Agents for Coding and Software Development

From code completion to software engineering

Carnegie Mellon University
Language Technologies Institute

Graham Neubig
Language Technologies Institute

Coding agents

1

Writes code

single completion

2

Modifies repositories

modify multiple files

3

Performs development

full software lifecycle

Three ingredients

Prompt

Tools

LLM

Models that write code

Writing code from a prompt

Language

syntax · APIs · idioms

Editing

condition on both sides

Reasoning

specification → behavior

Training a coding model

Pre-training

large corpora including code

Mid-training

code mixtures · longer contexts

RL post-training

reasoning · execution rewards

Pre-training: learning from code and text

- **Mix sources:** web text, technical prose, mathematics, source code, and documentation.
- **Preserve structure:** indentation, file boundaries, APIs, and text–code relationships.
- **Evaluate during training:** held-out loss by domain + functional coding tasks.

$$\mathcal{L}_{\mathrm{LM}} = -\mathbb{E}_{x \sim D_{\mathrm{mix}}} \sum_t \log p_\theta(x_t \mid x_{<t})$$

Groeneveld et al., 2024 · OLMo, § 2 ↗

Pre-training: choosing and cleaning data

- **Filter by language and source:** remove generated dumps, broken encodings, and low-value repetition.
- **Deduplicate:** forks, vendored libraries, near-identical files, and copied solutions.
- **Control provenance:** record licenses, source dates, opt-outs, and sensitive-data filtering.
- **Prevent leakage:** split by repository or problem family; decontaminate evaluation tasks.

StarCoder · data selection, filtering, and decontamination ↗

Pre-training: splitting code into tokens

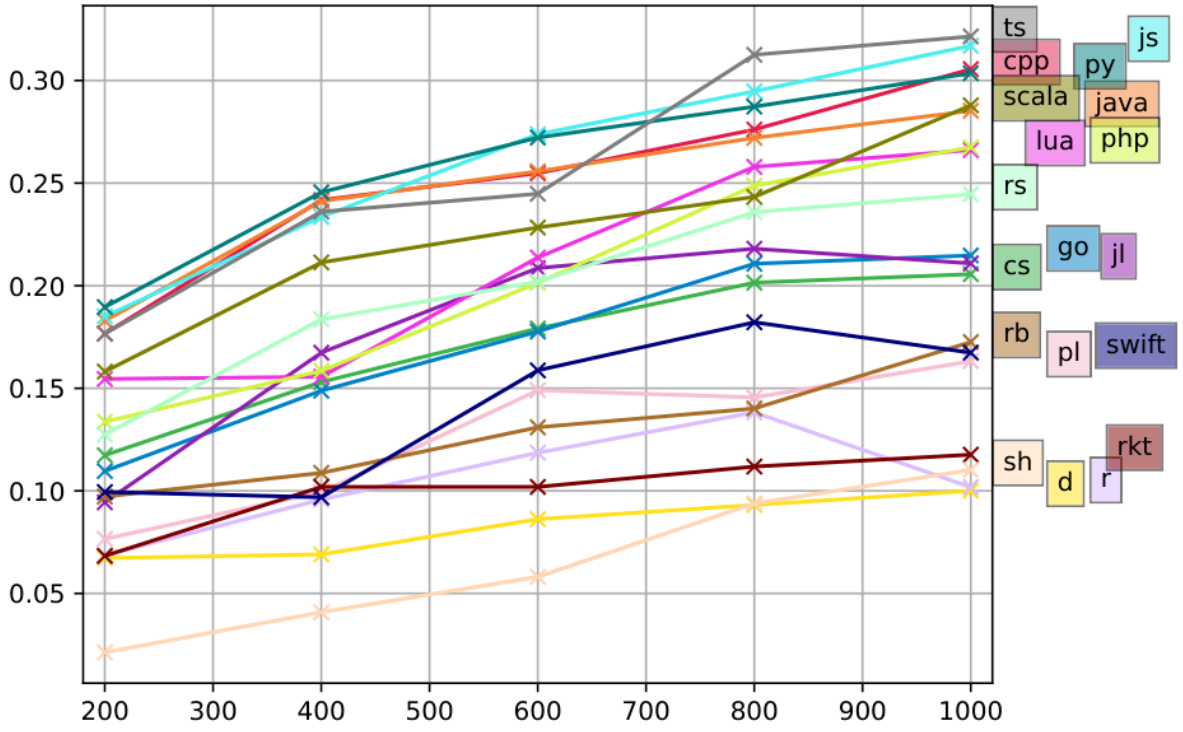
- **Learn code-aware subwords:** byte-level BPE on code + prose; identifiers can span tokens.
- **Preserve whitespace:** spaces, tabs, and newlines carry syntax and literal content.
- **Compress frequent runs:** merge whitespace into tokens instead of spending one token per space.

Source	`if·ready:↵····return·value↵`
Tokens	`if | ·ready | : | ↵··· | ·return | ·value | ↵`

Actual StarCoder2 split · "·" = space, "↵" = newline, "|" = token boundary

StarCoder § 5.3 ↗ Chen et al., 2021 · whitespace-run tokens ↗ StarCoder2 tokenizer ↗

Pre-training on a large code corpus



StarCoderBase

15.5B parameters

1T tokens · 80+ languages

Source, issues, commits, notebooks

Most languages improve with more tokens.

Horizontal: training tokens (billions) · Vertical: pass@1 · Each curve: one language

Li et al., 2023 · StarCoder ↗

Mid-training: adding more code

- **Continue language-model training:** increase code exposure while retaining text and math.
- **Choose the mixture empirically:** compare code gains with general-capability retention.
- **Code Llama:** Llama 2 → 500B additional tokens for 7B / 13B / 34B models.
- **Then extend context:** train on related files and longer sequences with suitable positions.

Rozière et al., 2023 · Code Llama, § 2 ↗

Mixing code, text, and math

Code / text / math	Common code	MATH	MMLU
100 / 0 / 0	**49.8**	10.3	23.8
70 / 20 / 10	48.3	**33.2**	**62.9**

Hui et al., 2024 · Qwen2.5-Coder, Table 3 (selected columns) ↗

Mid-training: handling longer inputs

types.py
Result.value can be None

→

worker.py
propagates Result

→

caller.py
complete the null check

- **Coherent sequences:** related files, paths, and cross-file dependencies.
- **Positions + length:** Qwen2.5-Coder trains at 8K → 32K; YaRN enables 128K.
- **Validate:** cross-file completion, infilling, and short-context retention.

Qwen2.5-Coder · § 3.2.2; see also Lecture 3 ↗

Infilling: using code before and after a gap

Fill the return-type annotation

```
def is_positive(x: int) -> ____:
    return x > 0
```

The body is to the right of the hole.

Use both sides to propose the span

```
def is_positive(x: int) -> bool:
    return x > 0
```

The suffix supplies evidence for `bool` .

Fried et al. · InCoder, § 2 and § 4 · illustrative example ↗

Infilling: marking and predicting gaps

Original	`prefix span suffix`
Training	`prefix [M0] suffix [M0] span [END]`
Inference	`prefix [M0] suffix [M0] → generate span`

- **One or more holes:** distinct sentinels associate each span with its location.
- **Zero-shot edits:** complete code, infer types, generate comments, rename variables.

Fried et al. · InCoder (2022 preprint; ICLR 2023), § 2 and § 4 ↗

Infilling: the value of right context

Completed functions passing tests (%)

Inference method	Suffix used for	Single-line	Multi-line
Left-to-right · 1 candidate	Not used	48.2%	24.9%
Left-to-right · 10 candidates	Reranking only	54.9%	28.2%
Causal-masked infilling	Generation	**69.0%**	**38.6%**

Fried et al. · InCoder, § 4.1 · HumanEval-derived infilling tasks ↗

Infilling: preserving code completion

Pass@1 · matched 52B-token training budget

Training objective	HumanEval	MBPP
Left-to-right language modeling	6.0%	8.9%
Causal masking	**8.0%**	**10.9%**

Fried et al. · InCoder, § 5 · objective ablation ↗ Related scaling study: Bavarian et al., 2022 ↗

Beyond infilling: more ways to learn from code

Commit diffs

before + message → after / diff

Muennighoff et al., 2023 · OctoPack ↗

Diagnostics

broken code + error → repair

Yasunaga & Liang, 2020 · DrRepair ↗

Tests

program + test outcomes → reward

Le et al., 2022 · CodeRL ↗

Execution traces

program → executed lines + states

Liu et al., 2023 · CodeExecutor ↗

Evaluating Generated Code

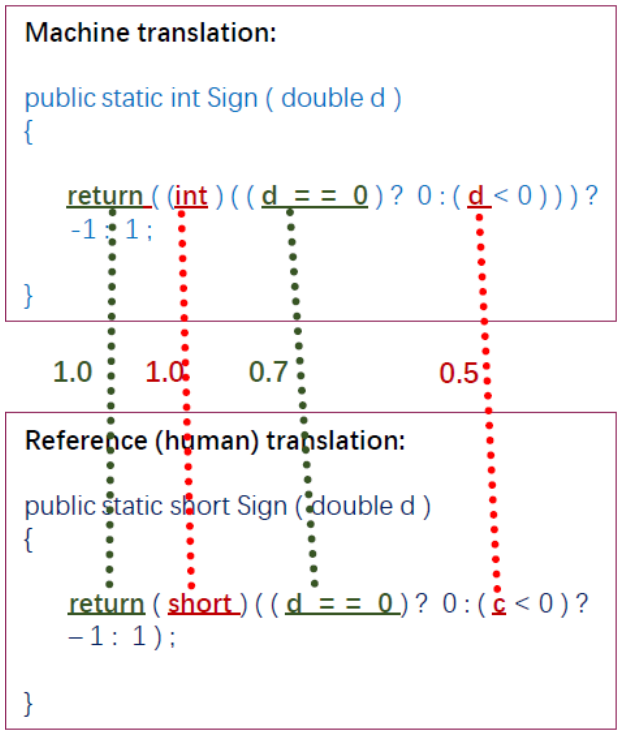
Comparing with a reference solution

- **Exact match:** does generated code reproduce the reference?
- **BLEU-4:** measure token n-gram overlap, with a brevity penalty.

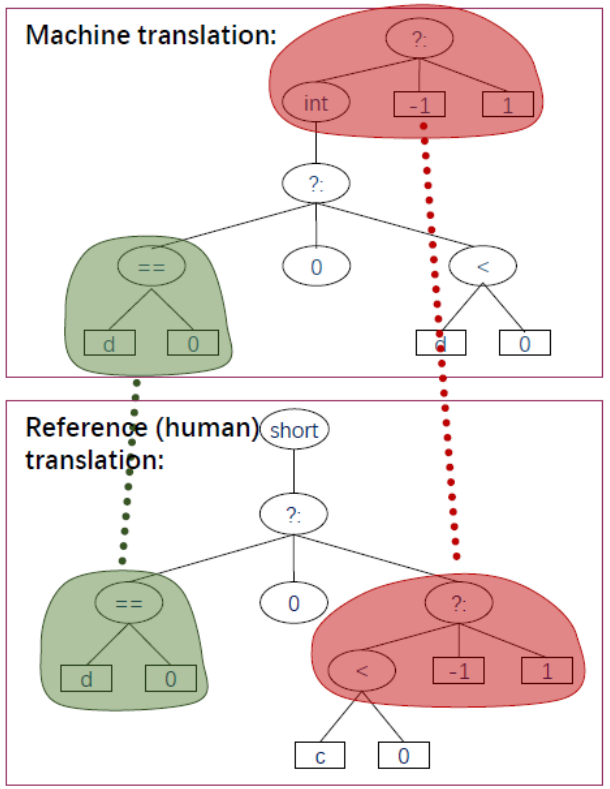
Integer x: is it positive?	Code	Behavior
Reference	`return x > 0`	Specification
Small textual change	`return x >= 0`	Wrong at x = 0
Different expression	`return not (x <= 0)`	Equivalent for integers

Yin & Neubig, ACL 2017 · § 5.1 ↗ Ren et al., 2020 · CodeBLEU ↗

Comparing code structure

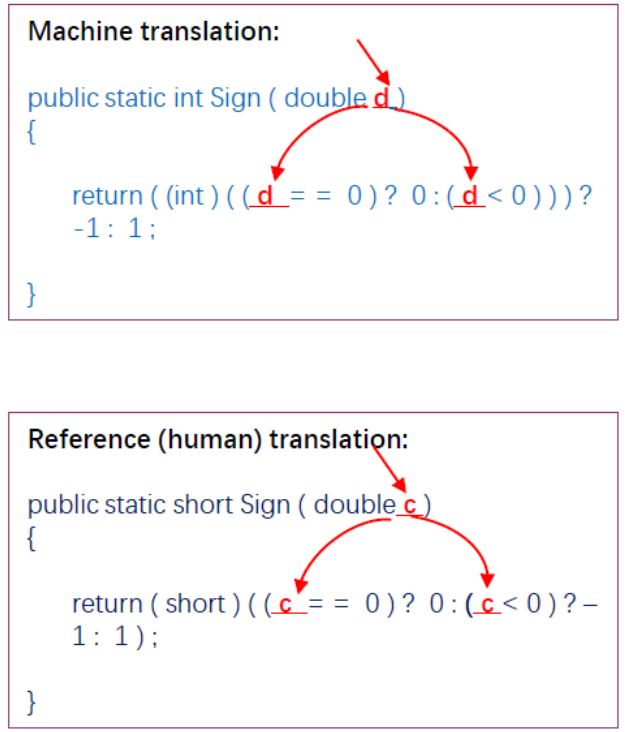


Weighted N-Gram Match



Syntactic AST Match

[('d', 7, 'comesFrom', [], []),
('d', 16, 'comesFrom', ['d'], [7]),
('d', 24, 'comesFrom', ['d'], [7])]

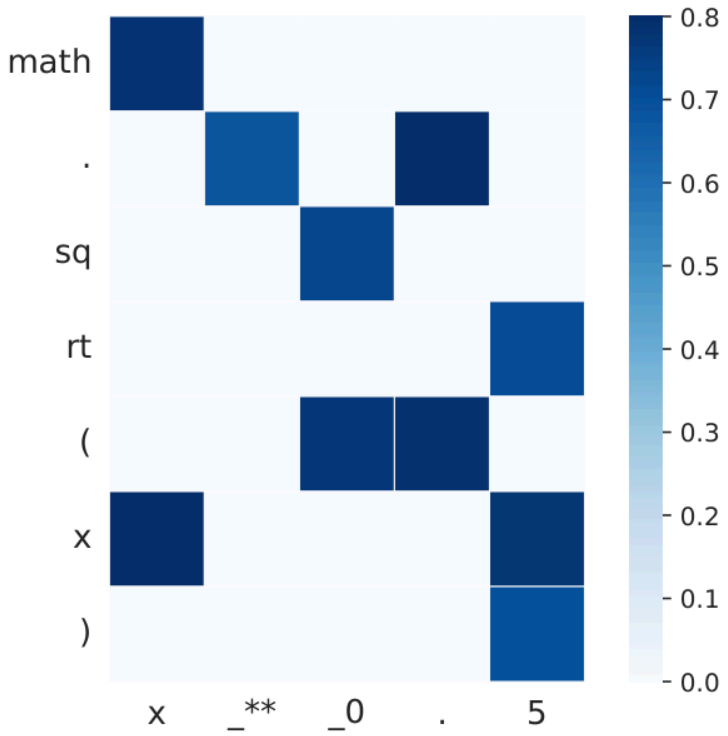


Semantic Data-flow Match

$$\text{CodeBLEU} = \alpha \cdot \text{N-Gram Match (BLEU)} + \beta \cdot \text{Weighted N-Gram Match} + \gamma \cdot \text{Syntactic AST Match} + \delta \cdot \text{Semantic Data-flow Match}$$

Ren et al., 2020 · CodeBLEU · original overview figure ↗

Comparing learned code representations

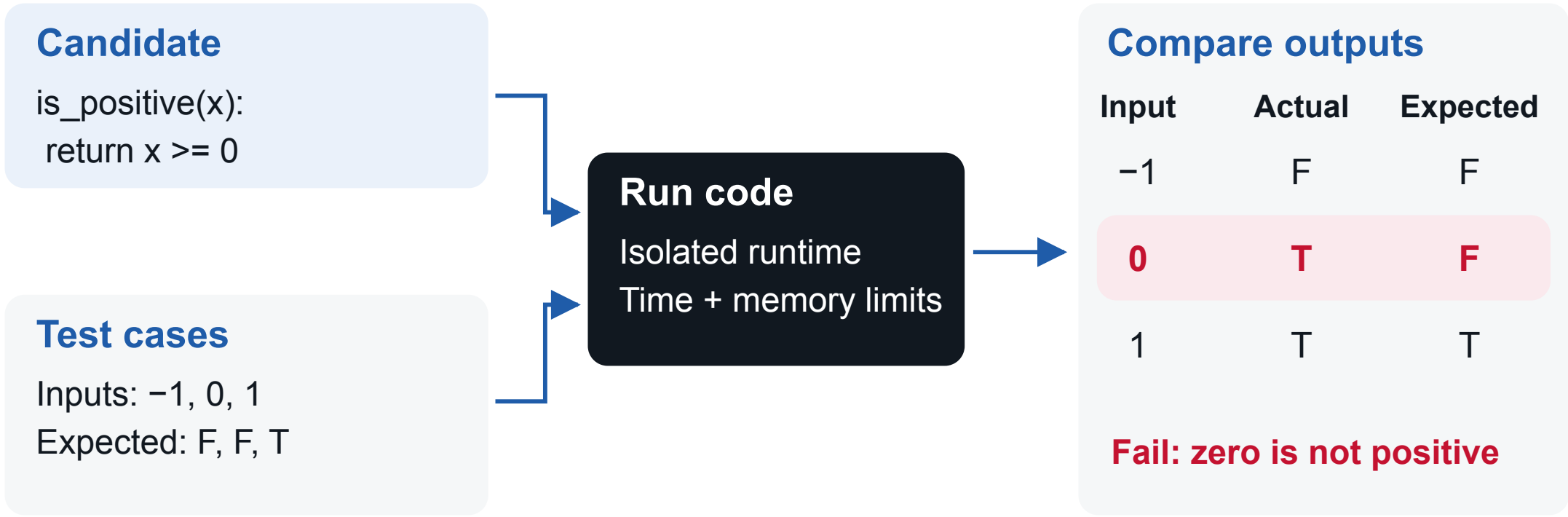


CodeBERTScore

- Encode prompt + each code snippet
- Compare contextual token vectors
- Best matches → precision / recall → F-score

Zhou et al., EMNLP 2023 · CodeBERTScore · original similarity heatmap ↗

Checking behavior by running code



Chen et al., 2021 · functional evaluation · classroom example ↗

Benefits and limits of execution checks

Benefit	Disadvantage / limit
Accepts different correct implementations	Finite tests can miss incorrect behavior
Detects semantic errors through execution	Needs trusted tests, dependencies, and isolated execution
Matches observed behavior	Requires time and a stable execution environment

Chen et al., 2021 · functional evaluation ↗ Liu et al., 2023 · EvalPlus / test adequacy ↗

Evaluating functions and full programs

HumanEval

164 Python functions

docstring → completion

hidden unit tests

CodeContests

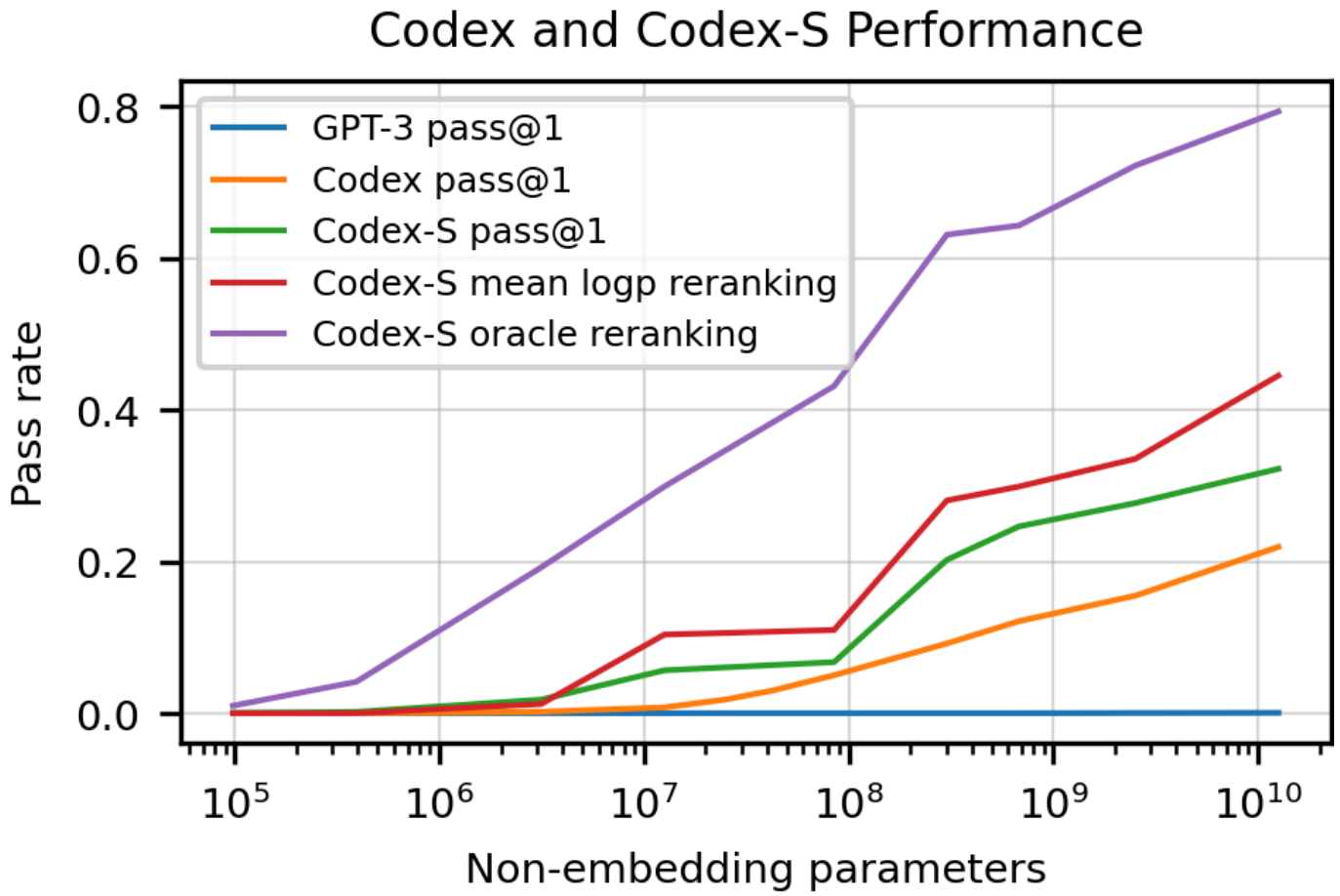
competitive problems · includes Codeforces

statement → full program

compile + test

HumanEval ↗ CodeContests / AlphaCode ↗

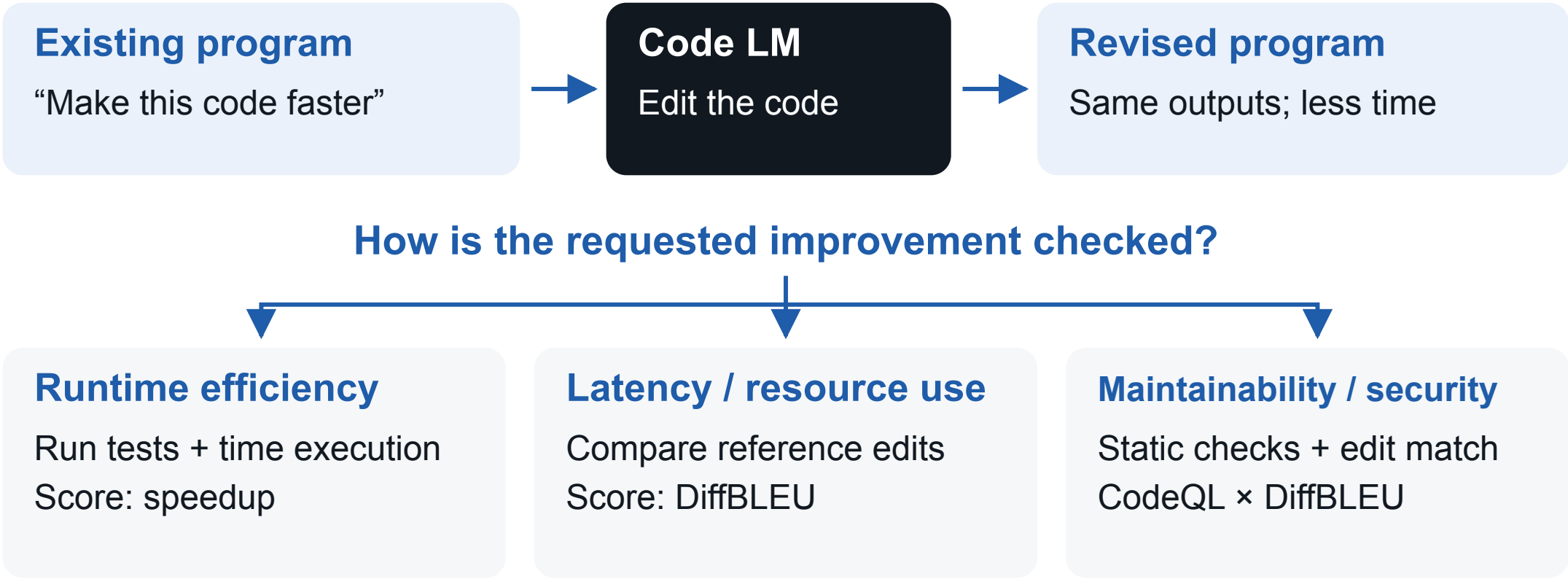
Success across multiple attempts



pass@k: probability that at least one of k sampled programs passes all tests

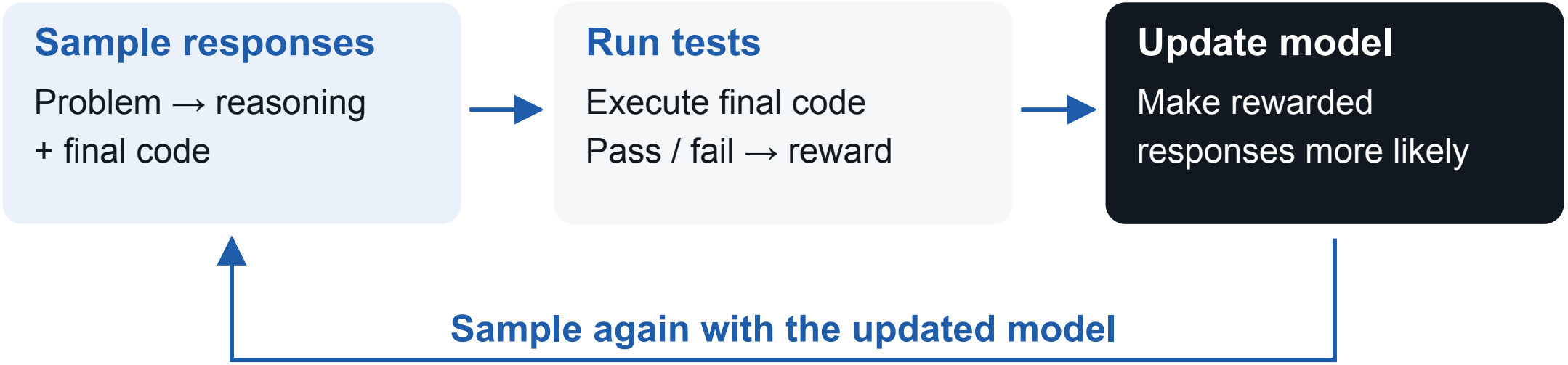
Chen et al., 2021 · HumanEval ↗

Non-functional requirements



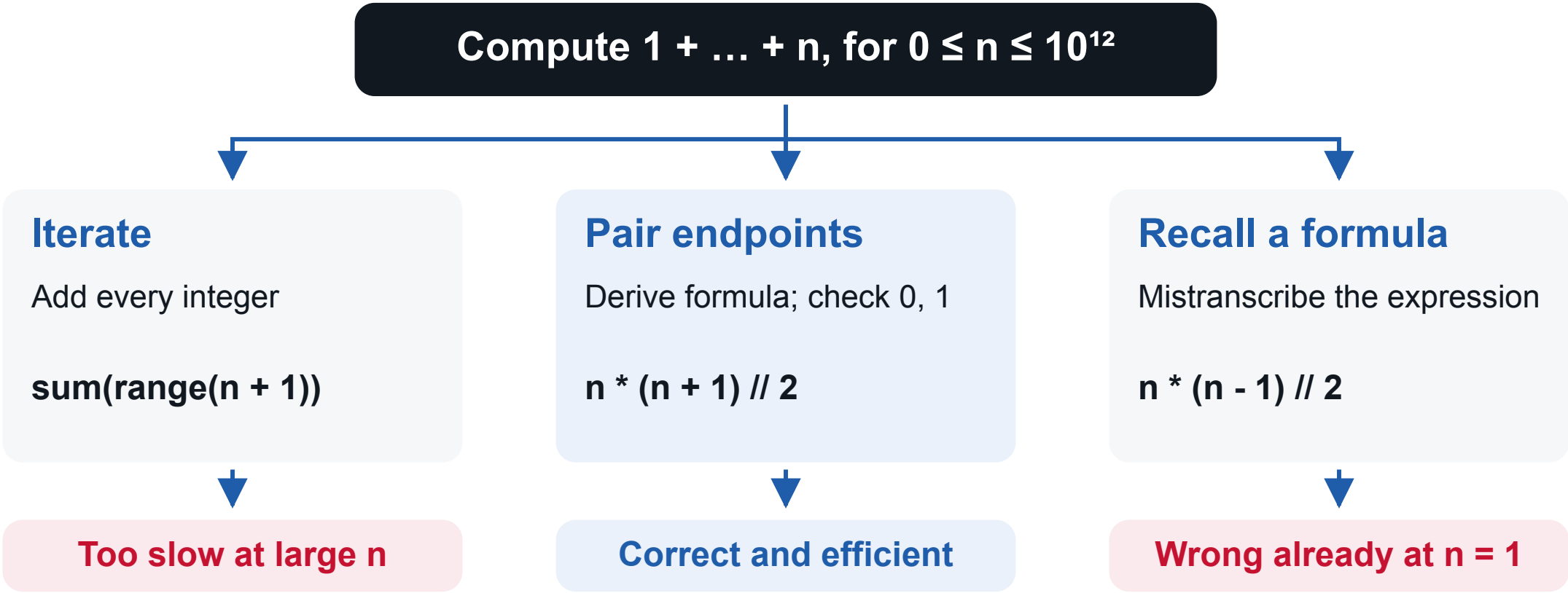
Singhal et al., 2024 · NoFunEval · 397 code-editing tasks ↗

Learning from test results

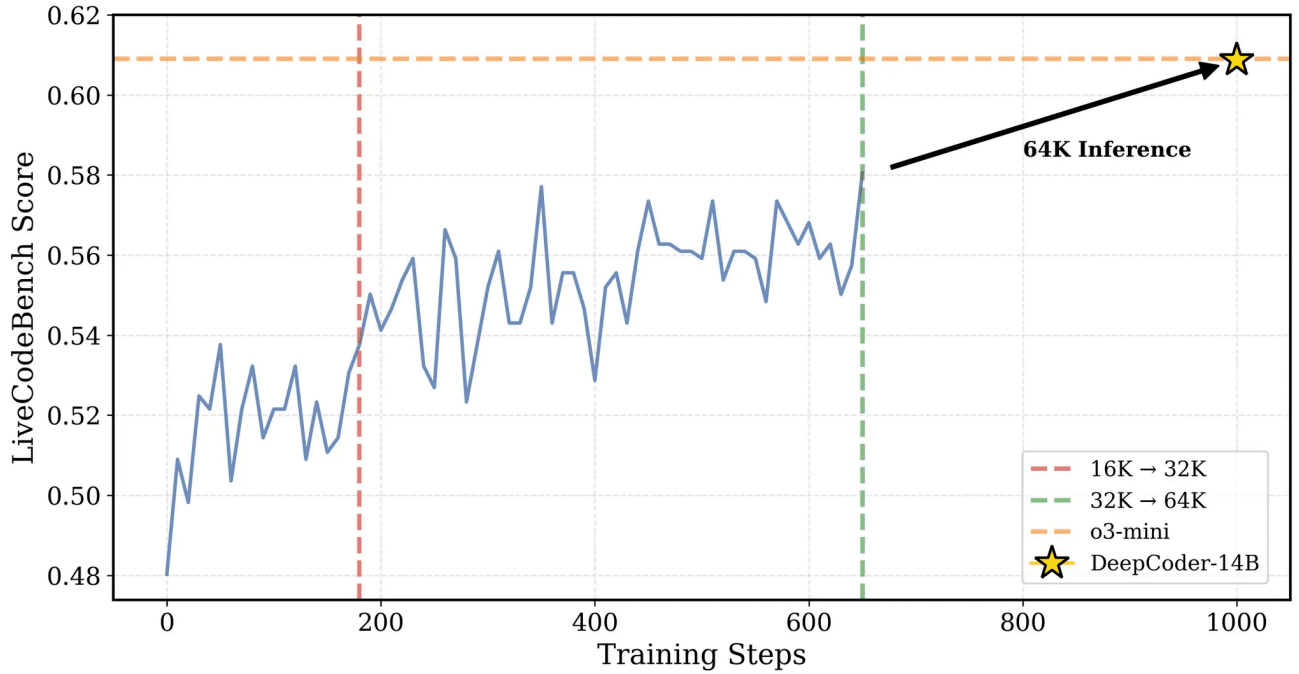


Luo et al., 2025 · DeepCoder · execution-reward RL ↗

Reasoning toward a solution



Coding accuracy after reinforcement learning



16K → 32K: response-token limit during RL (K = 1,000 tokens).
★ / **64K:** more tokens at evaluation, no further RL. Orange dashed line: o3-mini.

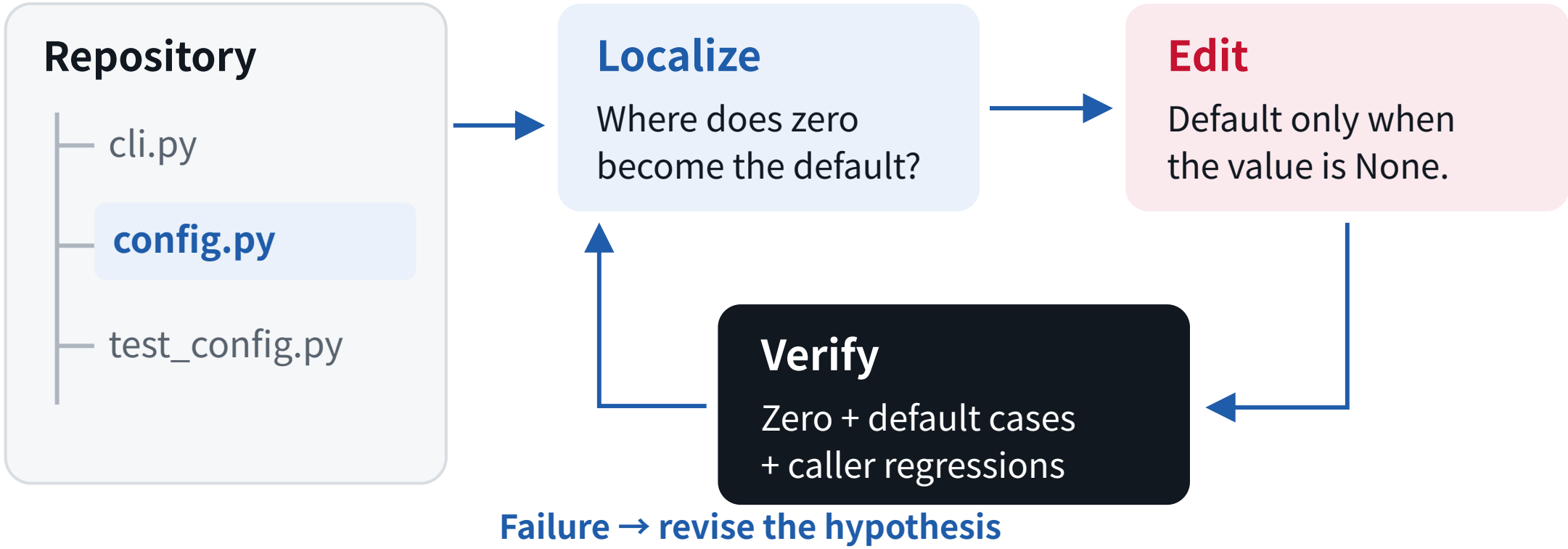
Luo et al., 2025 · DeepCoder · LiveCodeBench coding tasks ↗

Agentic coding

From single-step to agentic coding

Single-step coding	Self-contained prompt	→	Reasoning + code	→	Evaluate the answer
Agentic coding	Issue + existing files	→	Search → edit → run tests ↺	→	Patch + evidence

The localize–edit–verify loop



Interaction-loop concepts: SWE-agent · original teaching diagram ↗

Localize: find the cause

Localize the implementation

```
$ rg 'retries' buggy_config.py
return config.get("retries") or 3
```

Zero is falsy: inspect the defaulting rule.

Reproduce the behavior

```
$ RETRY_MODULE=buggy_config python3 \
  -m unittest test_config.RetryTests.test_zero
AssertionError: 3 != 0
FAILED (failures=1)
```

Now repair the condition, not the test.

Edit: change the defaulting rule

Before

```
return config.get("retries") or 3
```

All falsy values select the default.

After

```
value = config.get("retries")
return 3 if value is None else value
```

An explicit zero now survives.

Verify: check the fix and regressions

Run the existing tests

```
$ python3 -m unittest -v test_config
```

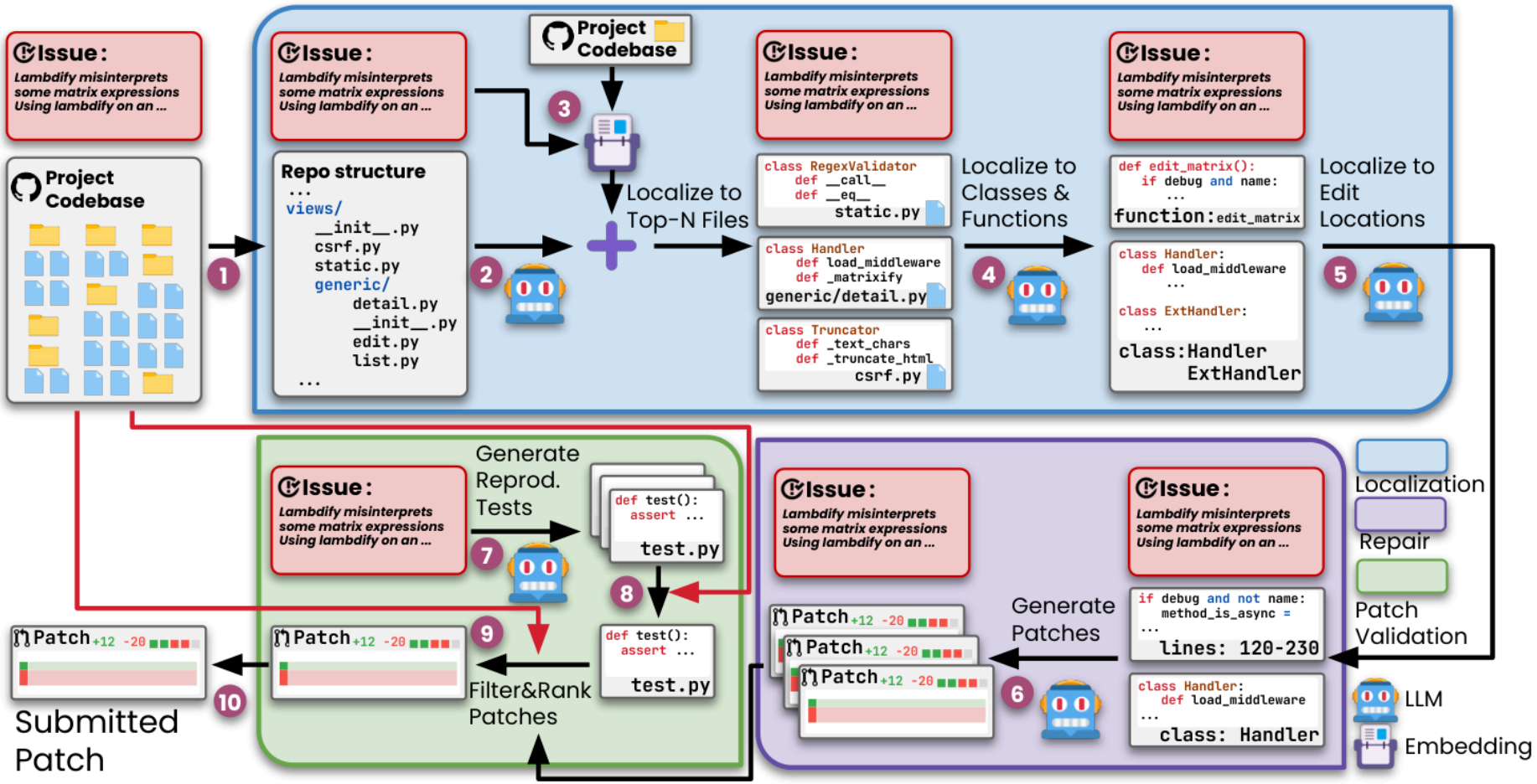
```
missing   → 3   PASS
None      → 3   PASS
zero      → 0   PASS
positive  → 2   PASS

Ran 4 tests: OK
```

Use failures to guide the next step

> **Zero still becomes three?**
>
> Localize where the value changes.

> **A default case breaks?**
>
> Edit the rule, then verify again.

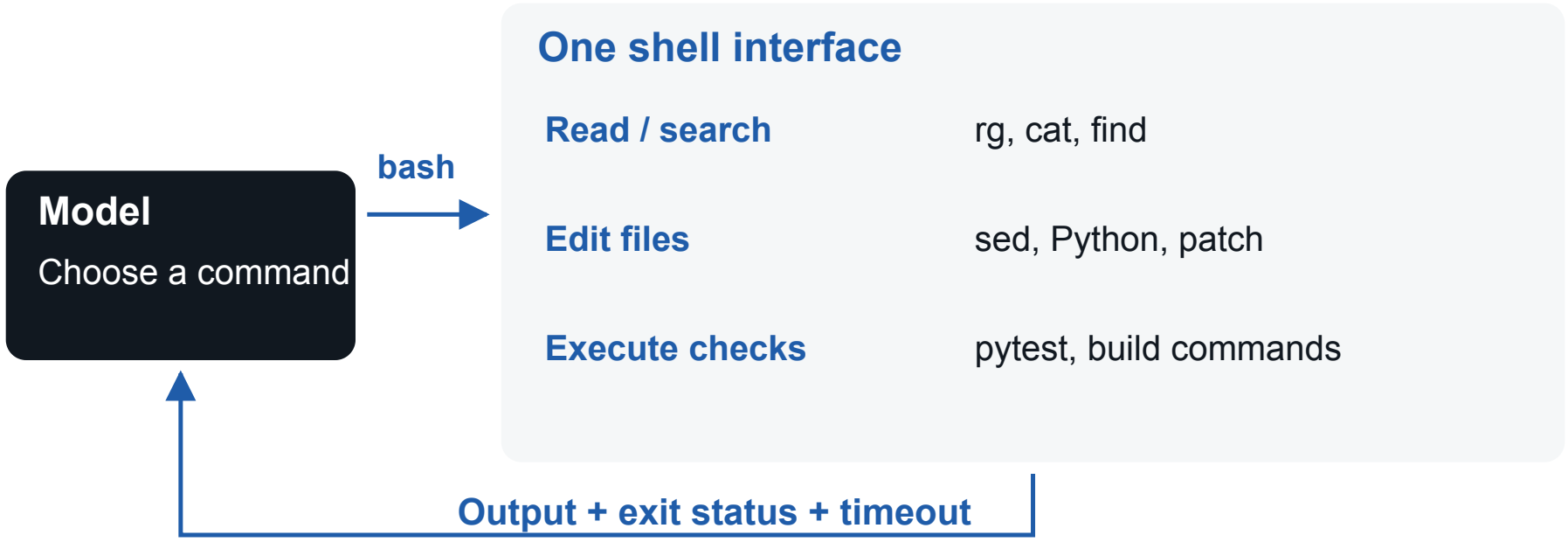
An alternative: a fixed workflow



Xia et al. · Agentless, original overview figure ↗

Coding toolsets

Coding with a shell-only toolset



mini-SWE-agent · shell-only interface and execution model ↗

Editing with shell commands

Replace text with sed

```
sed 's/^RETRIES = 3$/RETRIES = 5/' \
  settings.py > settings.new
mv settings.new settings.py
```

Match the line, write the result, replace the file.

One shell, many editing methods

sed / awk

Text patterns and line-based changes

Python scripts

Compute replacements across files

Whole files and text replacements

Whole file · settings.py

```
RETRIES = 5
TIMEOUT = 30
```

Simple to apply; repeats unchanged code.

Pi write · OpenCode write · OpenHands create

Search / replace · send old and new

```
settings.py
<<<<<<< SEARCH
RETRIES = 3
=======
RETRIES = 5
>>>>>>> REPLACE
```

Compact; needs an unambiguous match.

OpenHands · Pi · OpenCode edit

Aider · formats ↗ OpenHands ↗ Pi ↗ OpenCode ↗

Diffs and patch commands

Unified diff

```
--- a/settings.py
+++ b/settings.py
@@ -1,2 +1,2 @@
-RETRIES = 3
+RETRIES = 5
 TIMEOUT = 30
```

Standard: line positions + context.
Aider: omit hunk line numbers.

File-operation patch · Codex syntax

```
*** Begin Patch
*** Update File: settings.py
@@
-RETRIES = 3
+RETRIES = 5
 TIMEOUT = 30
*** End Patch
```

File operations + context; no line counts.

Codex · OpenCode / OpenHands (preset-dependent)

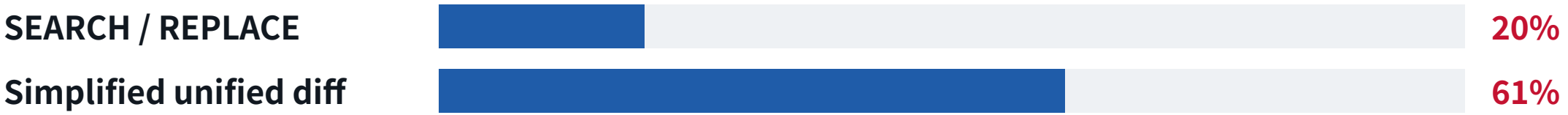
Aider · unified diffs ↗ Codex · parser ↗ OpenCode · selection ↗ OpenHands · GPT-5 preset ↗

Diff format can make a big difference

Method inside a large class → Move to a top-level function → Check syntax and code preservation

Aider refactoring benchmark: **89 Python tasks**

GPT-4 Turbo (gpt-4-1106-preview) · benchmark success rate



Aider · unified-diff experiment and refactoring benchmark ↗

Finding relevant code

Start with the symptom

```
rg 'retries' .
```

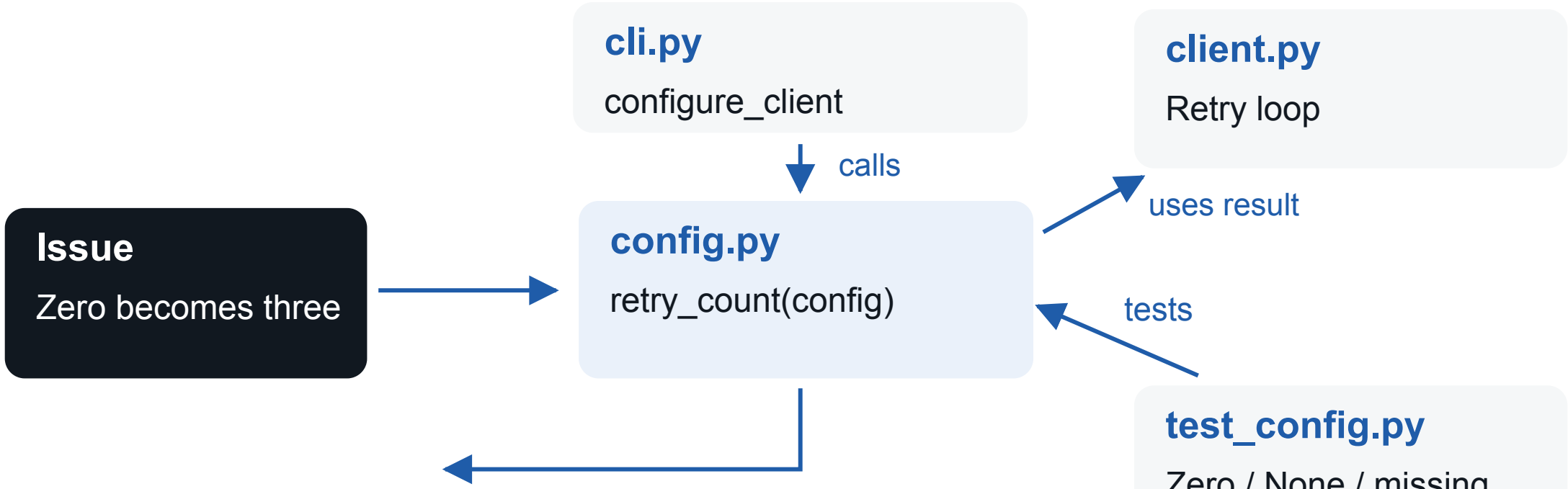
A flag, a loader, and a retry loop may all match.

Follow the value

> **Configuration input**
>
> Where is zero first read?

> **Defaulting logic**
>
> Where does zero become three?

Navigating code with dependency graphs

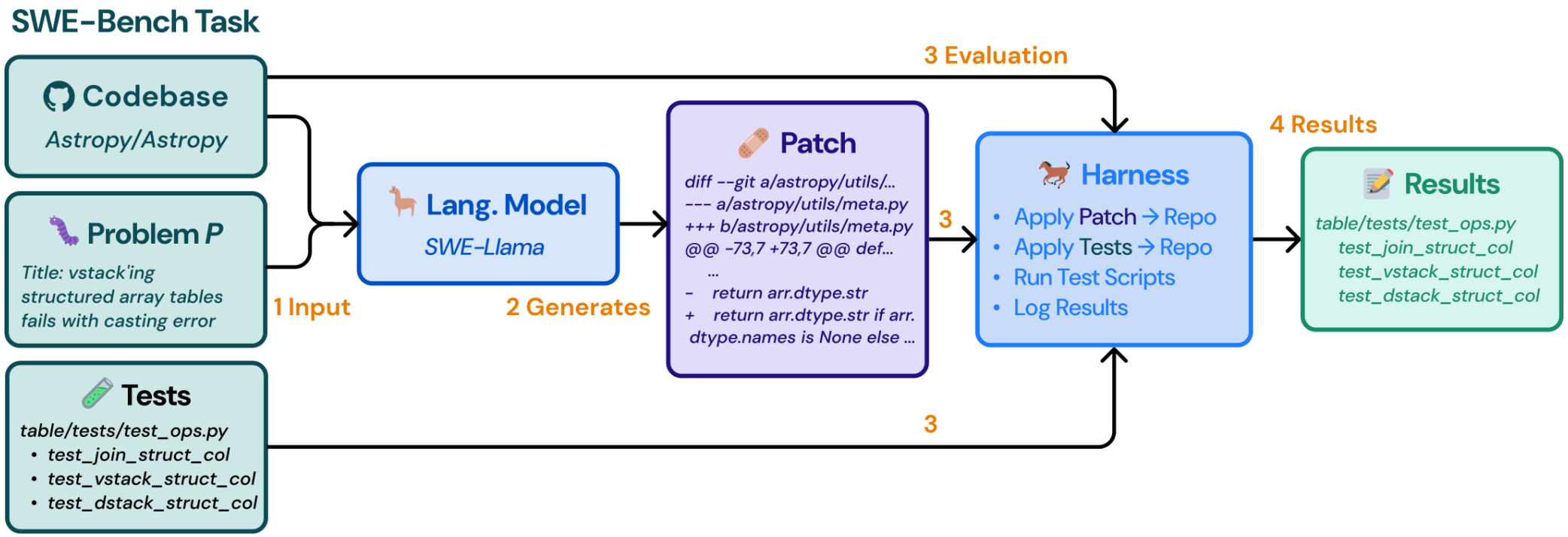


Retrieve the relevant neighborhood,
then check it with execution.

Chen et al., 2025 · LocAgent ↗

Coding agent evaluation and training

Evaluating issue resolution



Fail → pass requested repair

Pass → pass preserved behavior

Jimenez et al. · SWE-bench evaluation pipeline, original source figure ↗

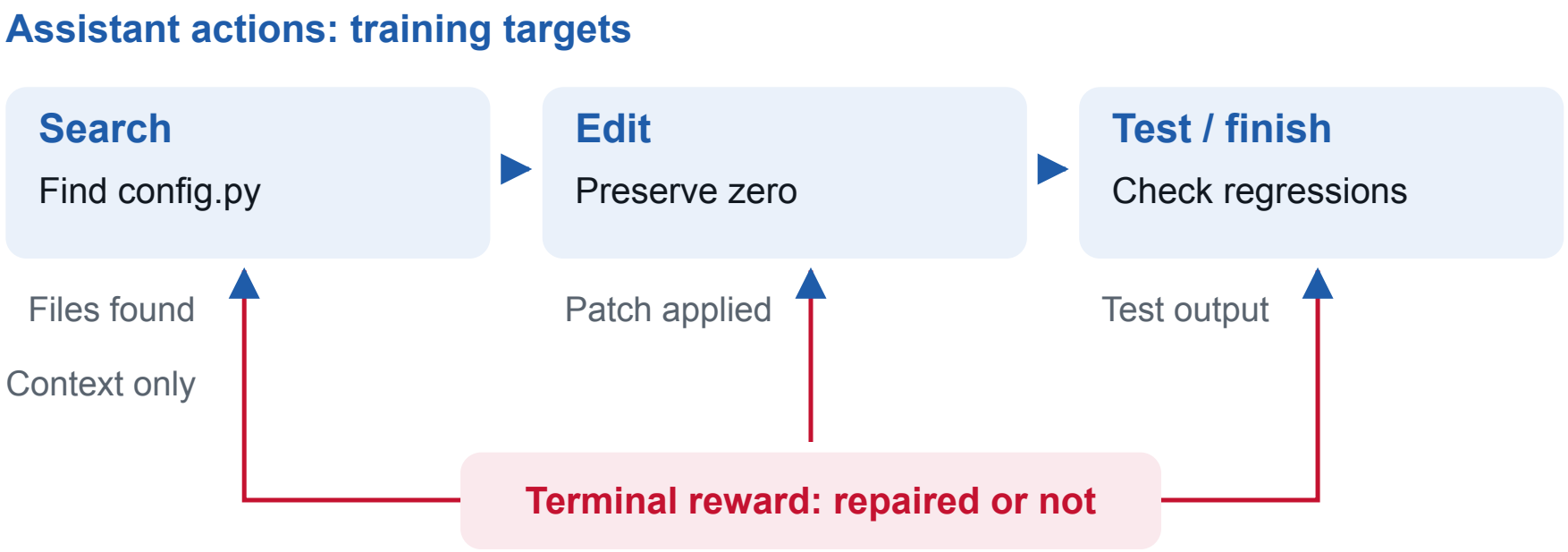
Building runnable training tasks

Runnable task	
Starting state	Files + dependencies + test command
Issue	Requested behavior + reproduction
Checks	Bug fails before; repair restores behavior

Validate the environment before collecting training trajectories.

SWE-Gym · executable training environments ↗

RL for multi-step code repairs



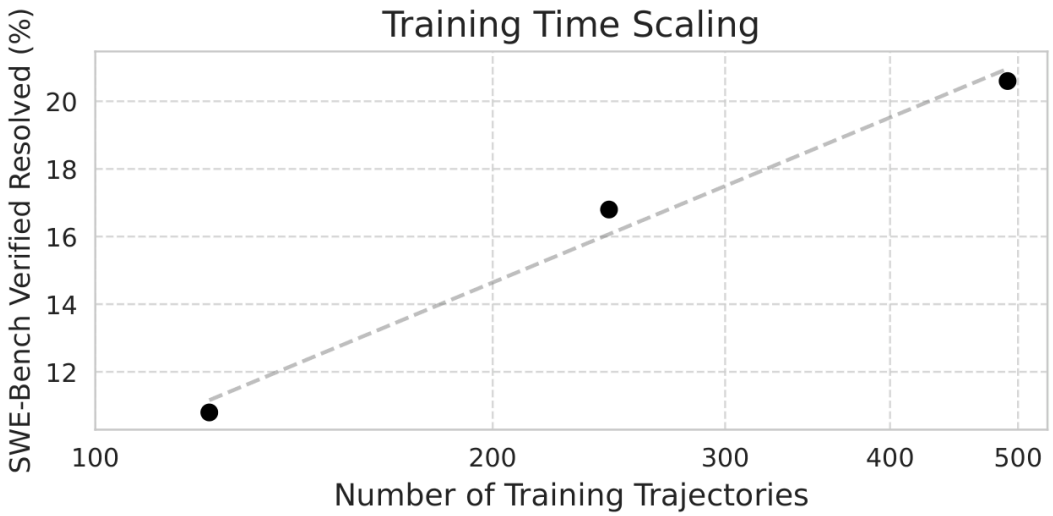
A late failure does not identify which earlier decision was wrong.

$$\max_\theta \ \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)]$$

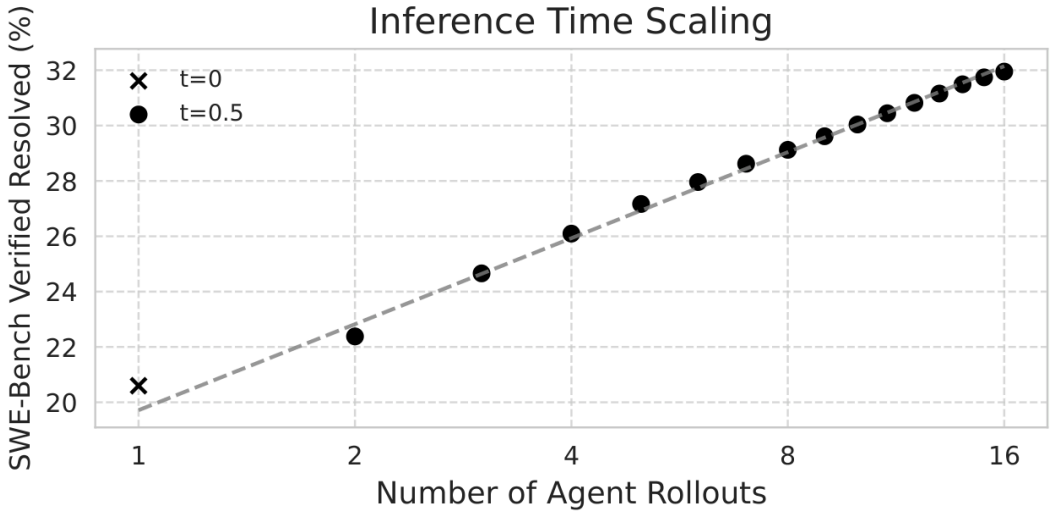
Jain et al., 2025 · R2E-Gym ↗

Training data and inference compute

Training scale

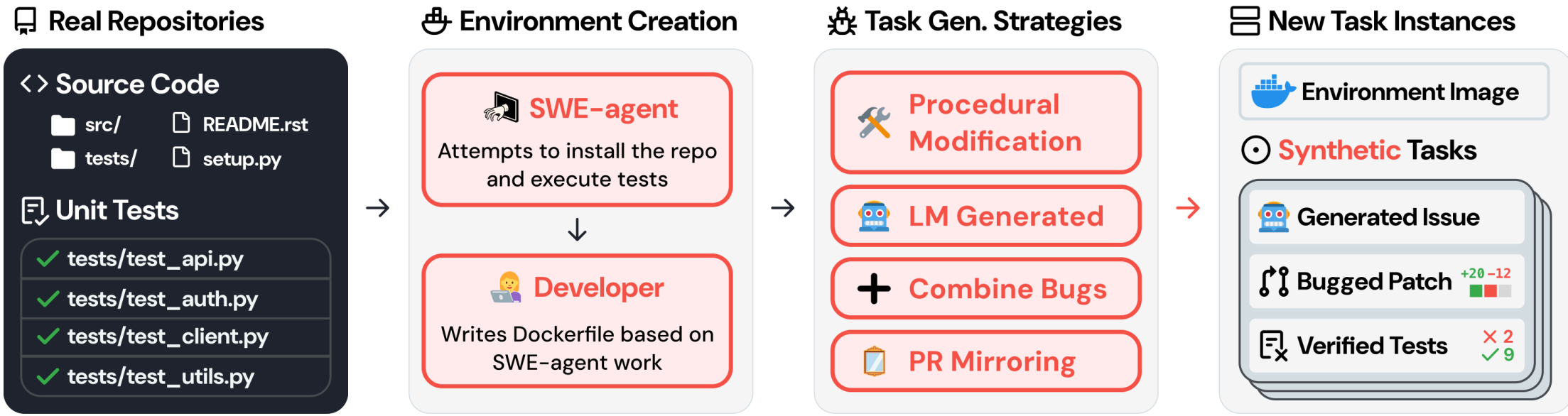


Inference scale: learned-verifier selection



Pan et al., 2024 · SWE-Gym · 32B model ↗

Creating repair tasks by injecting bugs



Prepare executable baseline

Mutate code, not tests

Keep new test failures

Yang et al., 2025 · SWE-smith, original pipeline figure ↗

Injecting and checking a bug

Working baseline

```
value = config.get("retries")
return 3 if value is None else value
```

All four existing tests pass.

Mutate the condition

```
value = config.get("retries")
return 3 if not value else value
```

Zero now becomes three; its test fails.

Before 4 pass

Mutated 3 pass · 1 fail

Restored 4 pass

SWE-smith · procedural mutation and execution validation ↗

Working across programming languages

Python

- pyproject.toml
- src/
- tests/

pytest

TypeScript

- package.json
- src/
- tests/

npm test

Rust

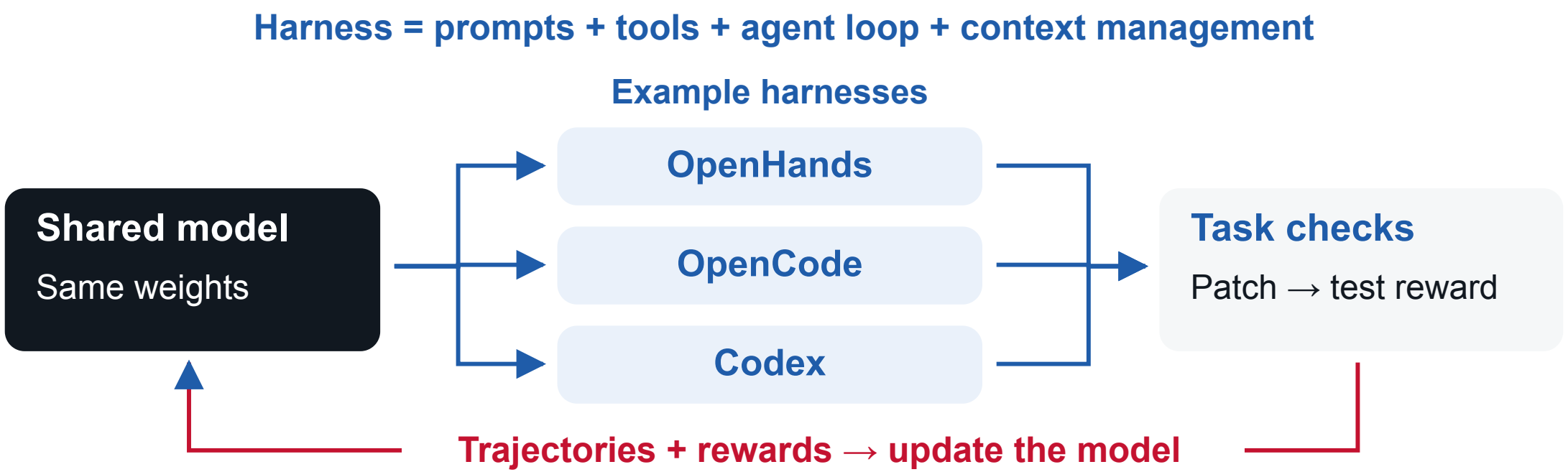
- Cargo.toml
- src/
- tests/

cargo test

Read the project configuration before choosing build and test commands.

Multi-SWE-bench ↗ SWE-bench Multilingual ↗

Multi-harness training



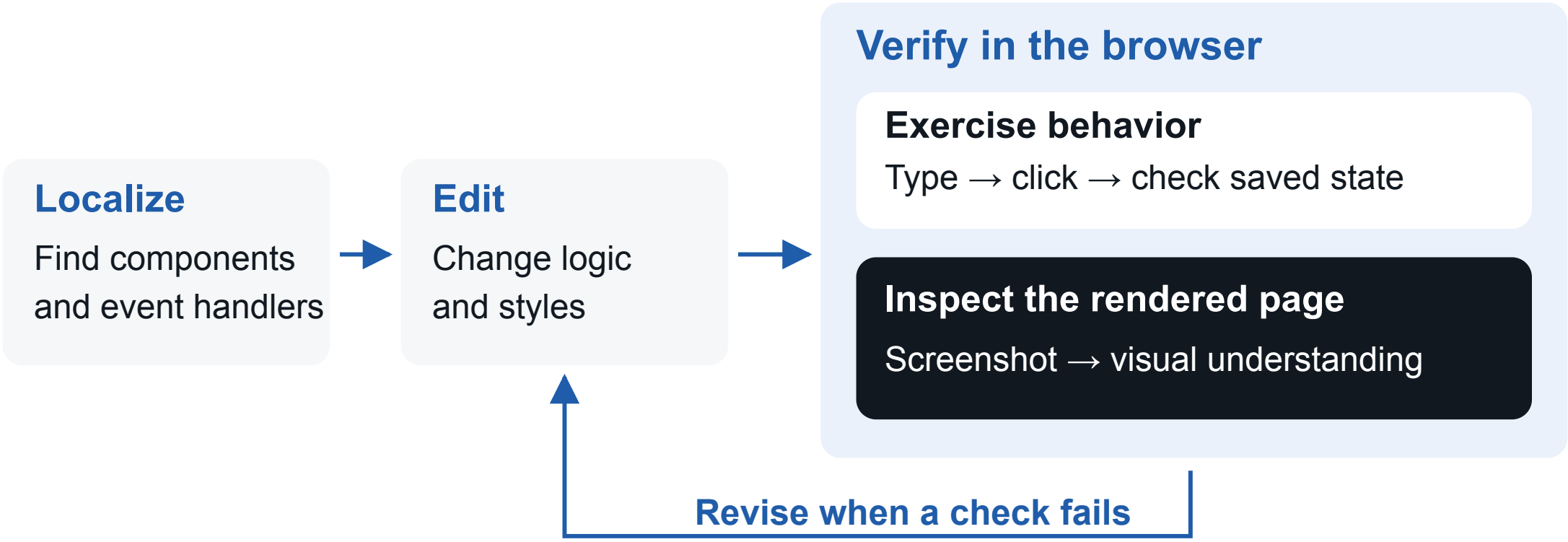
Nemotron 3 Ultra: ≥2 harnesses per task distribution

Polar: native harnesses call a shared model API proxy

Nemotron 3 Super ↗ Nemotron 3 Ultra ↗ Polar ↗ Harness implementations ↗

Frontend development

Agent-powered frontend development

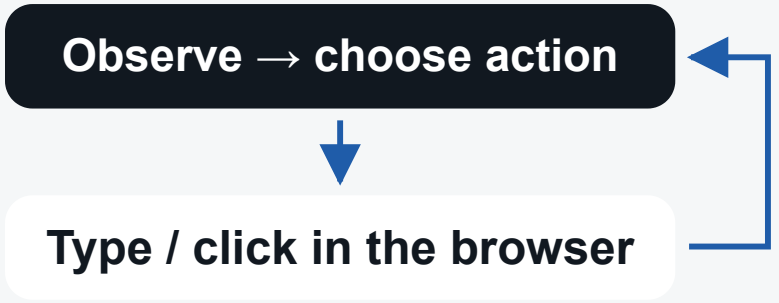


Yang et al., 2024 · SWE-bench Multimodal · browser and screenshot tools for SWE-agent ↗

Browser agents and automation scripts

Browser agent

Model chooses each next action



Adaptive exploration · next class

Playwright script

Model writes a repeatable check

1	Fill Name with spaces	fill()
2	Click Save contact	click()
3	Assert saved state	expect()
4	Capture an image	screenshot()

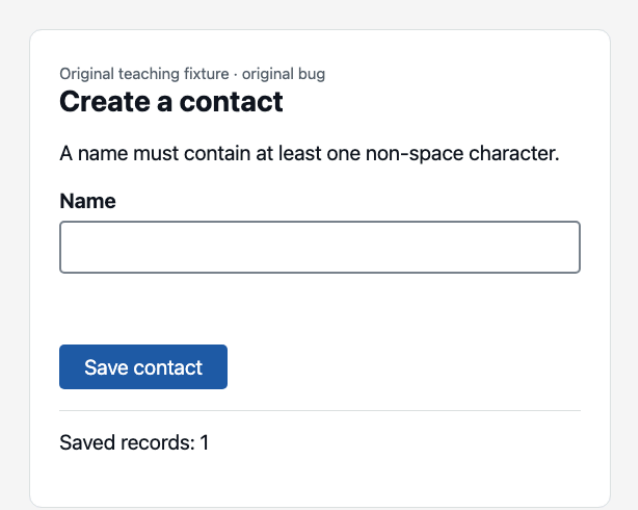
Programmed sequence + assertions

Screenshots → visual inspection or approved-baseline comparison

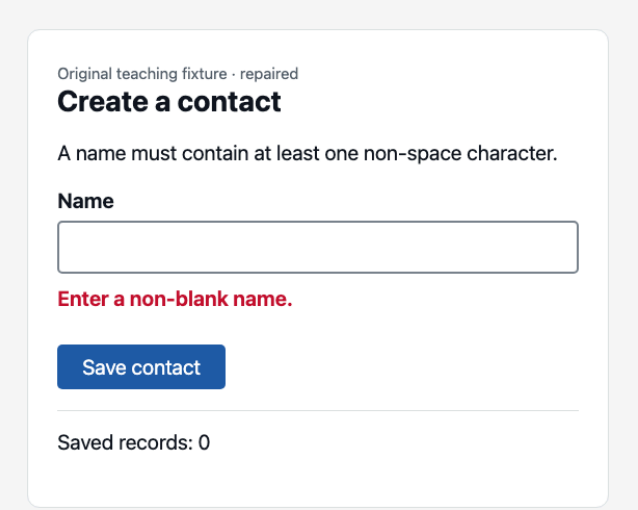
Koh et al., 2024 · VisualWebArena ↗ Playwright actions ↗ Assertions ↗ Screenshot comparisons ↗

Verifying a frontend repair

Before: blank record saved



After: blank name rejected



Edit `value.length` → `value.trim().length`

Behavior

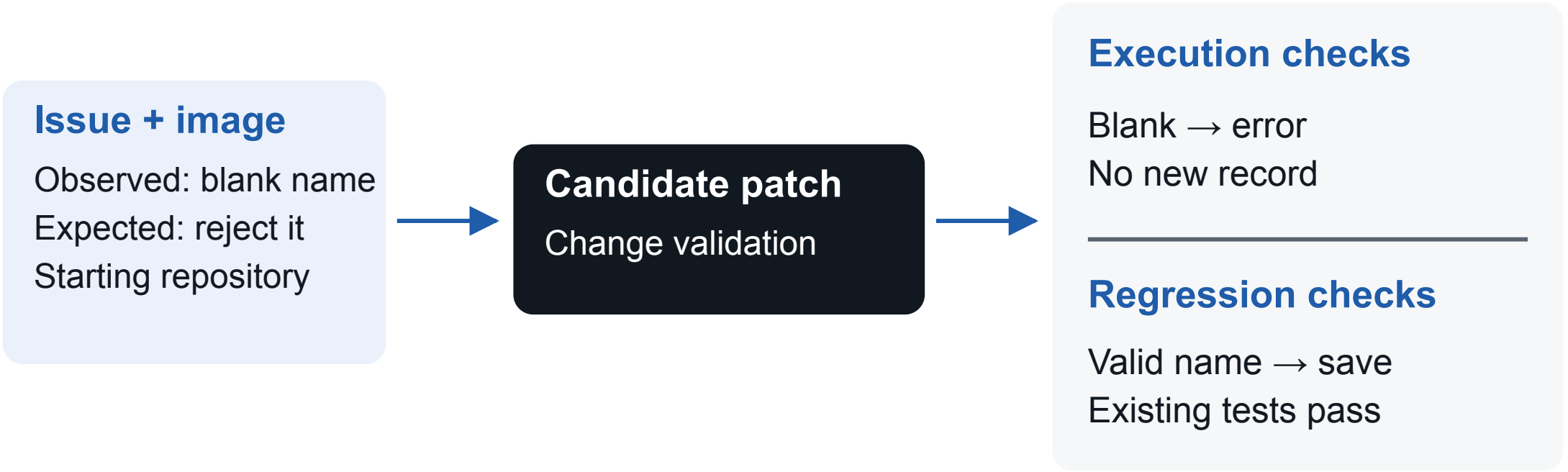
0 records; valid names still save

Appearance

Error visible and readable beside Name

Runnable classroom example ↗ Playwright assertions ↗ Visual checks ↗

Evaluating visual issue resolution

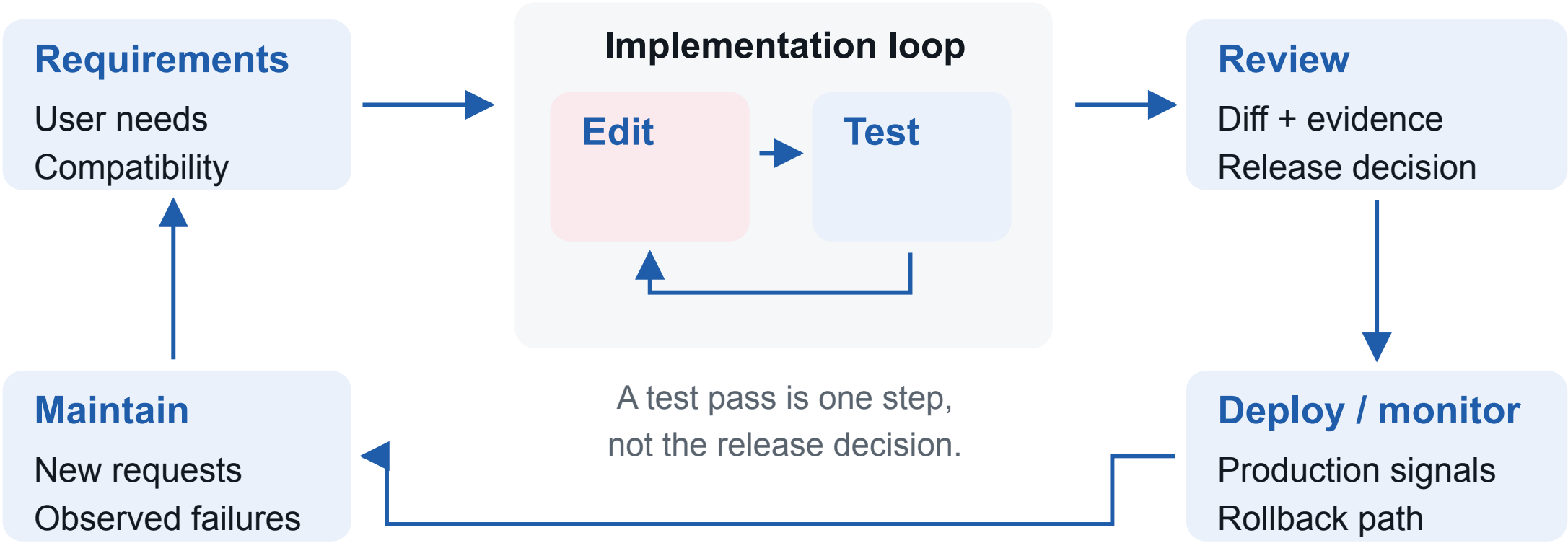


An image in the issue does not make image similarity the grading rule.

SWE-bench Multimodal · issue images and test-based patch evaluation ↗

Broader software development

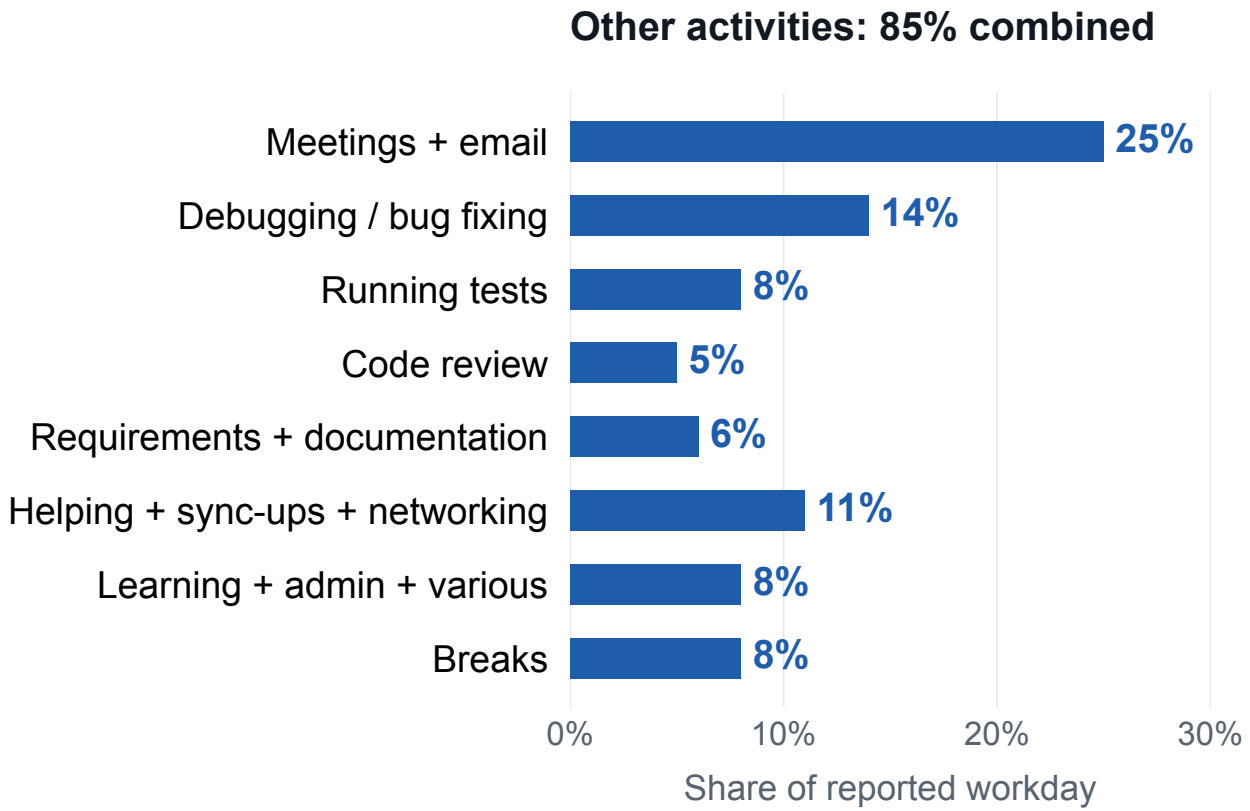
From requirements to maintenance



Where developers spend their time

Coding

Reading / writing code and tests

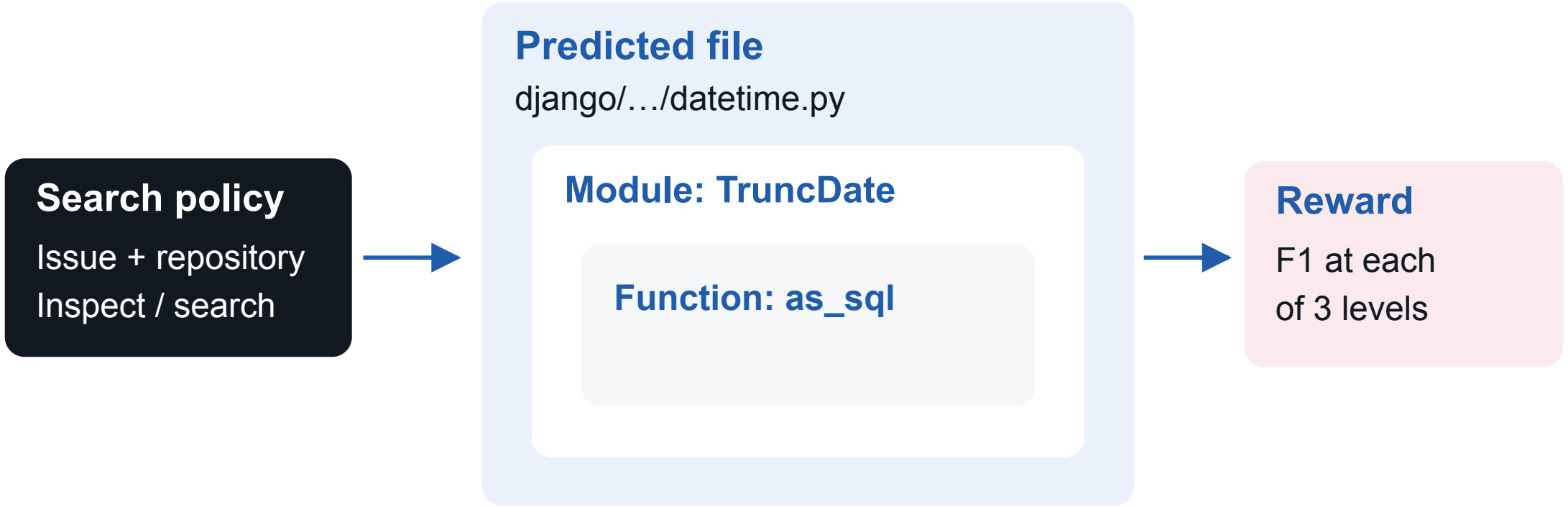


Meyer et al., 2019 · Table 2 · 5,928 self-reported Microsoft workdays ↗

Comparing software development tasks

Task	Artifact	Environment / verifier	Horizon
Repair	Patch	Repository / regression tests	One issue
Library creation	Implementation	Scaffold / package tests	Many functions
Test generation	New test	Buggy + fixed versions / discrimination	One behavior
CI repair	Code or configuration	Workflow / required checks	Build run
Evolution	Accumulated changes	Persistent app / milestone + regressions	Dependent tasks

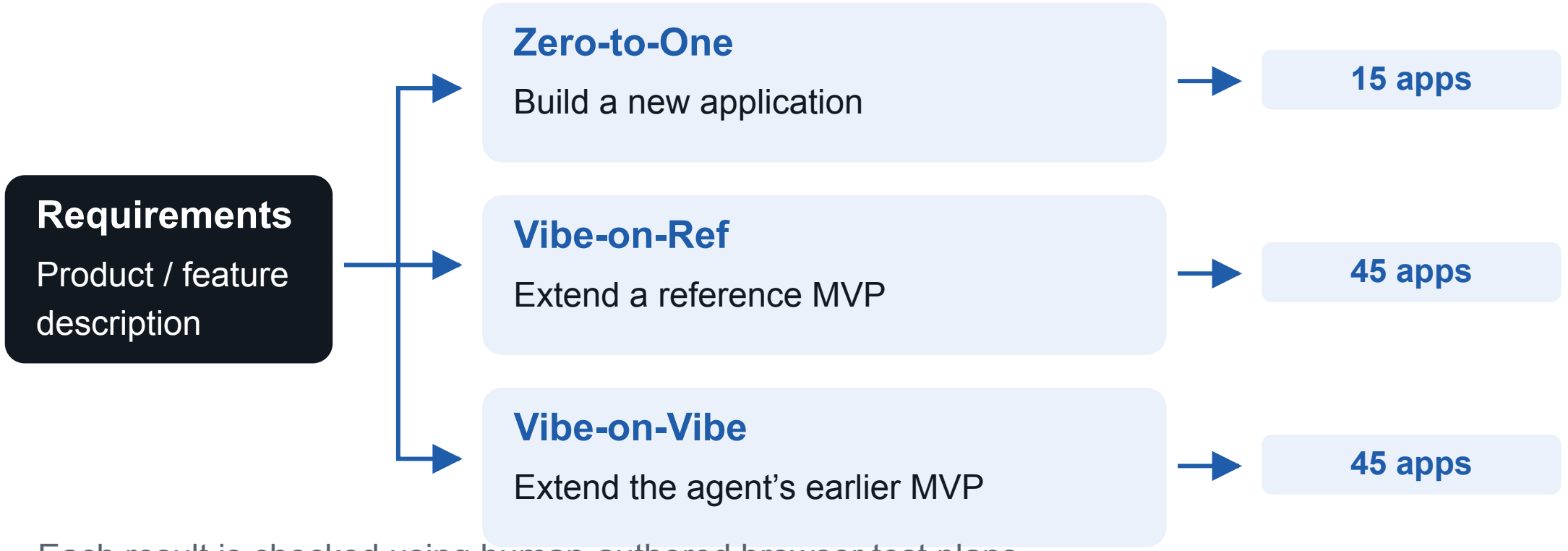
Task: localization



Compare with locations in the gold patch; the patch is not policy input.

CodeScout, 2026 ↗

Task: app creation and modification



Each result is checked using human-authored browser test plans.

ViBench · collection and evaluator definitions checked 2026-09-07 ↗

Task: library implementation

Commit0

Library scaffold

parse(…) → [missing body]
evaluate(…) → [missing body]
Documentation + tests

Implement together

Run integrated package tests

ProgramBench

Reference executable

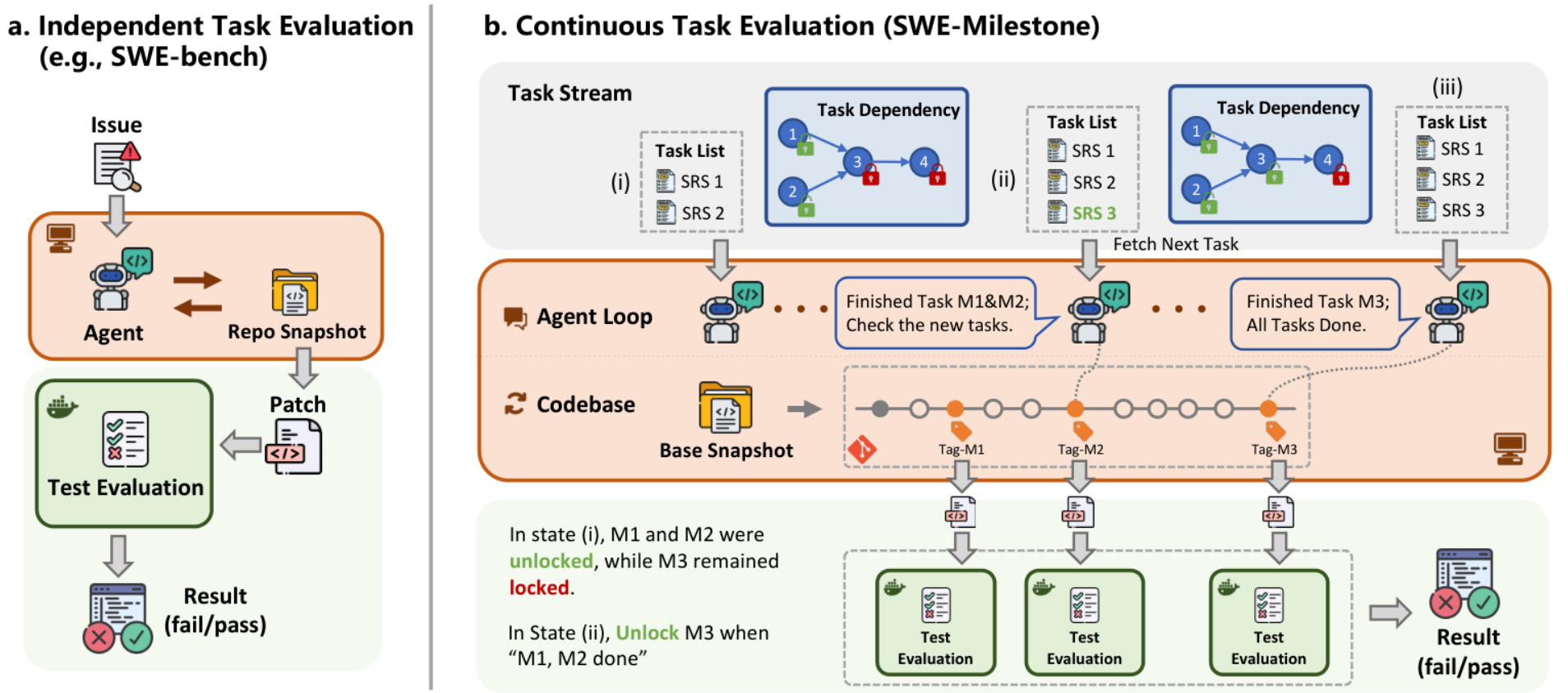
Probe inputs → observe outputs

New implementation

Match observable behavior
Check against the reference
Different internal designs can work

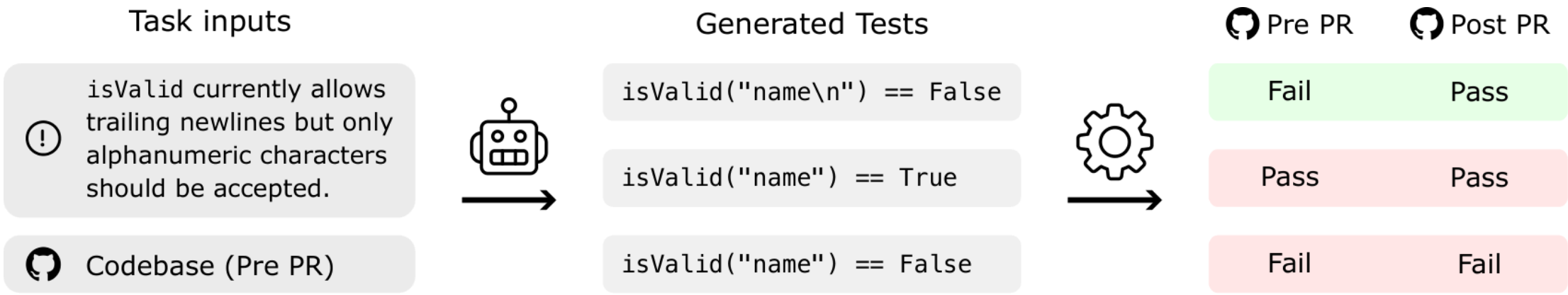
Commit0 ↗ ProgramBench ↗

Task: software maintenance



SWE-Milestone, 2026 ↗

Task: test generation



Mündler et al., 2024 · SWT-Bench ↗

Task: CI repair

Observed build failure

```
Runtime: Node 16
Package requires: Node >=18
npm ci: EBADENGINE
```

Inspect the package’s supported runtime.

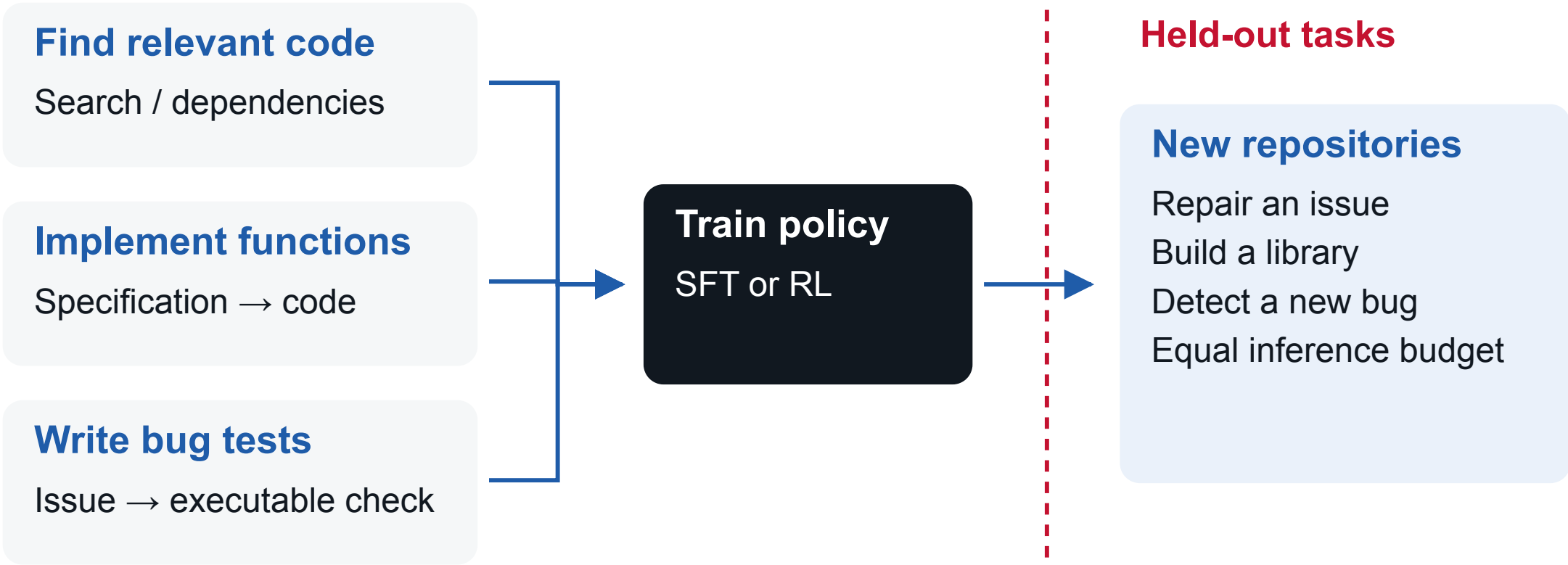
Causally relevant change

```yaml
# setup-node configuration
with:
  node-version: '20'
```

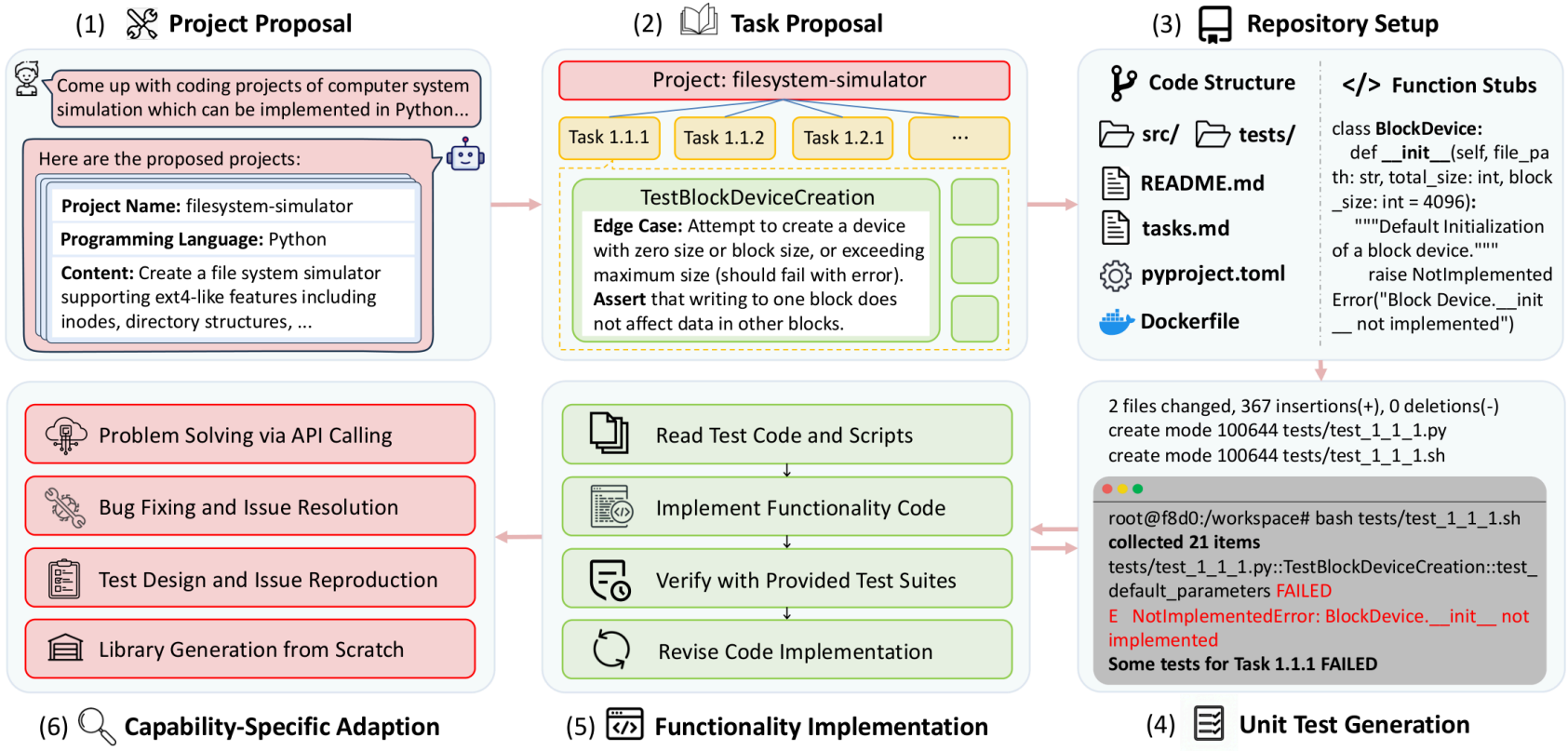
Rerun install, build, and tests.

LCA CI Builds Repair · dataset revision ebf12dad ↗

Learning skills for new tasks

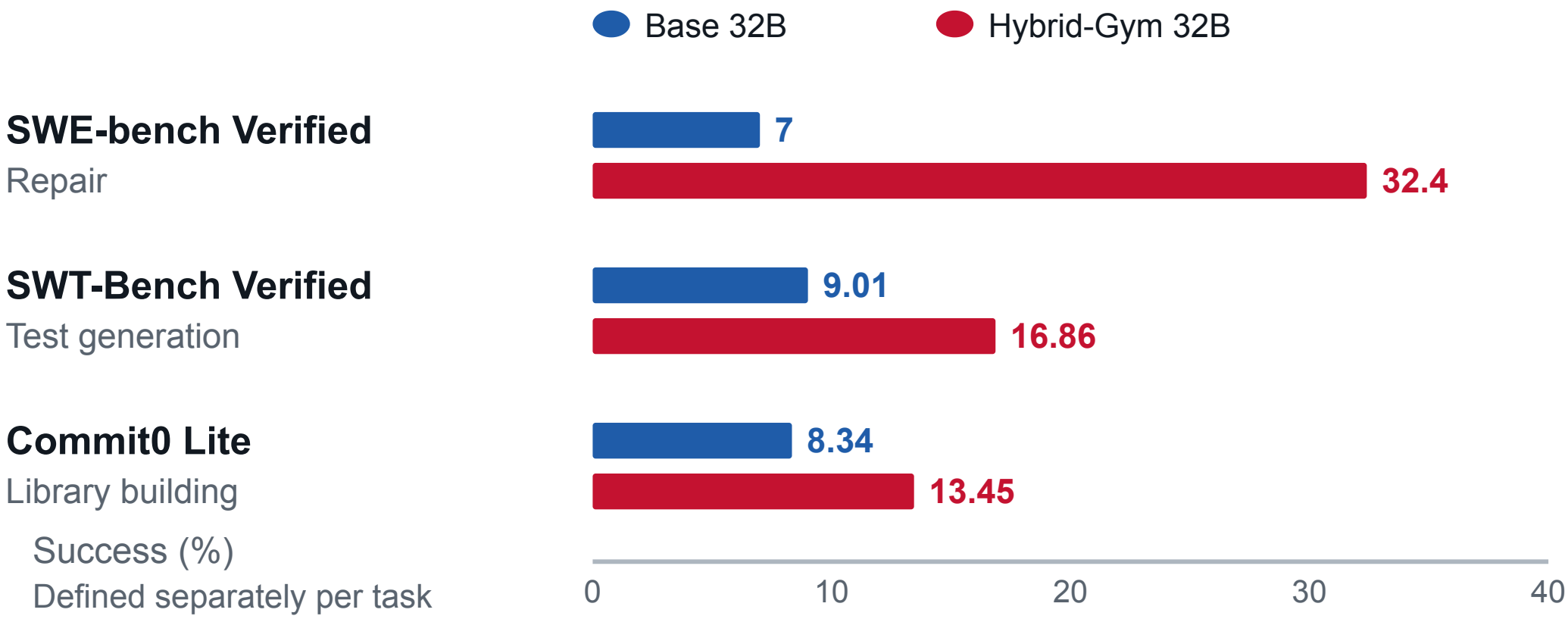


Generating projects for coding practice



Zhu, Gandhi & Neubig · SWE-Playground ↗

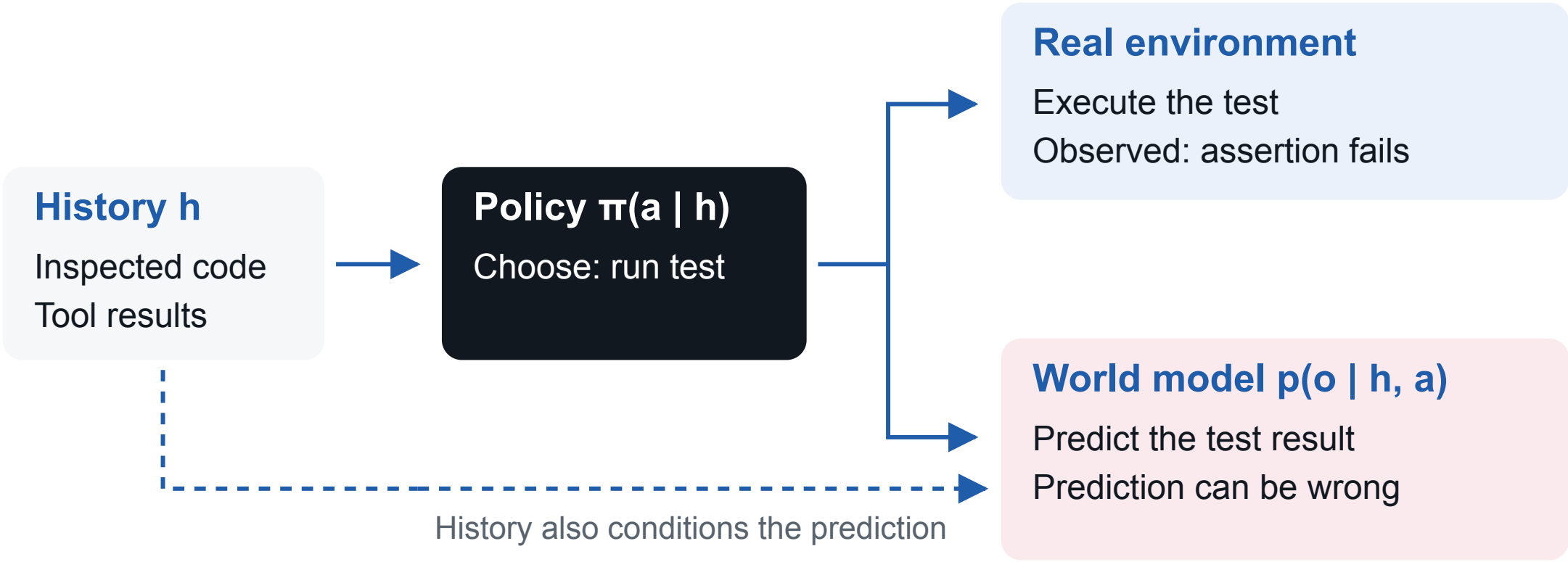
Transferring training to new development tasks



Hybrid-Gym, 2026 · selected 32B results ↗

Models that predict code behavior

Choosing actions and predicting their effects



Predicting execution: shared references

Predict the next state

```
items = [1]
alias = items
alias.append(2)
result = len(items)
```

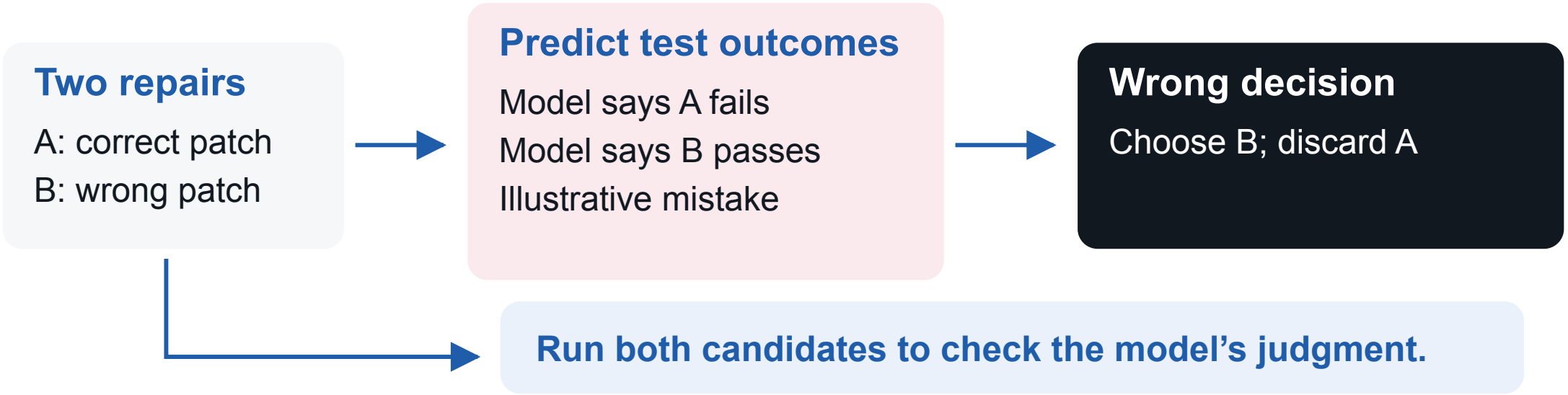
What are items and result?

True execution

```
items  = [1, 2]
alias  = [1, 2]
result = 2
```

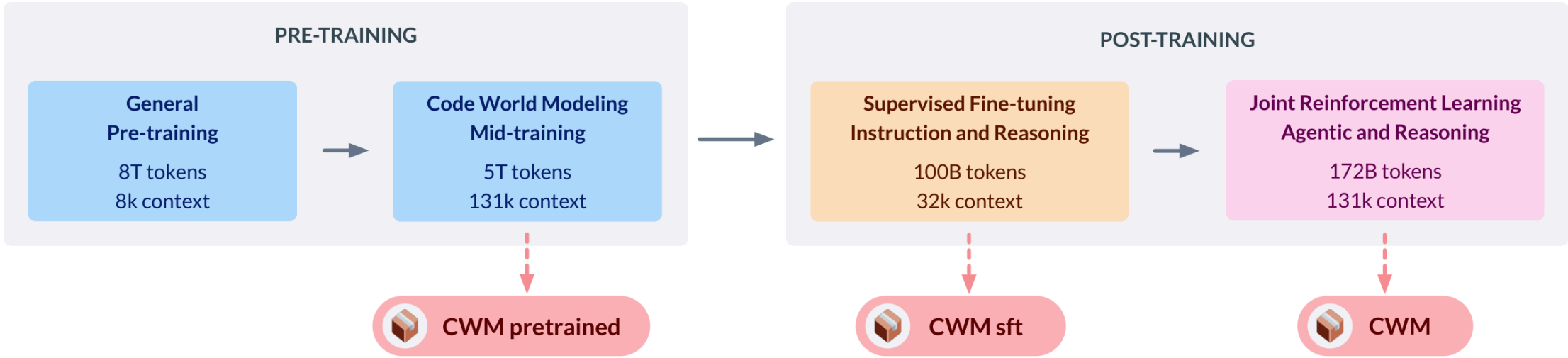
Wrong prediction: treating alias as a copy.

When predictions help an agent



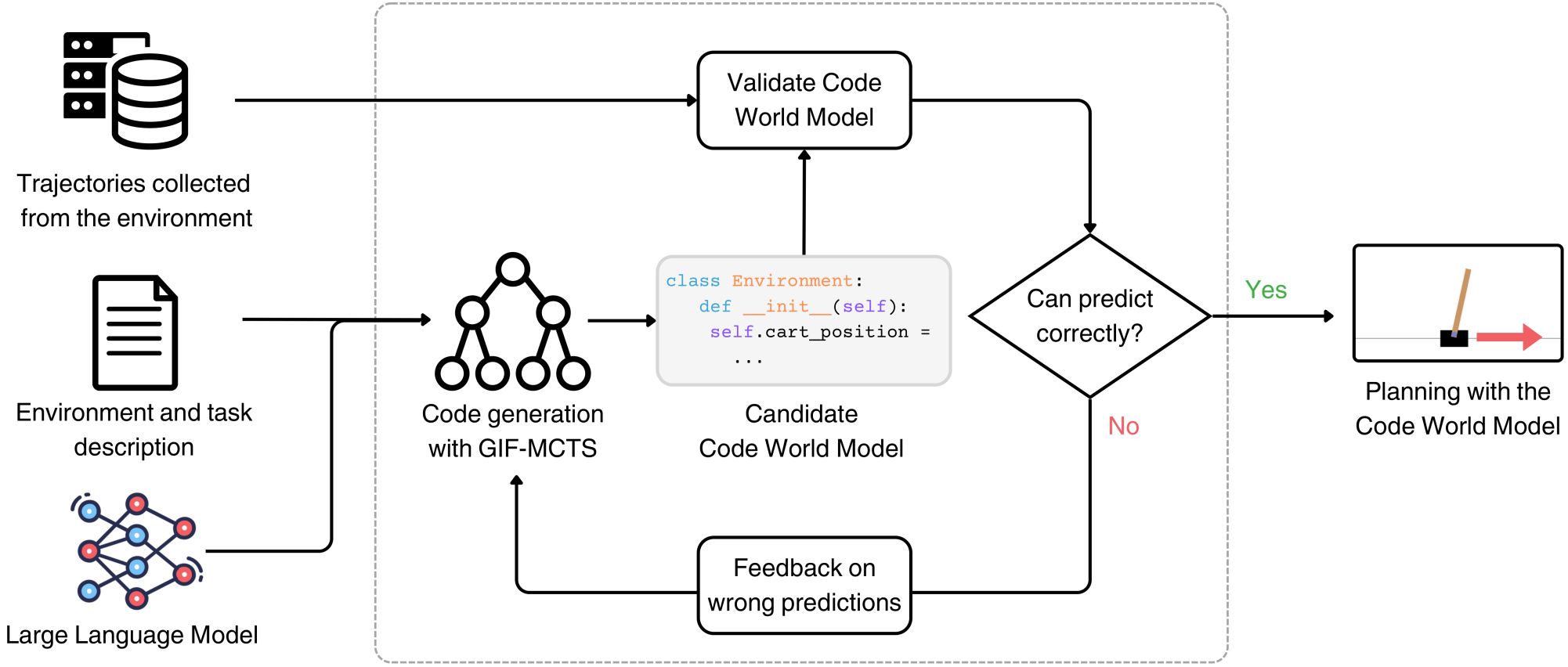
Compare task success AND total latency / tokens / tool calls.

Learning from program execution



FAIR CodeGen Team et al., 2025 · CWM ↗

Building and using simulators



Dainese et al., 2024 · Code World Models / GIF-MCTS ↗

Open questions about predicting code behavior

- **Better decisions?** Measure prediction errors and task success on the same problems.
- **Lower cost?** Include latency, tokens, and real environment calls.
- **New settings?** Hold out programs, dependencies, and environments.
- **When to execute?** Vary the fallback threshold and measure failures as well as savings.

Conclusion

Designing a coding agent

- **Code models:** pre-training, mid-training, and RL from execution rewards.
- **Agentic coding:** localize, edit, and verify with the right tools.
- **Agent training:** runnable tasks, repair rewards, and diverse harnesses.
- **Frontend development:** browser interaction and visual verification.
- **Broader development:** localization, apps, libraries, maintenance, tests, and CI.
- **Code-behavior prediction:** learned models that support execution and planning.

Questions

Next Class: Domains 2 — GUI Agents