META AI (FAIR) RESEARCH

RLEF:
Grounding Code LLMs
in Execution Feedback

Reinforcement Learning for Iterative Code Synthesis with Multi-Turn Execution Feedb
ack

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Quentin Carbonneaux
Taco Cohen, Gabriel Synnaeve

{jgehring,gab}@meta.com

Overview

A comprehensive journey through the RLEF methodology

LLM Agents and Execution Feedback

Two Crucial Skills for LLM Agents

1. Intent Deduction

Accurately understanding user intent when prompted, achieved through instruction fine-tuning according to user preferences (Ouyang et al., 2022; Rafailov et al., 2023)

2. Feedback Grounding

Taking into account feedback on intermediate actions to arrive at desired outcomes. Crucial for grounding generations in concrete situations.

Key Insight

Frame code generation as an iterative task with repeated attempts and execution feedback

→ Actions = code solutions

→ Observations = execution feedback

→ Reward = binary (tests pass/fail)

→ Optimization = end-to-end RL

</> The Code Generation Challenge

Problem: SOTA LLMs struggle to iteratively improve code compared to independent sampling (Kapoor et al., 2024; Xia et al., 2024)

Feedback Sources: Error messages, unit test results, compilation errors, runtime performance

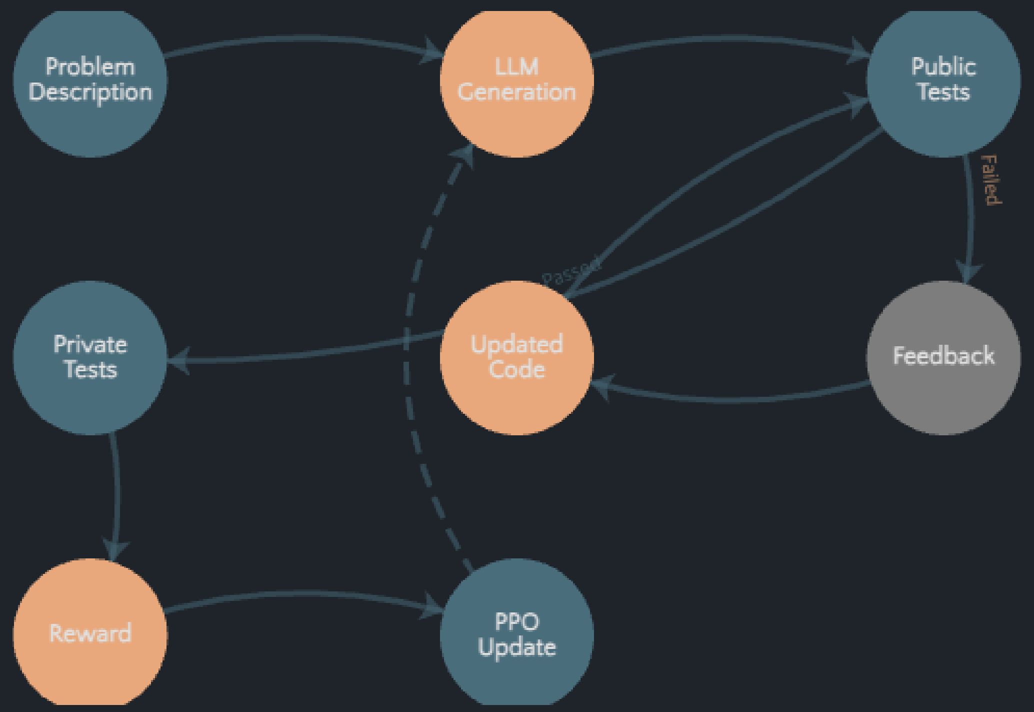
Gap: Utilizing execution feedback has failed to yield substantial improvements when considering computational demands

Autonomous LLM Applications

- **Web interaction:** Query search engines, interact with websites (Yao et al., 2022)
- **Information retrieval:** Ensure accurate answers with up-to-date data (Mialon et al., 2024)
- **Software development:** Generate code from high-level descriptions (Yang et al., 2024)

Complete Framework Overview

RLEF Training Loop



Two Test Sets

Public Tests

Provide execution feedback during attempts; accessible during it erative generation

Private Tests

Determine final correctness; compute binary reward for policy o ptimization

Key Benefits

- **Guards against shortcuts** — prevents copying expected outputs
- **Accelerates iteration** — limited public tests speed up generatio n
- **Enables RL training** — provides reward signal for optimization
- **Single model** — both synthesis and repair capabilities

Conversation Flow

Problem → Generate → Execute → Feedback → Improve → Repeat → Submit → Re ward → Update

Turn Limits

Episode terminates when public tests pass OR specified turn limit reached (default: 3 attempts)

Multi-Turn Conversation Setup

Conversation Flow

1 **Initial Prompt**
Start dialog with problem description, query LLM for initial solution

2 **Execute & Verify**
Verify solution against public test set → passed/failed tests, errors

3 **Format Feedback**
If any public test fails, format feedback and append to dialog

4 **Query Update**
Query LLM for updated solution with full context (problem + prev solutions + feedback)

5 **Submit Final**

Two Test Sets: Why Separate?

1. Guard Against Shortcuts

If test inputs/outputs are fixed, held-out tests prevent optimization shortcuts where LLM copies expected outputs based on execution feedback

2. Accelerate Iteration

Running full test suite may be computationally demanding. Limited public tests speed up iterative generation while maintaining effectiveness

Dialog Context Structure

```
# Turn 1
User: Problem description
Assistant: Code solution
# Turn 2
User: Feedback + "Give it another try"
Assistant: Updated code
# ... repeat
```

Execution Feedback Types

- ✓ Wrong Answer
- ✓ Exception
- ✓ Timeout
- ✓ Out of Memory

RLEF: MDP Formulation & PPO

MDP Components

Policy π: Language model

Observations o_0: Problem description

Actions a_t: Textual responses (code)

Observations o_t: Past observations, actions, execution feed back

Termination: Public tests pass OR turn limit reached

Reward: Binary (all tests pass/fail)

Reward Function

$R(s_t, a_t) = r(s_t, a_t) - \beta \log[\pi(a_t|c_t)/\rho(a_t|c_t)]$

$r(s_t, a_t) = +1$ (all tests pass)

$r(s_t, a_t) = -1$ (any test fails)

$r(s_t, a_t) = -0.2$ (invalid code)

$\beta = 0.05$ (KL regularization)

Hybrid Architecture

Token-Level Policy

Model policy at token level for fine-grained optimization

Turn-Level Value

Predict response value from last token of prompt; single ad vantage per response

Early experiments show this hybrid approach works bes t

PPO Optimization

Algorithm: Proximal Policy Optimization

Baseline: Value function $V(c_t)$

Advantage: $A_t = -V(c_t) + \Sigma R(s_i, a_i)$

Learning rate: 2e-7 (AdamW)

Discount γ: 1 (no discounting)

KL penalty: Geometric mean (not product)

Training updates: 12k (8B), 8k (70B)

GPUs: 288 (8B), 2,304 (70B)

Failure Mode Addressed

Issue: Invalid code in non-final responses

Solution: Small penalty (-0.2) for invalid responses

KL Penalty Design

Geometric mean of token probabilities counters bias towar d shorter generations, especially for non-final responses

03 / EXPERIMENTAL SETUP

CodeContests Benchmark & Models

CodeContests Benchmark

Competitive programming problems with natural language descriptions, public/private tests. High difficulty: algorithms, data structures, runtime efficiency.

13,328	117	165
Training	Validation	Test

Test Set Characteristics

Public Tests

1-7 test cases (median: 1) for feedback

Private Tests

Hidden tests for final correctness evaluation

Llama 3 Models

Strong instruction-following and code generation capabilities.

Llama 3.0 Instruct

8B, 70B parameters | Baseline models

Llama 3.1 Instruct

8B, 70B parameters | Primary models with enhanced coding

Training Configuration

Language: Python 3

Turn Limit: 3 attempts

Updates: 12k (8B), 8k (70B)

Time: ~20 hours

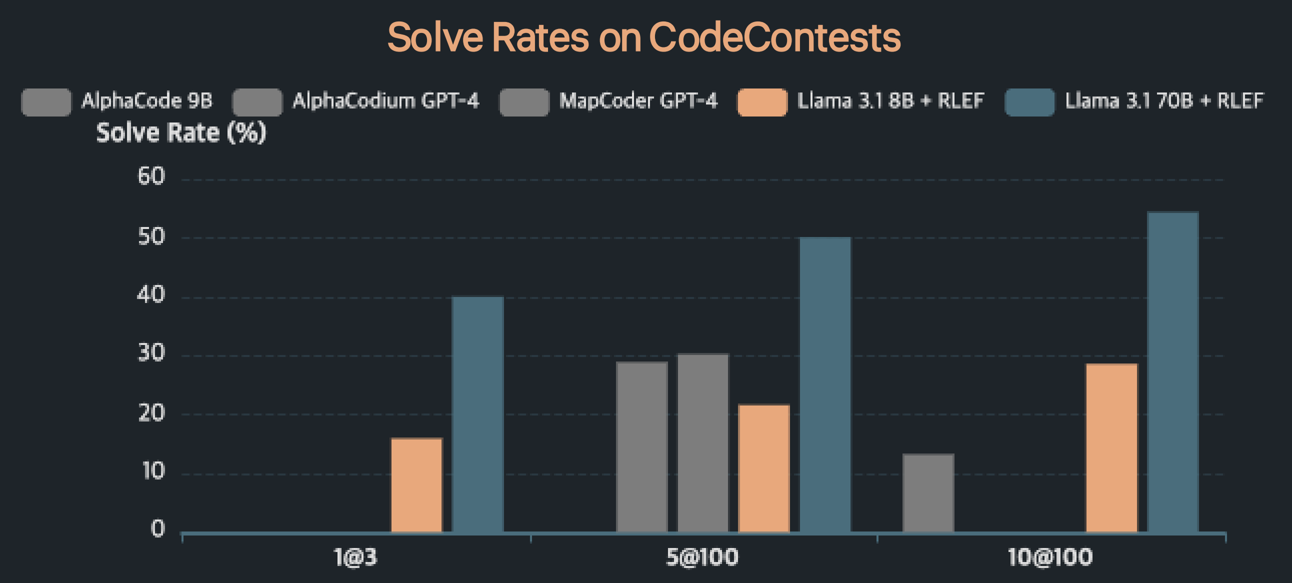
Evaluation Metric

n@k Average Solve Rate

Expectation that any of n solutions from k samples is correct. Enables fair comparison across sample budgets.

04 / MAIN RESULTS

State-of-the-Art Performance



Comparison with prior work across sampling budgets

Key Achievements

70B Model: New SOTA

Beats AlphaCodium+GPT-4 (38.0 vs 29) with 1 vs 100 samples!

8B Model: Competitive

Matches AlphaCode 9B (16.0 vs 13.3) with 3 vs 1,000 samples!

10x Sample Reduction

Achieves SOTA with an order of magnitude fewer samples

Notable Comparisons

- **vs GPT-4:** RLEF 70B (37.5) with 3 samples vs GPT-4 Agent (29) with 100 sample s
- **vs AlphaCode 2:** RLEF 70B (37.5) vs estimated AlphaCode 2 (34.2) on valid set
- **100 samples:** RLEF 70B reaches 54.5 (valid) and 54.5 (test) — beats all prior wor k

+94%
8B Valid
8.9 → 17.2

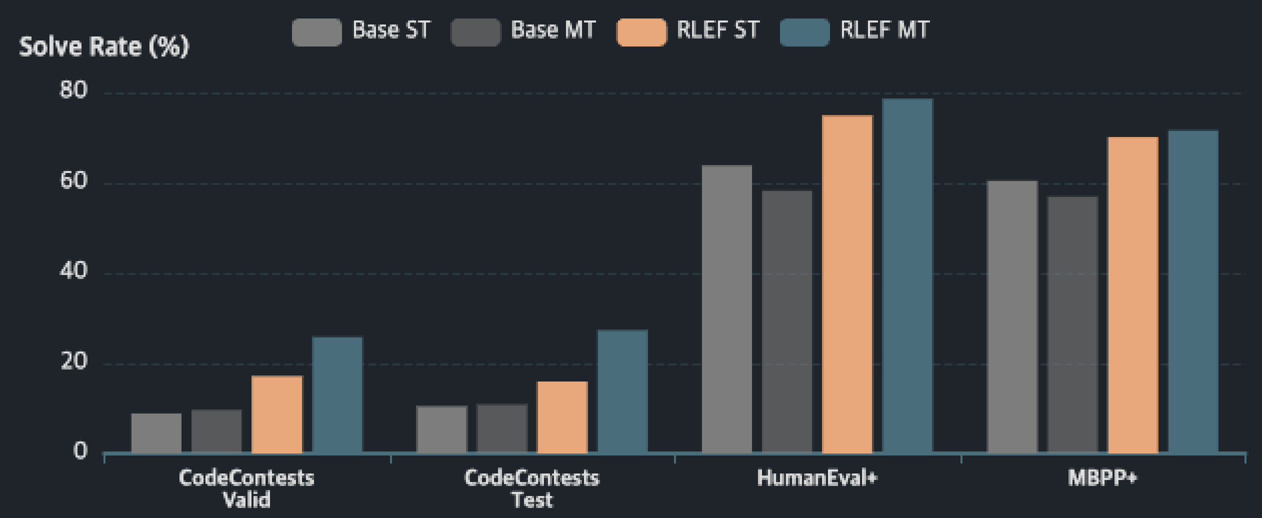
+45%
8B Test
10.5 → 16.0

+45%
70B Valid
25.9 → 37.5

+46%
70B Test
27.5 → 40.1

05 / INFERENCE-TIME BEHAVIOR

Single vs Multi-Turn Analysis



ST = Single-Turn, MT = Multi-Turn

Key Findings

Base Models: Rarely benefit from multi-turn feedback; sometimes perf orm worse

RLEF Models: Effectively leverage feedback to achieve larger gains

GPT-4o: Also shows stronger performance with independent sampling

Generalization

- **HumanEval+:** Improvements carry over with different feedback forma tting
- **MBPP+:** Notable improvements despite simpler programming questio ns

8B CodeContests

Base ST: 11.8% | MT: 9.7%

RLEF ST: 16.0% | MT: 27.4%

Gain: +17.7% from multi-turn

70B CodeContests

Base ST: 25.3% | MT: 10.5%

RLEF ST: 40.1% | MT: 27.4%

Gain: +17.1% from multi-turn

Cross-Domain

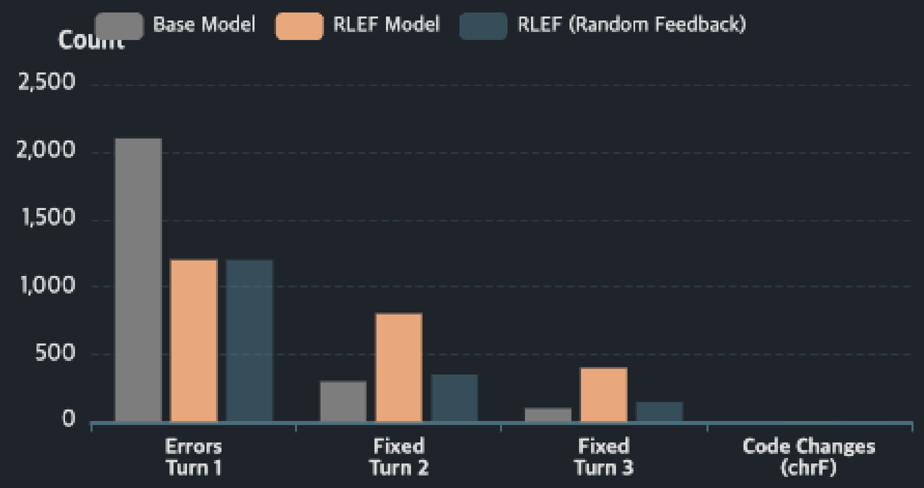
HumanEval+: 80.7% (+3.8%)

MBPP+: 71.7% (+3.2%)

Generalizes to new domains

Error Recovery & Code Changes

Error Analysis Over 20 Rollouts



8B model results (70B similar)

Random Feedback Ablation

With random feedback, self-repairs are severely impaired, proving RLEF models meaningfully leverage true execution feedback.

17.2
True Feedback

12.2
Random Feedback

Key Observations

1. Fewer Initial Errors

RLEF-trained models produce fewer wrong outputs in first response but are more prone to timeout (efficiency focus)

2. Better Error Recovery

Significantly improved recovery from all error categories (output, exception, timeout, OOM) in subsequent responses

3. Larger Code Edits

Higher chrF scores indicate more substantial changes. Base models repeat same code despite feedback

Pass@1 vs Pass@10 Analysis

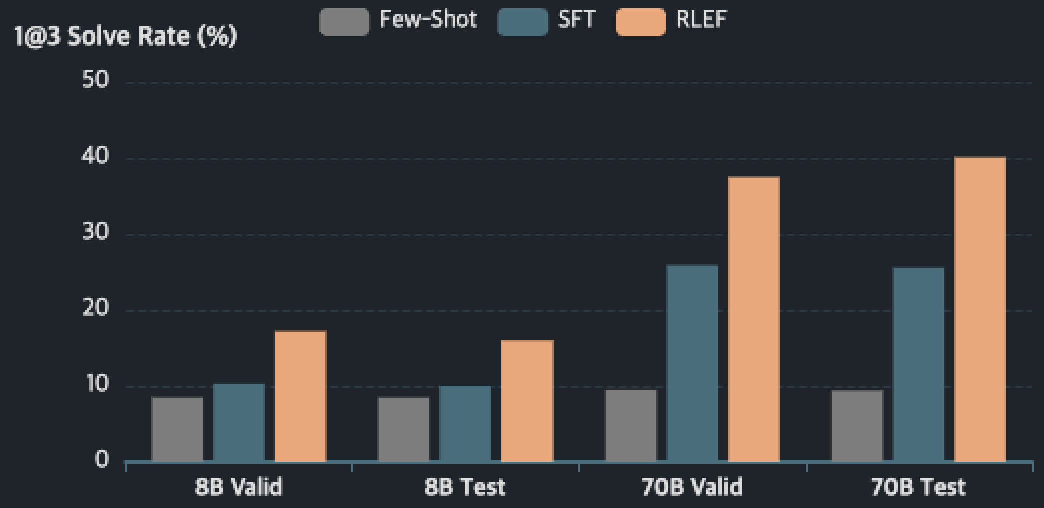
Random feedback drops pass@1 significantly (less targeted repair) but affects pass@10 less (can still sample diverse solutions)

→ **Pass@1:** Measures precision of arriving at correct solution

→ **Pass@10:** Measures recall (whether any solution passes)

Learning Iterative Code Synthesis

Methods for Iterative Capabilities



1@3 solve rates on CodeContests (Instruct models)

Few-Shot Prompting

Finding: Detrimental to Instruct models. Base models also achieve lower perform ance than zero-shot.

Supervised Fine-Tuning (SFT)

Filter 313k successful trajectories from Llama 3.1 70B → fine-tune Base & Instruct models.

10.3	17.2
8B SFT Valid	8B RLEF Valid

Result: Moderate improvements on validation only. RLEF significantly outperform s SFT.

Single-Turn vs Multi-Turn Training

Compare traditional single-generation vs iterative setup with same training loop.

8B Model

Single-turn hurts performance on test set

70B Model

Benefits from single-turn training; shows transfer to multi-turn inference

Positioning RLEF in Code Generation

Agentic Frameworks

AlphaCodium, MapCoder, Reflexion, LDB: Rich manual scaffolding, chaining seve ral LLM calls with code execution.

Issues:

- **Inference cost:** Dozens of LLM calls per solution
- **Independent sampling competitive** (Kapoor et al., 2024)
- **Large models required** for effective feedback

Prior RL Work

Le et al. (2022), Xu et al. (2024): Train code LLMs with scalar rewards from unit te sts.

- **CodeRL:** Policy gradients + next-token loss on rewards
- **DeepSeek-Coder:** Binary reward from unit tests

Limitation: No textual feedback during generation

RLEF Key Distinctions

1. Textual Execution Feedback

Not just scalar rewards — provides rich textual feedback for code synthesis and repair in a si ngle model

2. End-to-End RL Training

Optimizes for leveraging feedback through multi-turn conversation, not just single generatio n

3. Sample Efficiency Focus

Shifts from large-sample inference to high accuracy with low sample budgets

Concurrent Work: SCoRe

Kumar et al. (2024): Two-stage RL for self-correction; outputs two successive sol utions.

Key Difference:

SCoRe doesn't leverage execution feedback at inference time, limiting its ability to adapt to new environments.

Impact, Limitations & Future

Key Contributions

1. RLEF Method

End-to-end RL for LLMs to leverage execution feedback, grounding future generations in envir onment feedback

2. SOTA Performance

Substantial improvements on CodeContests; reduces sample budget by 10x

3. Generalization

Improved multi-turn performance carries over to HumanEval+ and MBPP+

Performance Breakthrough

54.5%
70B 10@100

40.1%
70B 1@3

Limitations

1. Single Solution Focus

Limited to improving single solution; doesn't handle task decomposition

2. Requires Test Cases

Iteration requires test cases, which may not be readily available

3. Domain Specificity

Training on competitive programming may limit broader applicability

Broader Impact

Positive

Amplifies utility for software development and quality control

Consideration

Requires quality control and guard-railing to promote safety

Safety

Execution confined to local sandboxes; follows AI agent governance frameworks

Future Directions

Combine with automatic unit test generation; extend to larger tasks requiring decomposition

KEY TAKEAWAYS

RLEF Enables Effective Feedback Grounding

State-of-the-Art

Achieves new SOTA results with both 8B and 70B models while reducing samples by 10x

Iterative Improvement

Enables LLMs to effectively leverage execution feedback over multiple steps for code synthesis and repair

End-to-End RL

Demonstrates importance of RL training for grounding LLMs in interactive environments

RLEF opens new possibilities for autonomous code generation and repair, shifting focus from large-sample inference to high