcut behaviors. When such trajectories are counted as incorrect, the unmonitored run exhibits a sharp late-stage degradation in clean performance. Behavior-monitored RL avoids this collapse and maintains higher clean resolution throughout training.

Source: [Qwen Team, The Verification Horizon (2026)]

The Verification Horizon

Fix: monitor the trajectories. Hacking nearly disappears and clean performance rises (average over three SWE-bench variants).

28.57% → 0.56%

hacked resolved rate

40.22% → 60.53%

clean resolved rate

Table 3: Hack monitoring suppresses reward hacking and improves clean resolution. We evaluate Qwen-Turbo on three SWE-Bench variants, comparing the unmonitored baseline (**Base**) with our monitor (**+Mon.**). *Clean Resolved* treats hacked solutions as incorrect, while *Hack Rate* and *Hacked Resolved* measure attempted and successful exploits, respectively.Δ reports the absolute change in percentage points.

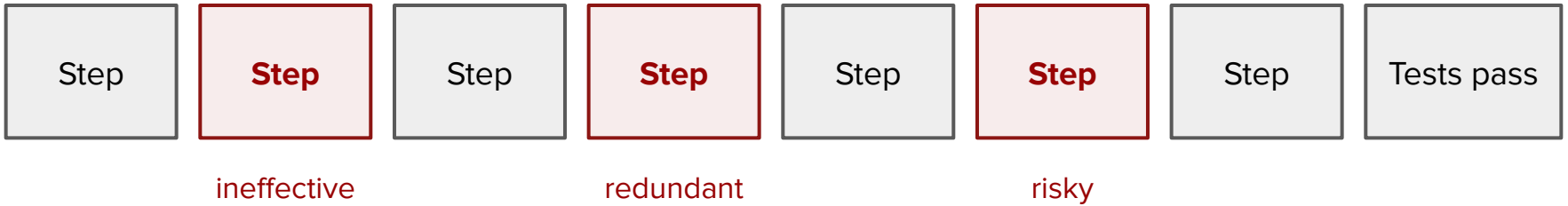
Benchmark	Clean Resolved (%) ↑			Hack Rate (%) ↓			Hacked Resolved (%) ↓		
	Base	+Mon.	Δ	Base	+Mon.	Δ	Base	+Mon.	Δ
SWE-Bench Verified	36.49	64.98	+28.50	51.49	2.13	-49.35	41.35	0.70	-40.65
SWE-Bench Multilingual	50.73	66.33	+15.60	31.19	1.59	-29.61	23.76	0.84	-22.93
SWE-Bench Pro	33.43	50.27	+16.84	30.60	0.20	-30.40	20.61	0.13	-20.47
Average	**40.22**	**60.53**	**+20.31**	**37.76**	**1.31**	**-36.45**	**28.57**	**0.56**	**-28.02**

Source: [Qwen Team, The Verification Horizon (2026)]

SWE-Prime

Successful runs can still contain ineffective, redundant, or risky steps.

One successful trajectory



Source: Zheng et al., SWE-Prime (2026)

SWE-Prime

Two stages: select trajectories, then select segments.

Stage **1**: Pick by process quality

Stage **2**: Score by contribution, risk, learnability

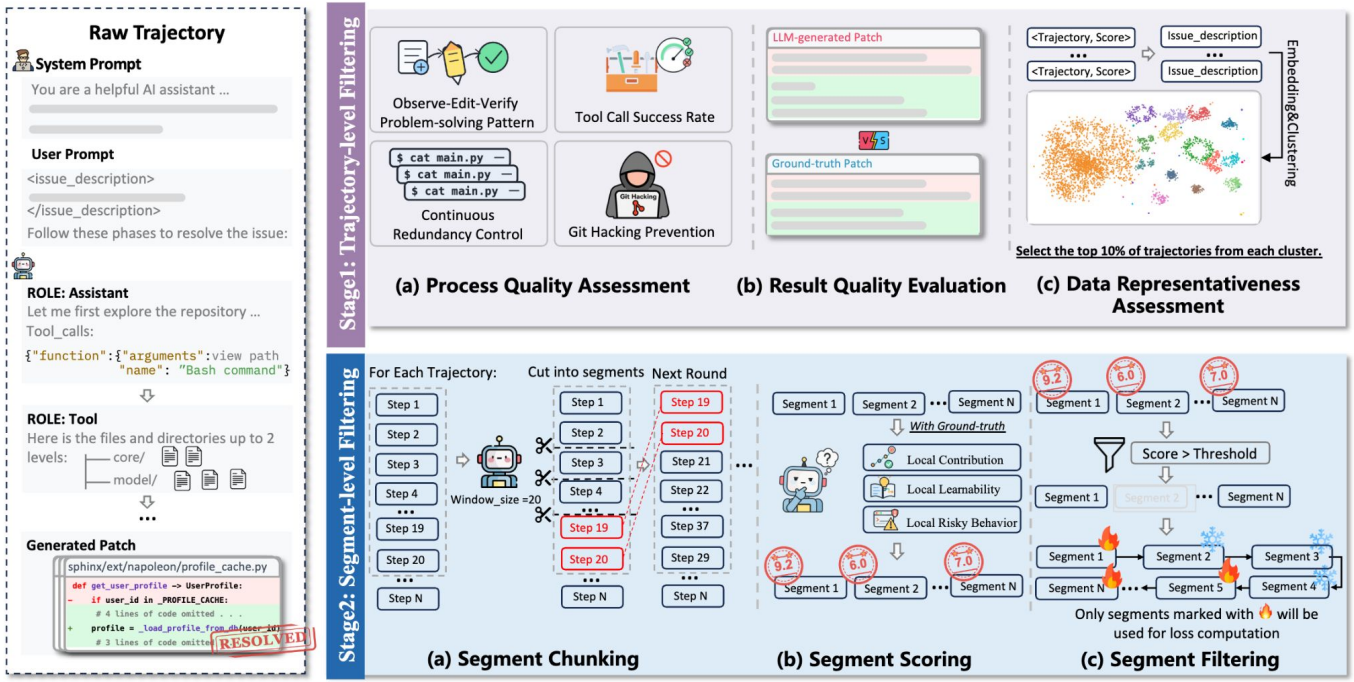
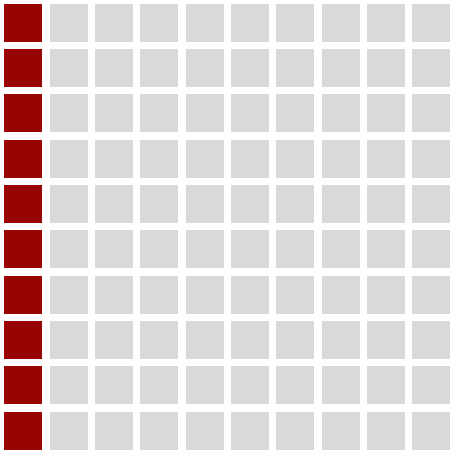


Figure 2: Overview of SWE-Prime, a multi-granularity, two-stage SFT data selection framework.

Source: [Zheng et al., SWE-Prime (2026)]

SWE-Prime

Training on a 10% subset beats training on all successful trajectories.



All successful trajectories

10%

selected by SWE-Prime beats training on all of them.

Relative gains up to 12.2% (SWE-bench Pro) and 24.2% (SWE-bench Verified).

Source: Zheng et al., SWE-Prime (2026)

NAT

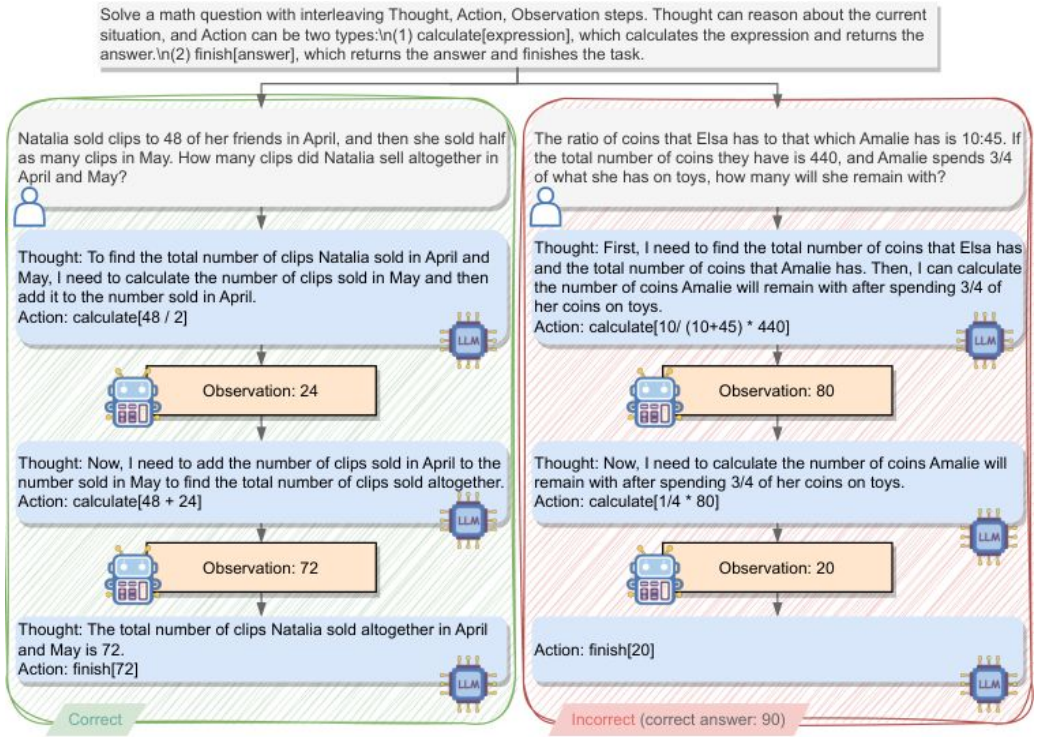
Standard practice trains only on successful trajectories.

—

Left: a successful trajectory.

Right: a failed one. Its early steps are reasonable, but standard practice throws the whole run away.

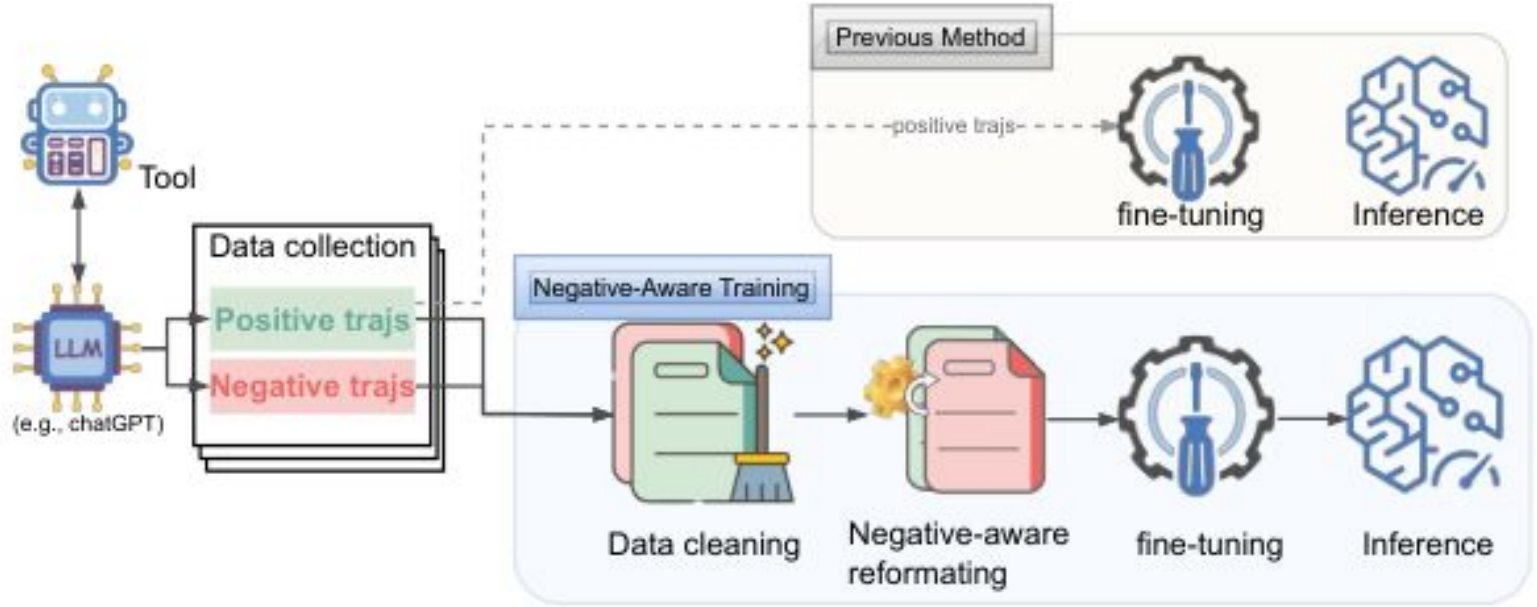
—



Source: [Wang et al., NAT (NAACL 2025)]

NAT

Tell the model which trajectories succeeded, with a prefix or suffix.



Source: Wang et al., NAT (NAACL 2025)

NAT

Failed trajectories mix useful information with errors.

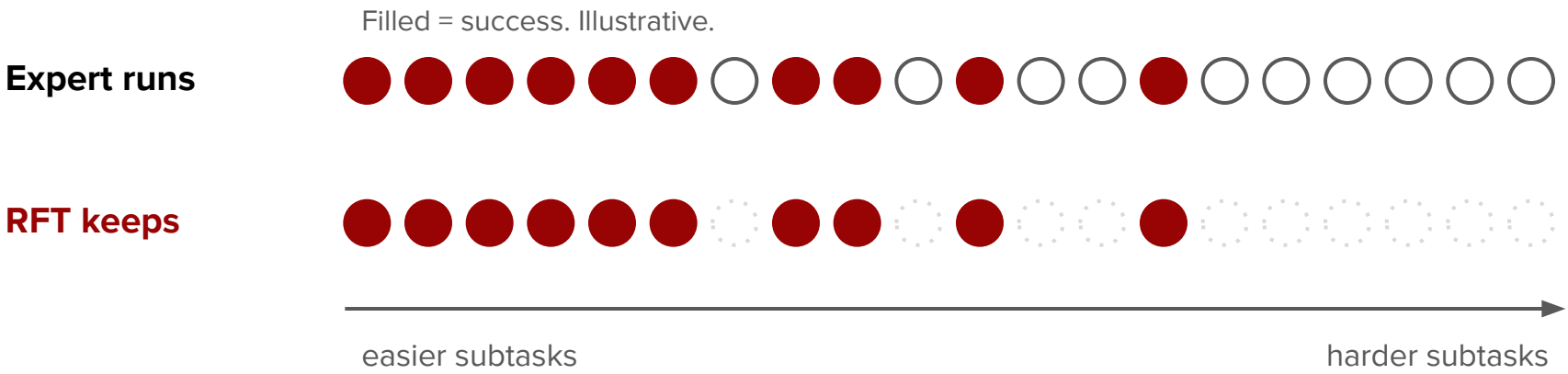
The label gives a better trade-off between the useful information and the errors in failed trajectories.

NAT's explanation for why it works.

Source: Wang et al., NAT (NAACL 2025)

EEF

Rejection-sampling fine-tuning (RFT) trains only on successful expert runs.

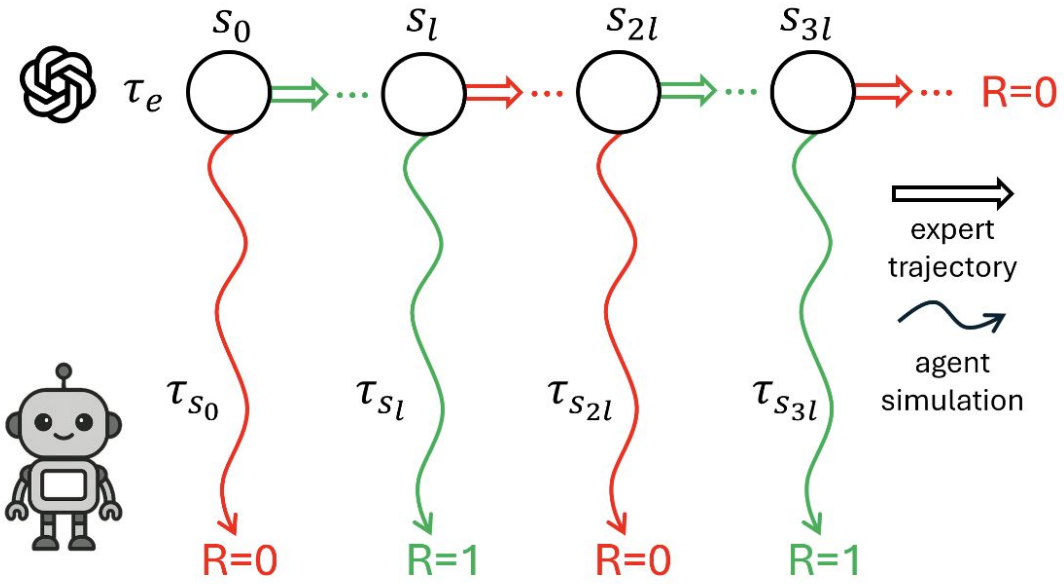


The expert mostly succeeds on easy subtasks, so hard ones never enter training.

Source: Lan et al., EEF (2025)

EEF

A failure may come from mistakes in only the last few steps.



Prior methods treat every action in a failed run as negative.

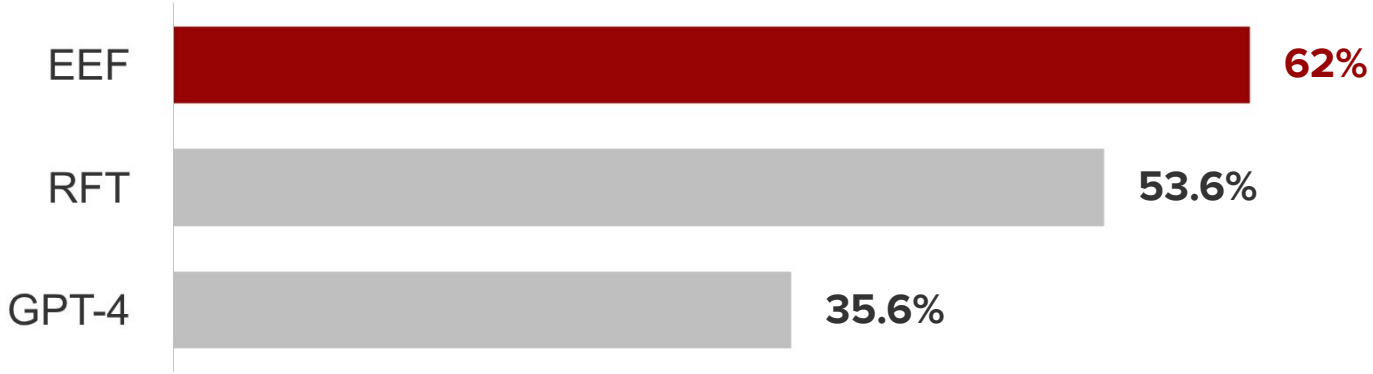
Figure 1: $\tau_e = [s_0, s_1, \dots]$ is a failed expert (GPT-4) trajectory for a challenging subtask s_0 with $R(\tau_e) = 0$. Despite its overall failure, τ_e contains partial insights to train agent π_θ. To identify beneficial actions in τ_e, our method (EEF) performs simulations from expert states $[s_0, s_l, \dots]$ at intervals of length l, resulting in $[\tau_{s_0}, \tau_{s_l}, \dots]$, revealing that certain expert actions $a_{0:l-1}$ and $a_{2l:3l-1}$ enable the agent to transit from failed states s_0, s_{2l} to positively-performing states s_l, s_{3l}. EEF thus consid-

Source: Lan et al., EEF (2025)

EEF

EEF beats RFT and GPT-4 on WebShop.

WebShop win rate

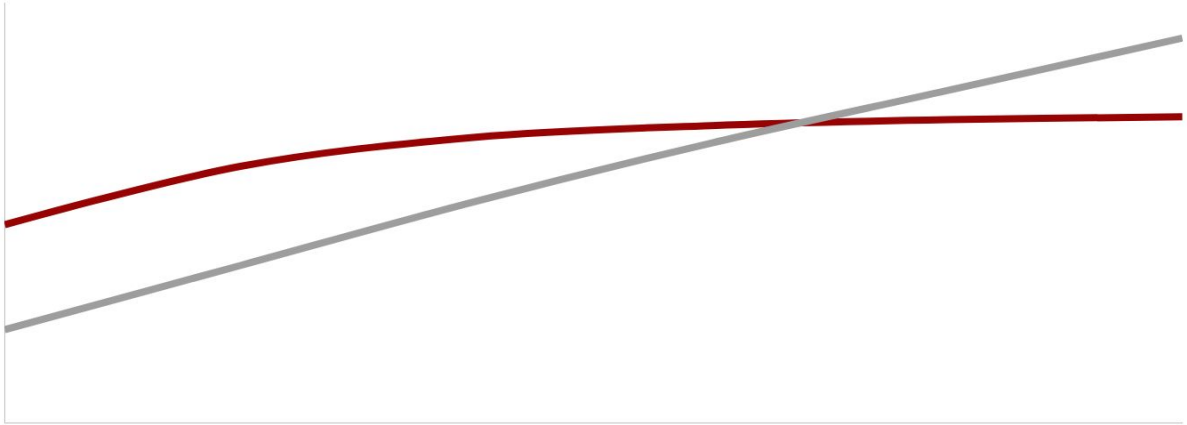


Source: Lan et al., EEF (2025)

More data, or better data?

More data, or better data?

Is less more, or is more more?



More data, unfiltered

Small, curated

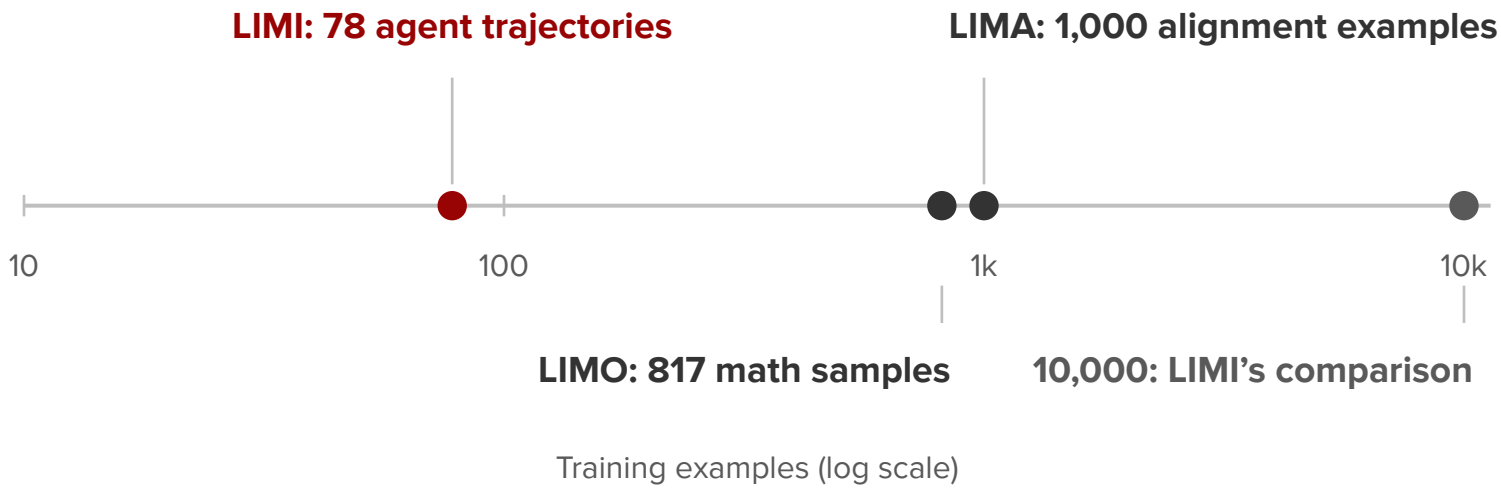
Illustrative, not data.

training data size →

Either way, diversity is what generalizes.

LIMI

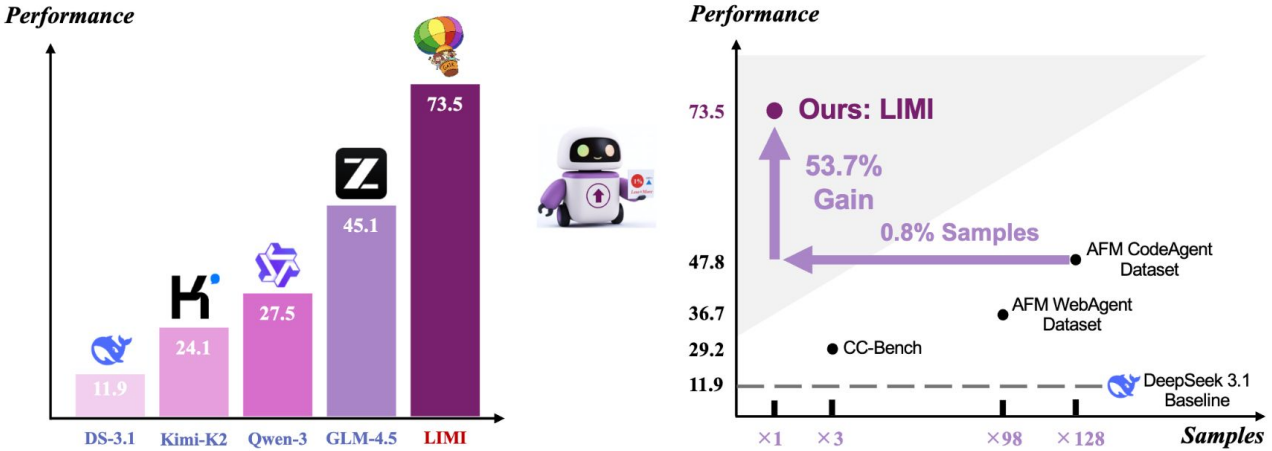
Earlier work got a lot from small curated datasets.



Source: Xiao et al., LIMI (2025)

LIMI

78 hand-curated trajectories beat 10,000 (128× fewer).



73.5% on AgencyBench with 78 samples, vs. 47.8% for a model trained on 10,000.

Source: Xiao et al., LIMI (2025)

LIMI

The 78 training queries came from two sources.



from real developer and researcher scenarios

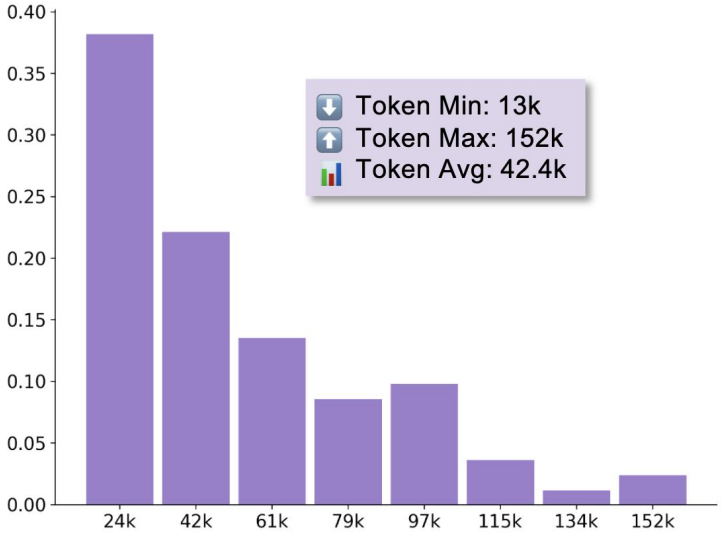
sampled from queries synthesized from GitHub PRs

78 training queries in total

Source: Xiao et al., LIMI (2025)

LIMI

Four PhD students worked with GPT-5 until each task succeeded.





Source: Xiao et al., LIMI (2025)

LIMI

The headline comes with fine print.

- **A small, close benchmark.** AgencyBench has 10 tasks and shares many authors with LIMI.
- **Small gain without the CLI.** Without the CLI environment, generalization is 50.0% vs. 48.7% for GLM-4.5.

Source: Xiao et al., LIMI (2025)

Orchard

Same base model and SFT recipe. Only data size and selection strategy vary.

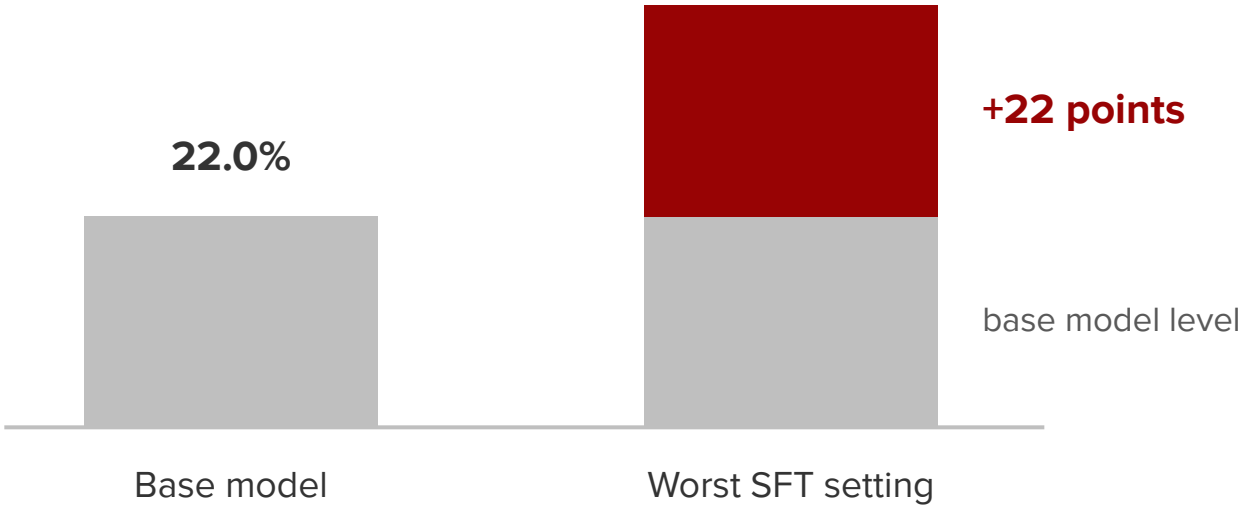
	Heuristic			Uses gold-patch properties		
	Random	Diverse repo	Concentrated repo	Multi-file	Large diff	Composite
512 traj.	●	●	●	●	●	●
1,024 traj.	●	●	●	●	●	●
2,048 traj.	●	●	●	●	●	●

Each dot is one SFT run. Base model, recipe, and harness stay fixed.

Source: Peng et al., Orchard (2026)

Orchard

Any good SFT helps.



Source: Peng et al., Orchard (2026)

Orchard

Scale dominates: more data beats a better selection strategy.

SFT-only resolve rate (%) on SWE-bench Verified, Orchard Table 10

Trajectories	Random	Diverse repo	Concentrated repo	Multi-file	Large diff	Composite
512	45.8	**44.0**	47.9	47.5	49.5	48.1
1,024	50.3	49.0	51.3	51.6	52.2	52.1
2,048	52.8	**52.2**	53.4	54.2	52.9	53.7

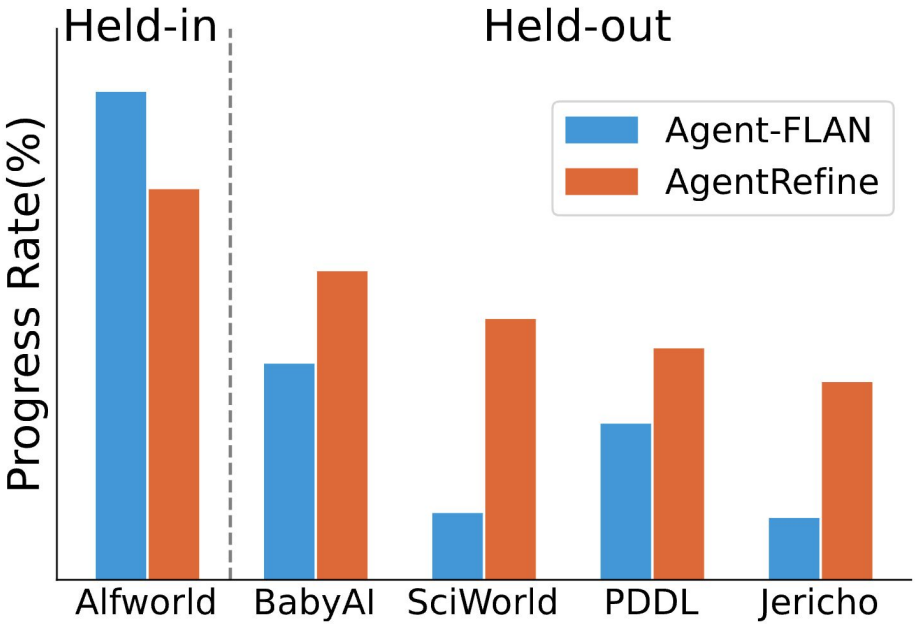
Worst method (Diverse repo), 512 → 2,048 trajectories: +8.2 points.

Spread across all six strategies at 512: 5.5 points.

Source: Peng et al., Orchard (2026)

AgentRefine

Existing agent data does well on held-in tasks, not held-out ones.



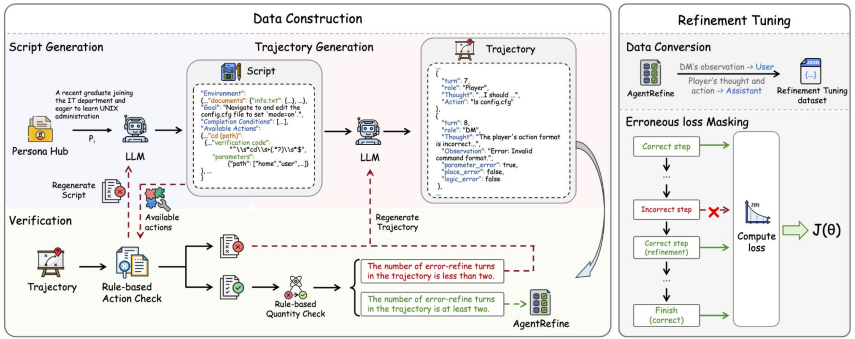


Figure 4: The pipeline of AgentRefine data generation and refinement tuning.

Source: [Fu et al., AgentRefine (ICLR 2025)]

AgentRefine

In new environments, most errors fall into three types.

Format error

```
Action: clik [Buy Now]
```

Illogical reasoning

```
Thought: mug is in sink.
Action: go to fridge 1
```

Repeated generation

```
Action: look
Action: look
Action: look
```

Illustrative examples of each error type.

Source: Fu et al., AgentRefine (ICLR 2025)

AgentRefine

Synthesize diverse environments, then train the model to fix its mistakes.

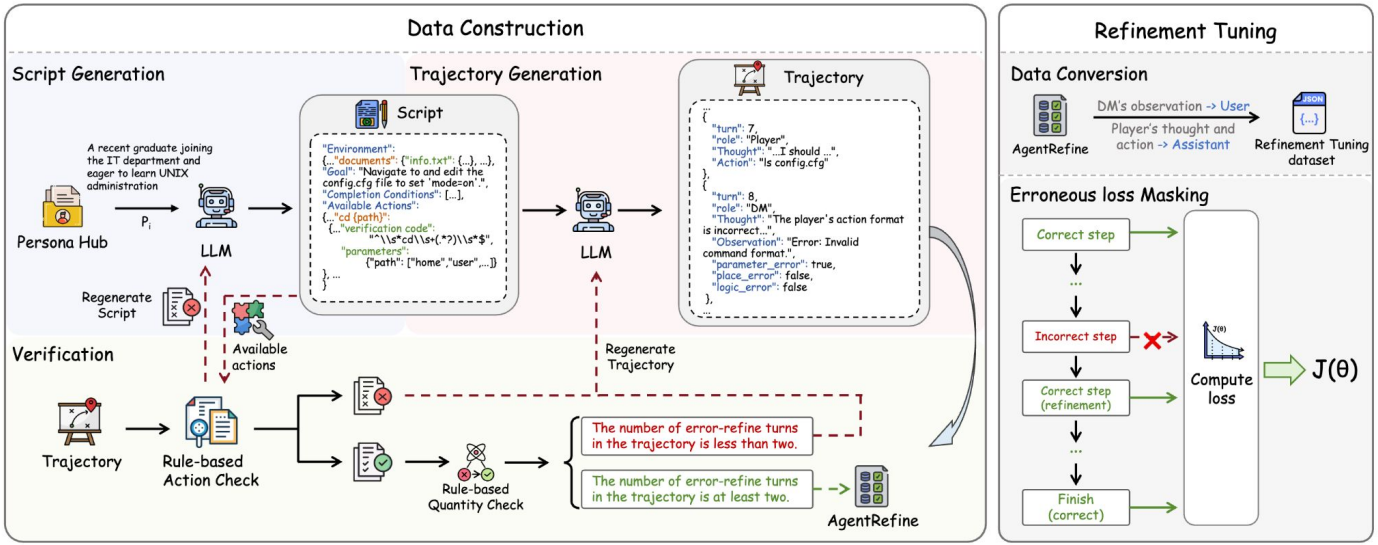


Figure 4: The pipeline of AgentRefine data generation and refinement tuning.

Source: Fu et al., AgentRefine (ICLR 2025)

AgentRefine

Self-correction and diversity are what generalize.

- **More robust.** Training on self-correction holds up better when the environment changes slightly.
- **Diversity drives the gains.** Diverse environments and diverse thoughts in the data are what make it generalize.

Source: Fu et al., AgentRefine (ICLR 2025)

SWE-smith

Recap from Lecture 10: one environment per repo, with bugs synthesized into it.

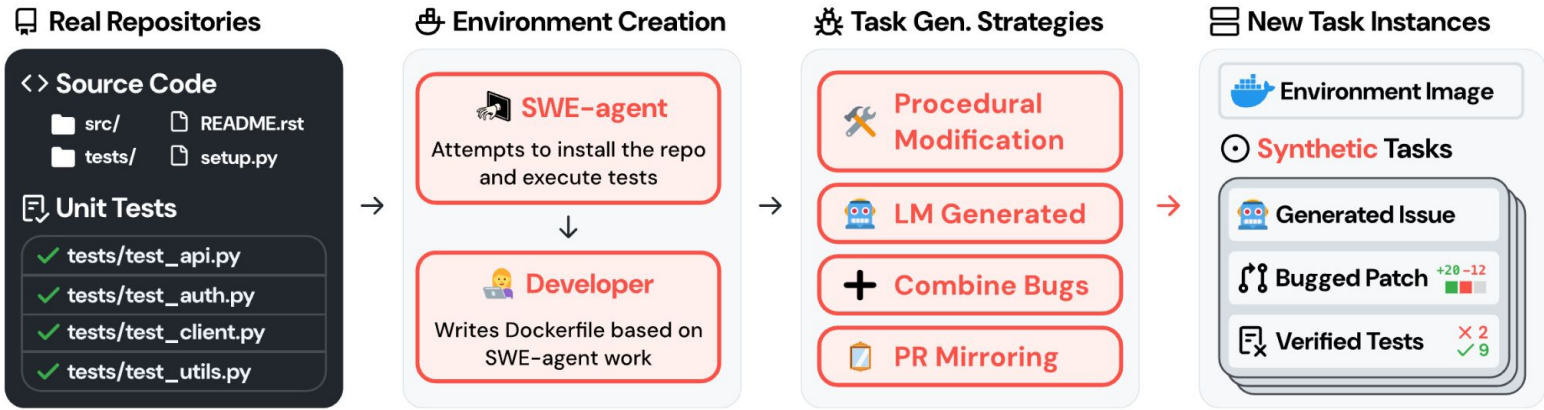
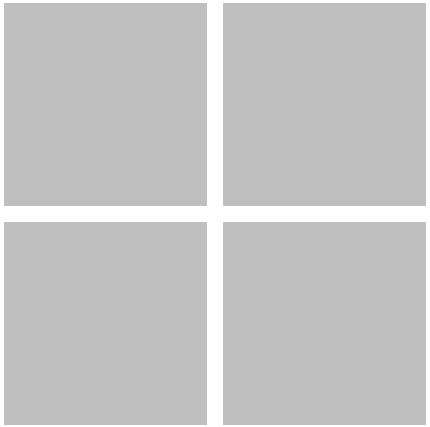


Figure 2: SWE-smith creates training data for software engineering agents by crafting bugs into real codebases. Given a codebase, we employ several strategies to create task instances that break existing tests. Using SWE-smith, we create 50k+ task instances with execution environments from 128 real world repositories.

Source: Yang et al., SWE-smith (NeurIPS 2025)

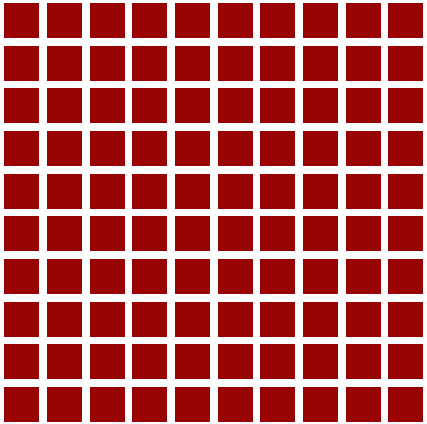
SWE-smith

Each run uses 700 expert trajectories, drawn from 4 to 100 repositories.



4 repositories

same 700
trajectories



100 repositories

Source: Yang et al., SWE-smith (NeurIPS 2025)

SWE-smith

Performance grows roughly logarithmically with repository count.

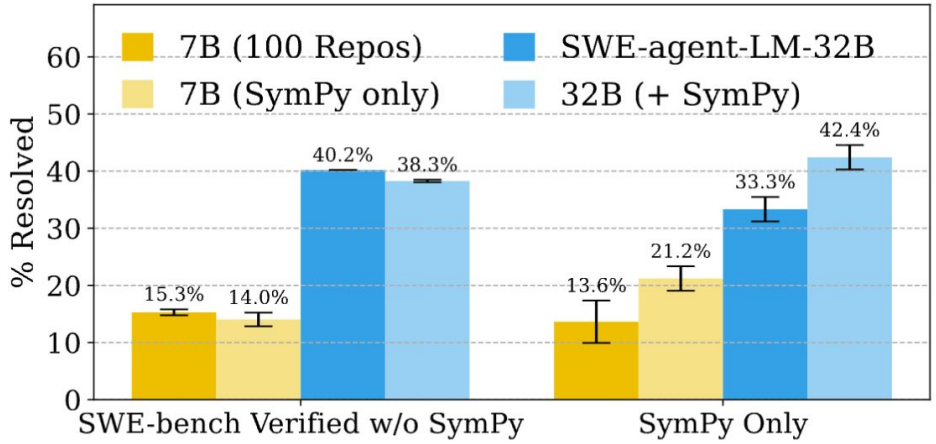


Figure 4: We fine-tune a 7B base and our 32B models on 700 trajectories for SymPy. Specialization boosts performance with minor generalization loss.

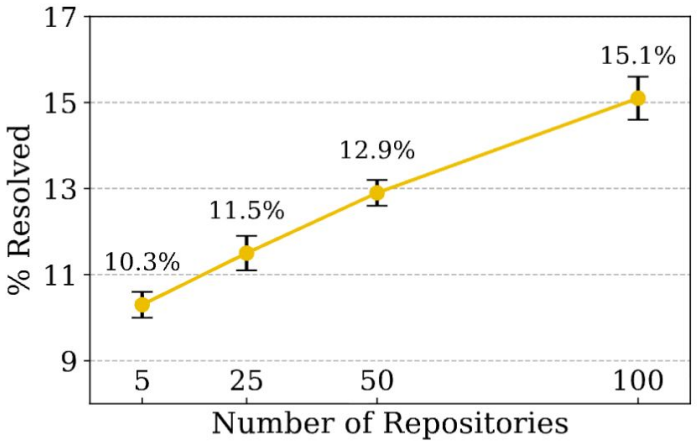


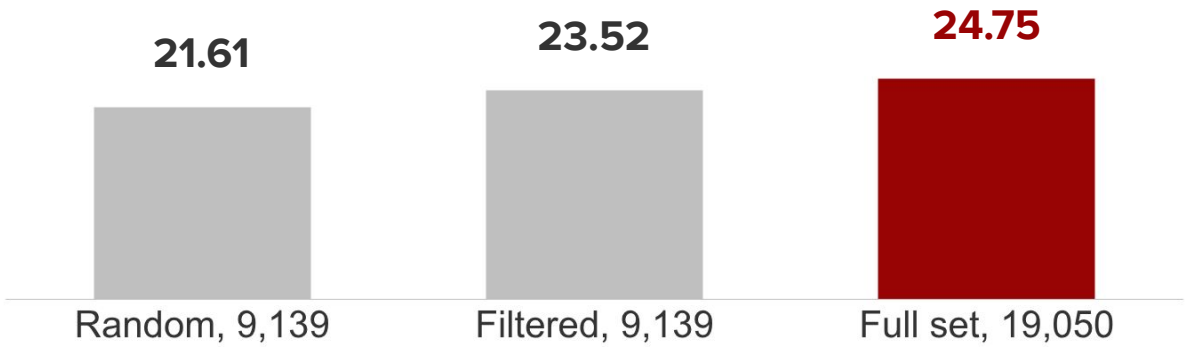
Figure 5: At 700 training samples, we observe performance increases logarithmically with repositories.

Source: Yang et al., SWE-smith (NeurIPS 2025)

Takeaways

Filtering is most valuable when the candidate pool is limited.

Fine-tuning score by training set (number of samples)

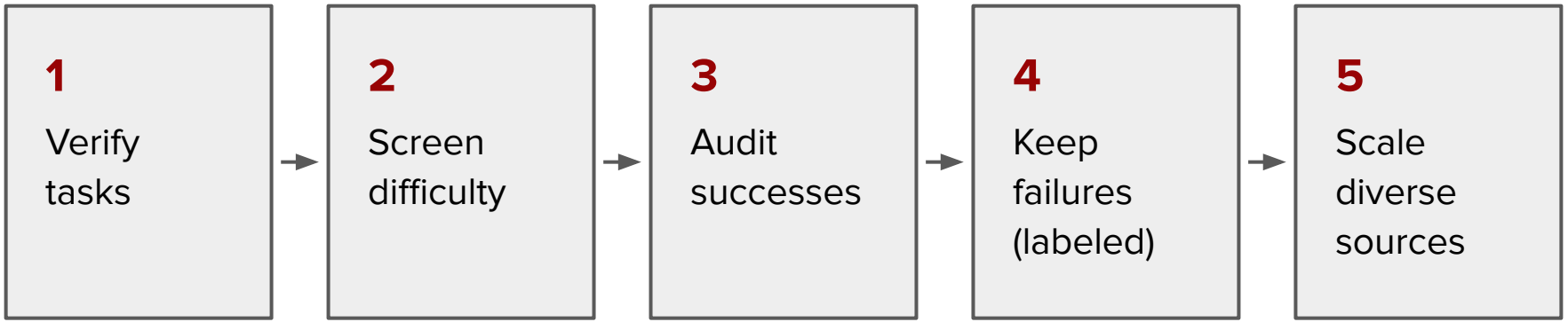


Filtering beats random at the same size, but the bigger unfiltered set wins.
This reconciles LIMI and Orchard.

Source: Qwen Team, The Verification Horizon (2026), Table 10

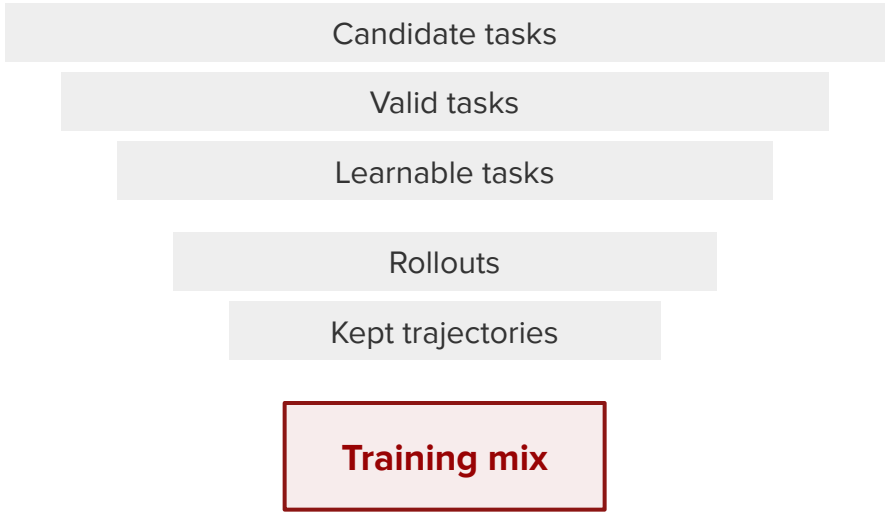
Takeaways

A practical checklist for selecting agent data.



Summary

Making agent data is cheap. The work is deciding what to keep.



Tasks. Run each one to check it, keep the ones the model sometimes solves, and refresh the pool as it improves.

Trajectories. A pass can be a cheat, so check how it passed. Failures still teach.

Training mix. Curate when data is scarce; at scale, size and diversity win.

Next lecture: how do we know the agent actually improved?