Rewards and benchmarks

Key takeaways

1. Reinforcement learning uses scores from the model's own attempts to update its weights.

2. A checker determines which behavior earns a reward.

3. Evidence of progress requires separate evaluation tasks and comparable conditions.

Common misunderstandings of model scores

Which claim would you defend right now? Choose before the evidence.

"Running tests and asking an agent to retry trains the model's weights."

"A passing reward proves that the program implements the whole request."

"A higher benchmark percentage identifies the better model."

Keep your choice. Look for the mechanism or counterexample that would make it fail.

Demonstrations and scored attempts

“Write a Python function that reverses a string.”

Instruction tuning

Train on the tokens of a chosen response.

One demonstration

```
def reverse(s):
    return s[::-1]
```

Reinforcement learning

Sample responses from the model and score each one.

A checker can accept slicing or a correct loop.

A reward is a numerical score for an attempt.

source: Ouyang et al., InstructGPT (arxiv.org/abs/2203.02155, 2022) · DeepSeek-R1, §§2.2.1–2.2.2 (arxiv.org/html/2501.12948v1, 2025)

A checker turns an attempt into a reward

Two illustrative responses

Candidate functions

```
def reverse_a(s):
    return s[::-1]

def reverse_b(s):
    return s
```

External checker

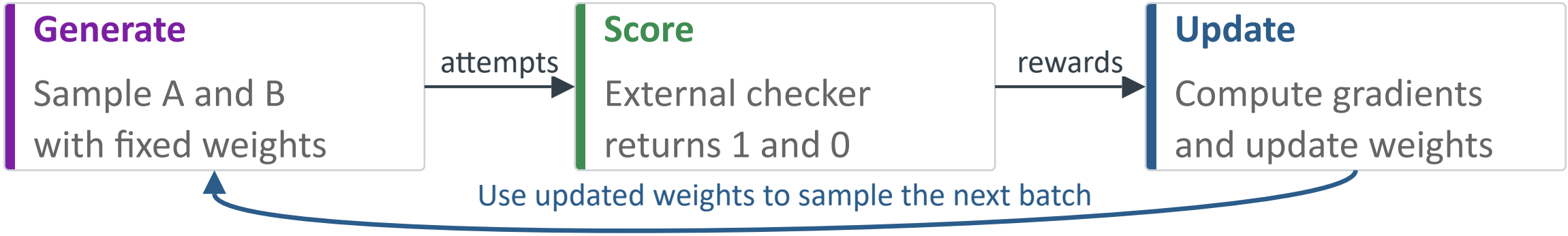
```
def reward(candidate):
    actual = candidate("hello")
    return int(actual == "olleh")

reward(reverse_a)  # 1
reward(reverse_b)  # 0
```

The checker supplies one number for the entire response.

Generate, score, update

Weights stay fixed during generation and scoring. The optimizer then changes them.



source: DeepSeek-R1, §§2.2.1–2.2.2 (arxiv.org/html/2501.12948v1, 2025)

Reward weights the response loss

A simplified policy-gradient loss. Practical methods constrain the update.

Sum the generated tokens' log probabilities, each conditioned on the prompt and earlier tokens.

```
response_logp = sum(token_log_probs)
baseline = 0.5  # fixed comparison score
response_term = -(reward - baseline) * response_logp
# Average the response terms for the batch loss.
```

Response	Reward	Reward – baseline	Effect the loss favors
A	1	+0.5	Higher response probability
B	0	−0.5	Lower response probability

Backpropagate through the model's log probabilities. No gradient goes through the checker. Sampled tokens, rewards, and the baseline stay fixed.

A retry uses the same weights

RL training

Generate attempts and score them.

An optimizer writes updated weights for future calls.

Inference

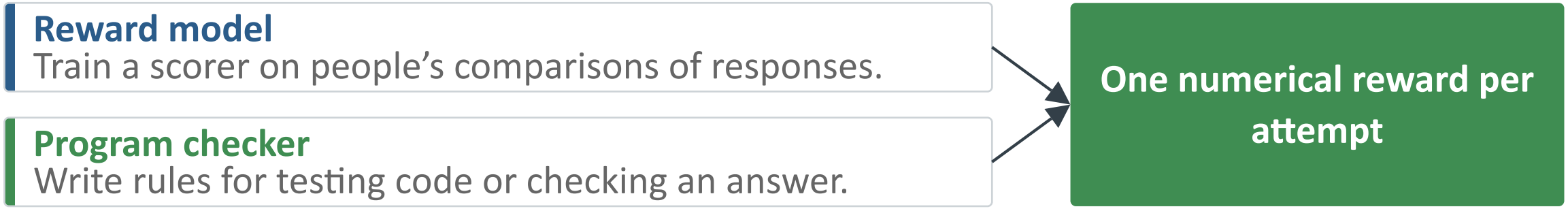
Use a test result in context and generate another attempt.

Longer reasoning or another retry generates more tokens with the same weights.

Both use compute to generate tokens. Training also computes an update.

Human comparisons and program checks

The same training loop can use either source of reward.



A reward model reflects the judgments it learned. A checker covers the cases it runs.

source: Ouyang et al., InstructGPT (arxiv.org/abs/2203.02155, 2022) · DeepSeek-R1, §§2.2.1–2.2.2 (arxiv.org/html/2501.12948v1, 2025)

A checker can reward the wrong program

A third illustrative response

```
def reverse_c(s):
    return "olleh"
```

Input	Expected	A returns	C returns
“hello” · rewarded case	“olleh”	“olleh” · pass	“olleh” · pass
“world” · fresh case	“dlrow”	“dlrow” · pass	“olleh” · fail

The checker and expected answers are protected from candidate writes.

A and C both earn 1 on the training case. A fresh input exposes C.

Training, development, and final evaluation

Training tasks	Use the rewards to update model weights.
Development tasks	Use results to choose settings or revise the agent.
Final evaluation tasks	Measure the chosen system on fresh tasks.

Once you tune against final results, those tasks become development evidence.

Public solutions can enter training. A model may then recall an answer during evaluation.

A SWE-bench task

Built from an issue and its merged fix

Solver inputs

The solver gets the issue text and the base commit, the repository version before the fix. It submits its own patch.

Benchmark construction

The reference patch implements the fix. The evaluation test patch adds or modifies tests.

With the test patch applied, compare results before and after the reference fix.

Selected tests	Without the reference fix	With the reference fix
FAIL_TO_PASS	Fail	Pass
PASS_TO_PASS	Pass	Pass

source: Jimenez et al., SWE-bench (arxiv.org/abs/2310.06770, ICLR 2024)

Instapoll

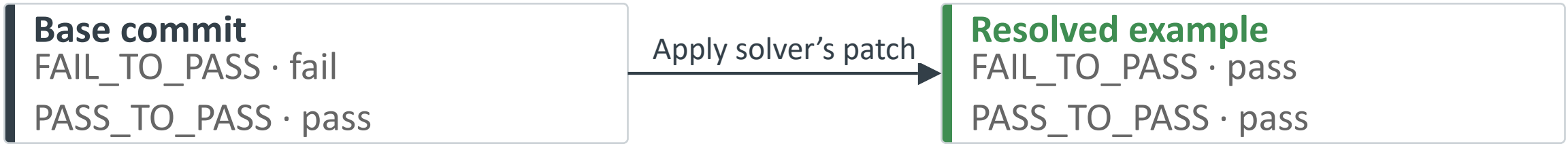
Instapoll · Review · Nov 17 A

A model’s patch for a SWE-bench task makes all its FAIL_TO_PASS tests pass. One test on the task’s selected PASS_TO_PASS list now fails. How does SWE-bench score the task?

- A. Resolved, because the patch passes all the FAIL_TO_PASS tests
- B. Partially resolved, with credit based on the fraction of selected tests that pass
- C. Unresolved, because the task requires every selected PASS_TO_PASS test to pass
- D. Referred for review to decide whether the failed test is related to the bug

What “resolved” means

Every test in both selected lists must pass on the solver’s patch.



The grader applies the evaluation test patch in both states. A different implementation can resolve the task by passing the same selected tests.

One failure in PASS_TO_PASS makes the task unresolved.

What a score compares

Two illustrative evaluation runs

The agent harness is the code around the model that builds prompts, runs tools, and decides when to retry or stop.

Condition	Run A	Run B
Task set	Same 60 issues	Same 60 issues
Model	Model A	Model B
Harness and tools	Same harness, shell, test runner	Same harness, shell, test runner
Budget per issue	1 attempt, 5 min total	3 attempts, 5 min total
Resolved	42 of 60	49 of 60

B solved more under its configuration. To measure the effect of changing the model, give both runs the same attempt limit.

Key Takeaways

1. The reward weights each sampled response's contribution to the loss. Gradients through its token log probabilities drive the optimizer's update.

2. The hardcoded function passed the one-case checker. A test result establishes what happened on the cases the checker ran.

3. Evaluate on fresh tasks after tuning. Hold the other conditions fixed when you want to attribute a score difference to one change.

Design the next comparison

10 min

- With the same model, the old harness solves 70 of 100 public issues and the new one solves 85 of the same 100. The old harness allows one attempt per issue, and the new one allows four.
- The developers repeatedly inspected those issues while tuning the new harness. You must decide whether to use it for a private service.
- In pairs, write one three-sentence proposal. State the claim you want to test, then the task set and budget, then the outcome you will measure.
- Use fresh tasks representative of the service and repeat each configuration to see how much results vary across runs.