

A digital landscape with mountains made of code and glowing paths. The mountains are composed of dark, layered, wavy lines that resemble code or data. Several glowing, golden-yellow paths wind through the landscape, leading to small, glowing, rectangular structures that look like gates or portals. The background is a dark, starry sky with a faint galaxy visible. The overall aesthetic is futuristic and digital.

# Let LLMs wander

## Engineering RL Environments

# Stefano Fiorucci

AI/Software Engineer/Explorer 🧑🏻‍🚀

🧑🏻‍💻 Open-Source AI orchestration - Haystack @deepset

💙 Small Language Models, Fine-Tuning, RL



[anakin87](#)



[anakin87](#)



[theanakin87](#)



[stefano-fiorucci](#)



Andrej Karpathy ✓

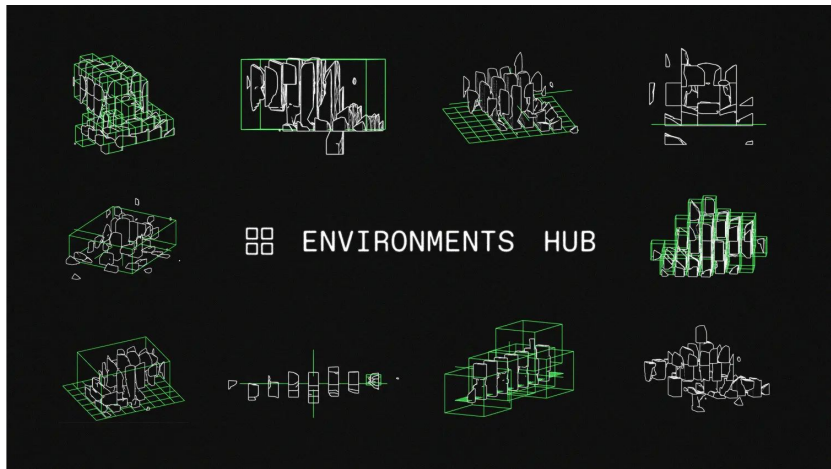
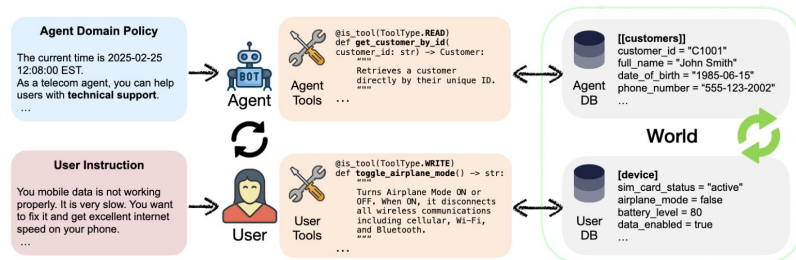
@karpathy



For friends of open source: imo the highest leverage thing you can do is help construct a high diversity of RL environments that help elicit LLM cognitive strategies. To build a gym of sorts. This is a highly parallelizable task, which favors a large community of collaborators.

## LLM RL Environments

### $\tau^2$ -Bench: Evaluating Conversational Agents in a Dual-Control Environment



### TextArena/ TextArena



A Collection of Competitive Text-Based Games for Language Model Evaluation and Reinforcement Learning

28

Contributors

43

Used by

356

Stars

84

Forks





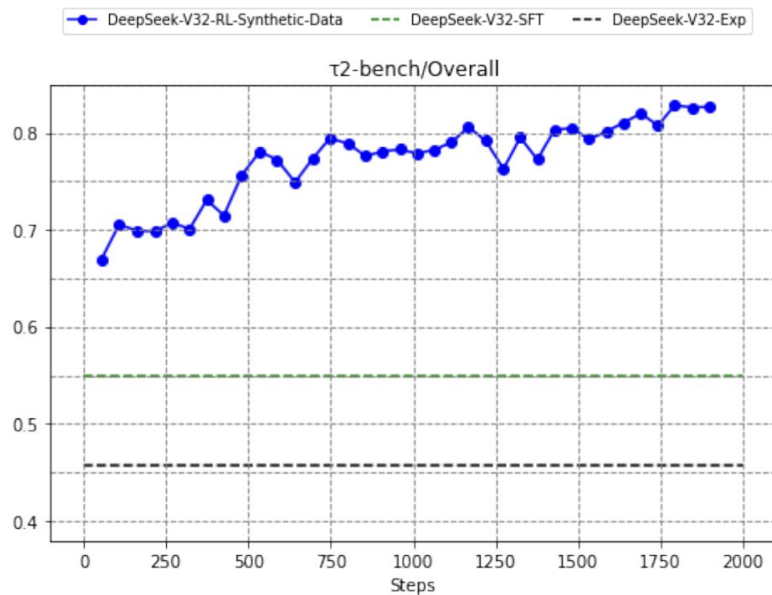
# LLM RL environments at scale

## DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models

tool-use scenarios. First, we implement a cold-start phase utilizing the DeepSeek-V3 ([deepseekv3](#)) methodology to unify reasoning and tool-use within single trajectories. Subsequently, we advance to large-scale agentic task synthesis, where we generate over 1,800 distinct environments and 85,000 complex prompts. This extensive synthesized data drives the RL process, significantly enhancing the model's generalization and instruction-following capability in the agent context.

### M2.1: Multilingual and Multi-Task Coding with Strong Generalization


Ultimately, we built a multi-language training system covering over ten languages including JS, TS, HTML, CSS, Python, Java, Go, C++, Kotlin, C, and Rust. We obtained over 100,000 environments usable for training and evaluation from real GitHub repositories, with each environment containing complete Issues, code, and test cases. To support




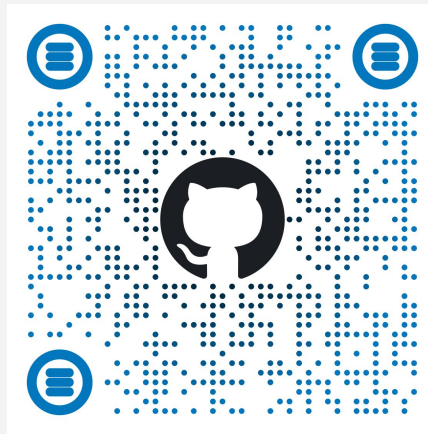
# Agenda

Materials 

 Reinforcement Learning and LLMs

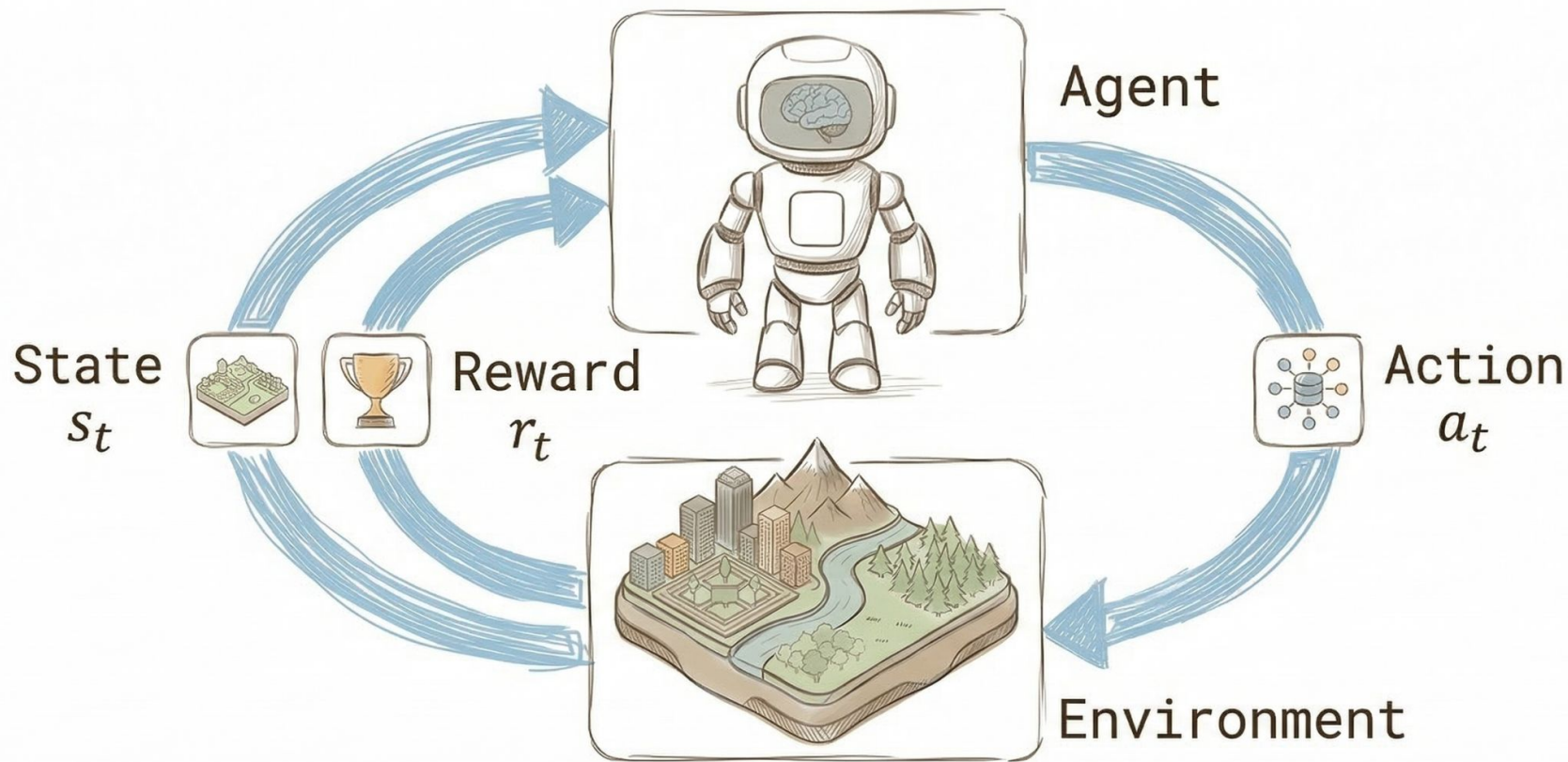
 Verifiers:  
Environments as software artifacts

  RL Training a Tic Tac Toe master

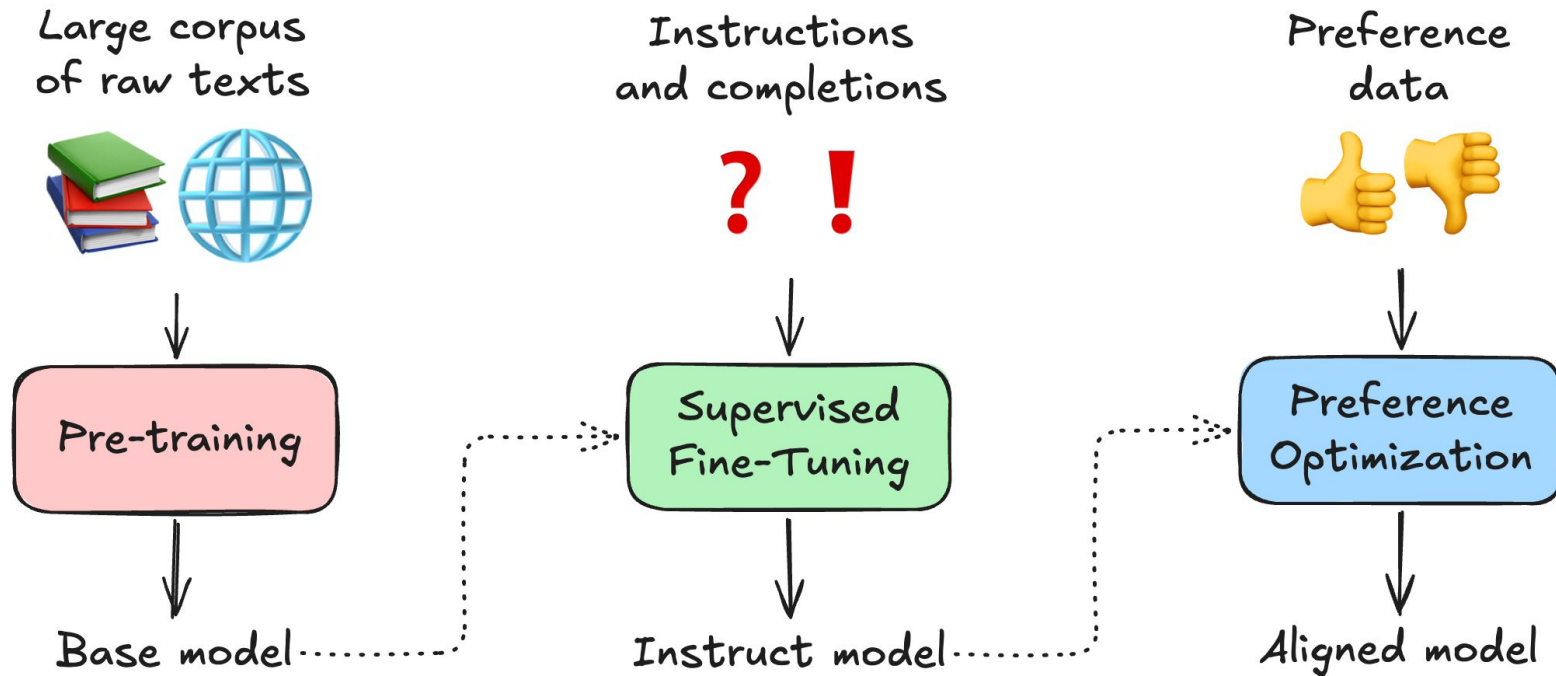


# Reinforcement Learning and LLMs

# RL: the Agent and the Environment



# Classic LLM training





# Supervised Fine-Tuning (SFT) - Data



**Prompt/Instruction/Question/User message**

Explain the basic principles of sound waves, including their properties and how they travel through different mediums.

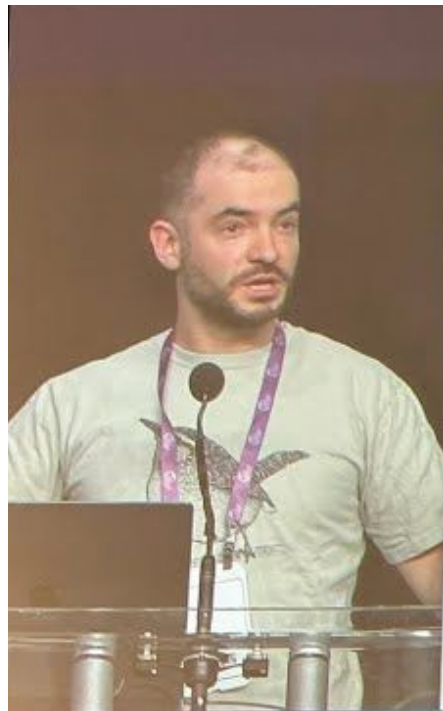


**Response/Completion/Answer/Assistant message**

Sound waves are a type of mechanical wave that propagate through a medium by causing the particles in the medium to vibrate. These vibrations transfer energy from one particle to another, allowing the sound wave to travel through the medium. The basic principles of sound waves include their properties and how they travel through different mediums...



Example dataset: mlabonne/FineTome-100k



## Pre-training as we know it will end

Compute is growing:

- Better hardware
- Better algorithms
- Larger clusters

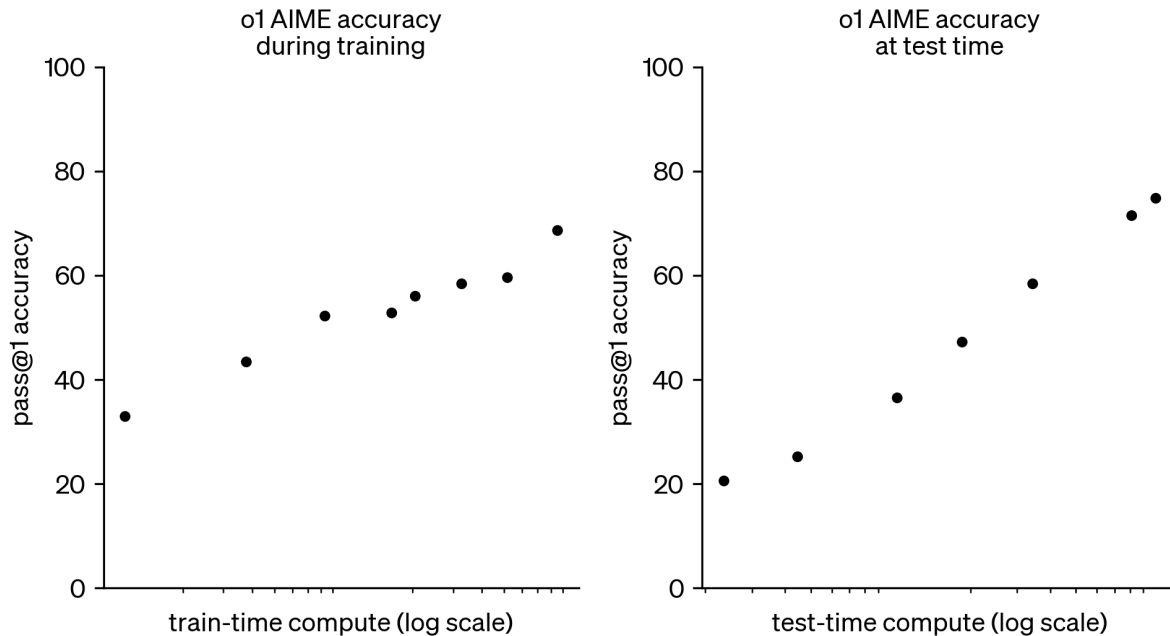
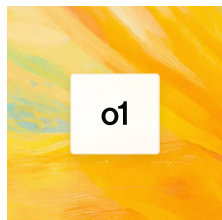
Data is not growing:

- We have but one internet
- **The fossil fuel of AI**

Internet. We have, but one Internet. You could even say you can even go as far as to say. That data is the fossil fuel of AI. It was like, created somehow. And now we use it.

*What comes next?*

# o1 improves with more compute



*But how?*

# DeepSeek-R1 contains a plausible answer

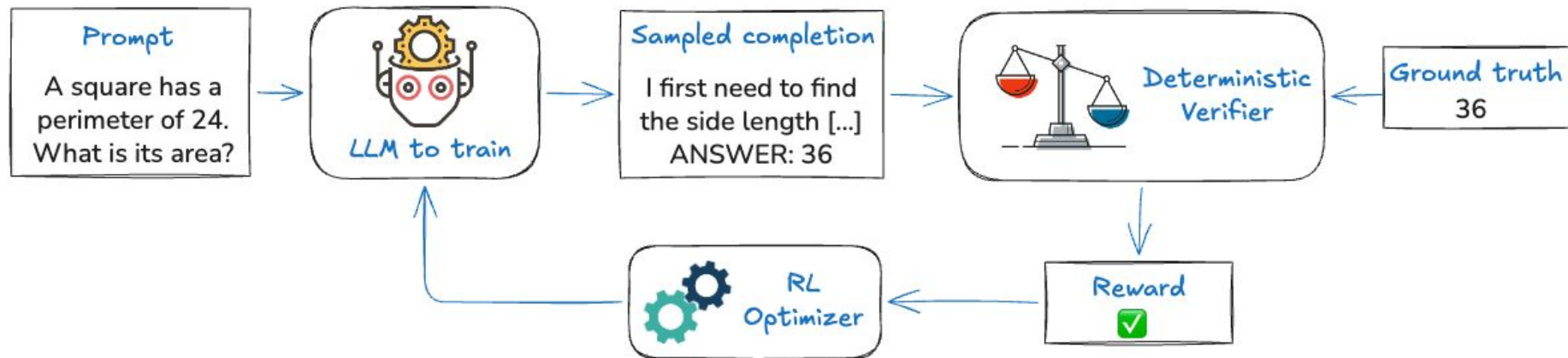


---

## **DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning**

- Reasoning
- Reinforcement Learning with Verifiable Rewards (RLVR)
- Group Relative Policy Optimization (GRPO)

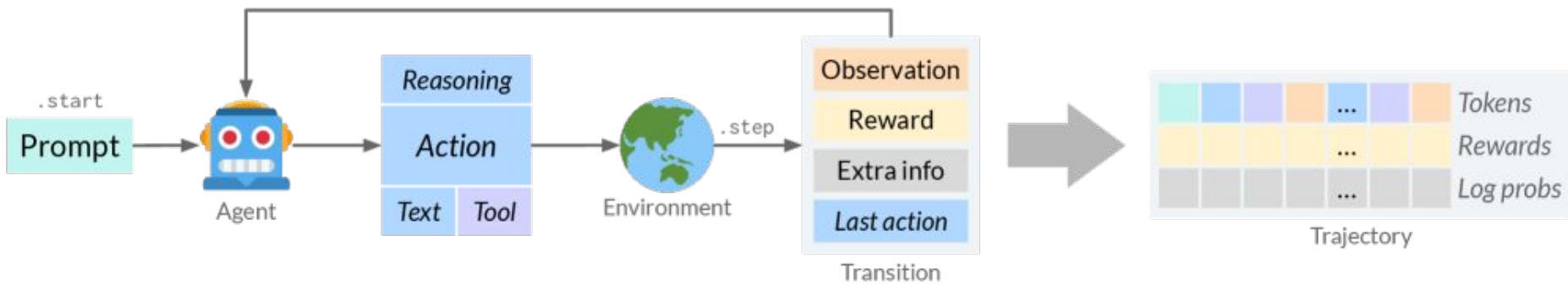
# Reinforcement Learning with Verifiable Rewards (RLVR)





# RL concepts > LLMs

From “CWM: An Open-Weights LLM for Research on Code Generation with World Models”



Agent > Language Model

Action > LM Response

Environment > Data, harnesses, scoring rules...



Andrej Karpathy ✓

@karpathy



In era of pretraining, what mattered was internet text. You'd primarily want a large, diverse, high quality collection of internet documents to learn from.

In era of supervised finetuning, it was conversations. Contract workers are hired to create answers for questions, a bit like what you'd see on Stack Overflow / Quora, or etc., but geared towards LLM use cases.

Neither of the two above are going away (imo), but in this era of reinforcement learning, it is now environments. Unlike the above, they give the LLM an opportunity to actually interact - take actions, see outcomes, etc. This means you can hope to do a lot better than statistical expert imitation. And they can be used both for model training and evaluation. But just like before, the core problem now is needing a large, diverse, high quality set of environments, as exercises for the LLM to practice against.

## SFT vs RL



Environments as software artifacts

# Verifiers by PRIME *Intellect*

Open-source toolkit to build RL environments for LLMs

- Eval + train
- Environments as Python packages
- Prebuilt environment types
- Parsing + reward handling
- Works with OpenAI-compatible APIs
- Async + parallel execution
- Training support: PRIME-RL, Tinker, SkyRL, ...

# Single Turn Environment

```
from datasets import load_dataset

import verifiers as vf

def load_environment(
    dataset_name: str = "PrimeIntellect/Reverse-Text-RL",
    dataset_split: str = "train",
    system_prompt: str | None = ("Reverse the text character-by-character. "
                                "Put your answer in <reversed_text> tags."),
) → vf.Environment:
    train_dataset = load_dataset(dataset_name, split=dataset_split).map(
        lambda x: {
            "question": x["prompt"],
            "answer": x["prompt"][::-1],
            "info": {},
            "task": "reverse-text",
        }
    )
    train_dataset = train_dataset.remove_columns(["prompt"])

    parser = vf.XMLParser(["reversed_text"], answer_field="reversed_text")

    def lcs_reward_func(completion, answer, **kwargs) → float:
        """
        LCS ratio of the reversed prompt and the parsed completion.
        """
```

```
def lcs_ratio(x: str, y: str) → float:
    """
    Return the longest common subsequence ratio of x and y.
    """
    from difflib import SequenceMatcher

    return SequenceMatcher(None, x, y).ratio()

response = parser.parse_answer(completion) or ""
return lcs_ratio(response, answer)

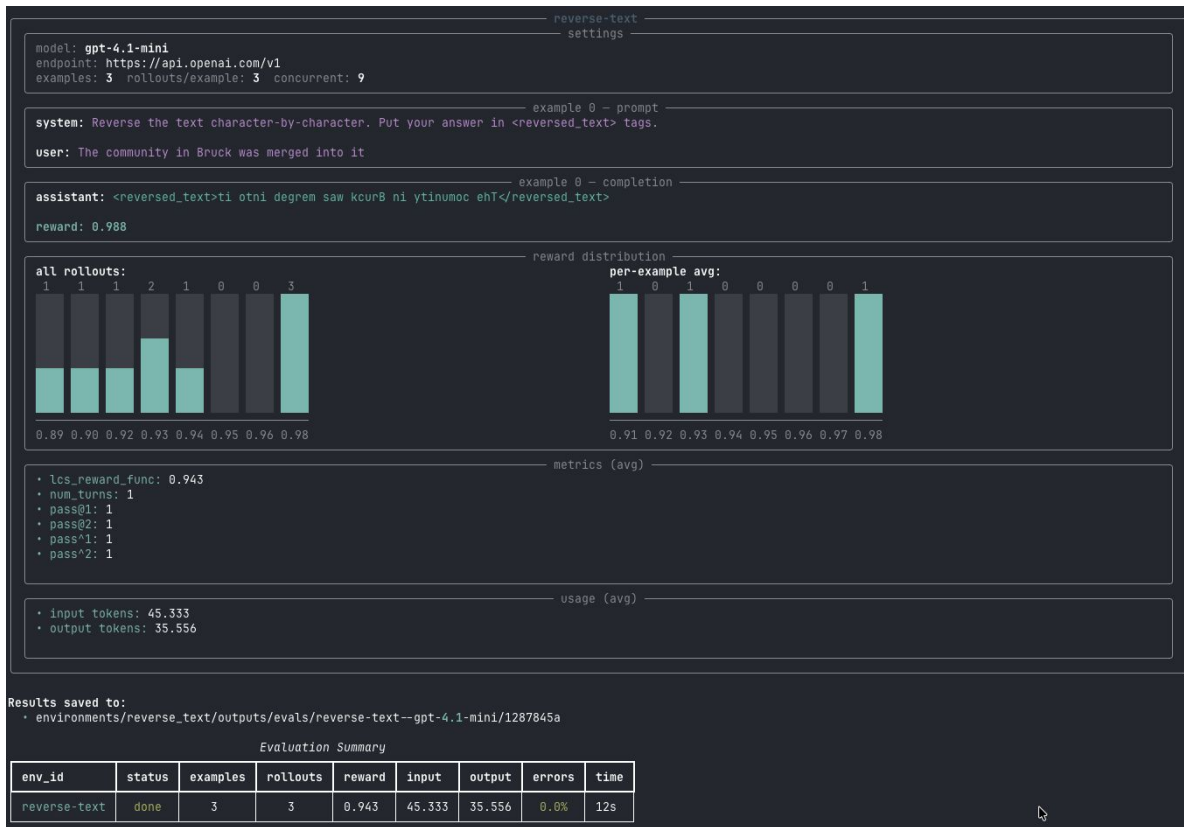
rubric = vf.Rubric(
    funcs=[
        lcs_reward_func,
    ],
    weights=[1.0],
)

vf_env = vf.SingleTurnEnv(
    dataset=train_dataset,
    system_prompt=system_prompt,
    parser=parser,
    rubric=rubric,
)
return vf_env
```



# Single Turn Environment: evaluation

```
prime eval run reverse-text -m gpt-4.1-mini -n 3 -r 3
```



# Multi Turn Environment

```
import verifiers as vf
from verifiers.types import Messages, State
from verifiers.rubrics.math_rubric import MathRubric
from verifiers.utils.data_utils import load_example_dataset

SIMPLE_PROMPT = """
Respond in the following format, using careful step-by-step reasoning.

<reasoning>
...
</reasoning>
<answer>
...
</answer>
"""

class DoubleCheckEnv(vf.MultiTurnEnv):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    @vf.stop
    async def double_checked(self, state: State) → bool:
        return len(state["trajectory"]) == 2

    async def env_response(
        self, messages: Messages, state: State, **kwargs
    ) → Messages:
        """Generate a response from the environment."""
        return [{"role": "user", "content": "Are you sure?"}]
```

```
def load_environment(
    dataset_name: str = "math",
    dataset_split: str = "train",
    num_train_examples: int = -1,
):
    dataset = load_example_dataset(
        dataset_name,
        dataset_split,
        n=num_train_examples,
    )

    rubric = MathRubric()

    vf_env = DoubleCheckEnv(
        dataset=dataset,
        system_prompt=SIMPLE_PROMPT,
        few_shot=[],
        parser=rubric.parser,
        rubric=rubric,
    )


    return vf_env
```

# Tool Environment

```
async def calculate(expression: str) → str:
    """Evaluate a mathematical expression.

    Args:
        expression: A mathematical expression to evaluate (e.g. "2 + 2 * 3")


    Returns:
        The result of the evaluation.
    """
    try:
        result = eval(expression)
        return str(result)
    except Exception as e:
        return f"Error: {e}"
```



```
async def lookup(term: str) → str:
    """Look up a term in the knowledge base.

    Args:
        term: The term to search for.

    Returns:
        Information about the term.
    """
    # your lookup logic here
```



```
vf_env = vf.ToolEnv(
    dataset=dataset,
    tools=[calculate, lookup],
    rubric=rubric,
    max_turns=10,
)
```

See also

[Wiki Search env](#)

# More Environments

- MCP Tool Env
- Stateful Tool Env
- Sandbox Env
- Recursive Language Models Env
  
- Integrations: Text Arena, Reasoning Gym, Open Env, ....

# Environments Hub

## Environments Hub

A community hub for discovering and sharing environments, both for RL training and downstream evaluation

[Learn More](#)

[Create Environment](#)

[Explore](#)

[My Stars](#)

[My Environments](#)

Search by name, author, description, tags...

By sections



Coding

Data & ML

Games

Math & Reasoning

Multimodal

Science & Medicine

Search & Tool Use

## Featured

9

[Show all](#)



hud

29 ★

### hud-text-2048

Text-based 2048 game for training agents to reach target tiles through strategic...

game

text

+2

Updated 7 months ago

v0.1.0



arcee-ai

28 ★

### ifeval

IFEval single-turn chat environment using RLVR-IFEval with JSON constraint reward...

rlvr

chat

+6

Updated 5 months ago

v0.1.1



aarush

15 ★

### vf-openbench

Environment for single-turn tasks in OpenBench

Updated 7 months ago

v0.1.0



bhogan94

13 ★

### q-programming-language

Your environment description here

eval

train

+1

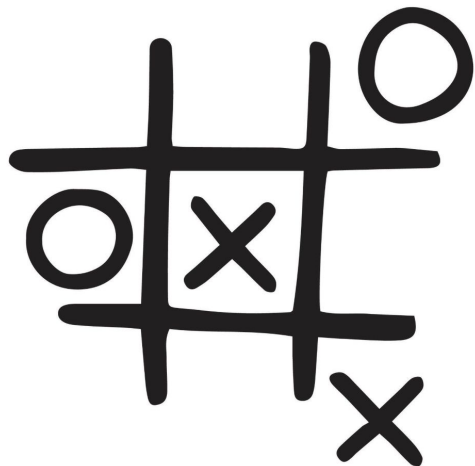
Updated 7 months ago

v0.1.0



# XO 🎮 RL Training a Tic Tac Toe master

# Why Tic Tac Toe?



Simple game with deterministic solution

Still, Language Models struggle

Multi-turn interactions

# Tic Tac Toe env: initial implementation

```
from typing import Any

import verifiers as vf
from datasets import Dataset
from verifiers.types import Messages, State

# OMITTED GAME HELPERS...

SYSTEM_PROMPT = f"""You are playing a game of Tic-Tac-Toe as X.
[...]
Your final answer must include the position you choose
inside <move>...</move> tags.
"""

def user_feedback(
    status: str, board: list[str | None],
    final: bool = False, as_messages: bool = True
) → str | list[dict[str, str]]:
    """Format consistent feedback to the model."""
    ...

class TicTacToeEnv(vf.MultiTurnEnv):
    async def setup_state(self, state: State) → State:
        state["board"] = [None] * 9
        # None=in progress, "X"/"O"=winner, "draw"=draw
        state["winner"] = None
        return state

    async def env_response(
        self, messages: Messages, state: State, **kwargs
    ) → Messages:
        """Process model's move and opponent's response."""
        board = state["board"]
        move = self.parser.parse_answer(messages) or ""
        free = get_free_positions(board)
```

```
# Validate move: must be one of the free positions
if move not in [str(p) for p in free]:
    state["winner"] = "0"
    final = user_feedback(
        "Game over! Invalid move. You lose.", board, final=True
    )
    state["final_env_response"] = final
    return final

pos = int(move)

# Apply model's move (X) and check for win
board[pos] = "X"
if check_win(board=board, player="X"):
    state["winner"] = "X"
    final = user_feedback("Game over! You won!", board, final=True)
    state["final_env_response"] = final
    return final
if not get_free_positions(board=board):
    state["winner"] = "draw"
    final = user_feedback("Game over! It's a draw.", board, final=True)
    state["final_env_response"] = final
    return final

# Opponent's move (O) - always random
opp_pos = get_random_move(board=board)
board[opp_pos] = "O"
opp_status = f"Opponent (O) played at position {opp_pos}."

if check_win(board=board, player="O"):
    state["winner"] = "O"
    final = user_feedback(
        f"Game over! {opp_status} You lose.", board, final=True
    )
    state["final_env_response"] = final
    return final
if not get_free_positions(board=board):
    state["winner"] = "draw"
    final = user_feedback(
        f"Game over! {opp_status} It's a draw.", board, final=True
    )
    state["final_env_response"] = final
    return final

# Game continues
return user_feedback(opp_status, board)
```

```
def win_reward_func(state: State, **kwargs: Any) → float:
    winner = state.get("winner")
    return 1.0 if winner == "X" else 0.5 if winner == "draw" else 0.0

def load_environment(
    num_examples: int = 100,
    **kwargs,
) → vf.Environment:
    # Create dataset - model always starts
    def make_dataset():
        rows = []
        for _ in range(num_examples):
            rows.append(
                {
                    "question": user_feedback(
                        "Game started. You are X.",
                        [None] * 9,
                        as_messages=False
                    ),
                }
            )
        return Dataset.from_list(rows)

    parser = vf.XMLParser(fields=["move"], answer_field="move")
    rubric = vf.Rubric(parser=parser, funcs=[win_reward_func])
    rubric.add_reward_func(parser.get_format_reward_func(), weight=0.2)

    return TicTacToeEnv(
        dataset=make_dataset(),
        system_prompt=SYSTEM_PROMPT,
        parser=parser,
        rubric=rubric,
        max_turns=10,
        **kwargs,
    )
```

# Tic Tac Toe env: improvements

- Varying who starts the game
  - Making the opponent stronger
  - Making models think
  - Handling invalid moves
- 
- Reducing noise in group-based RL with seeds
  - Reducing noise across batches with stratified sampling

[Full code](#)

# Evaluate existing models on Tic Tac Toe

```
prime env install anakin87/tictactoe && prime eval run tictactoe ...
```

Model vs random opponent	% Wins	% Draws	% Losses	% Follows format	% Games w invalid moves
openai/gpt-5-mini	90	9	1	100	0
LiquidAI/LFM2-2.6B	40	11	49	27.8	40

Model vs optimal opponent	% Wins	% Draws	% Losses	% Follows format	% Games w invalid moves
openai/gpt-5-mini	0	76	24	100	0
LiquidAI/LFM2-2.6B	0	11	89	24.7	43



# LiquidAI/LFM2-2.6B -> Tic Tac Toe master

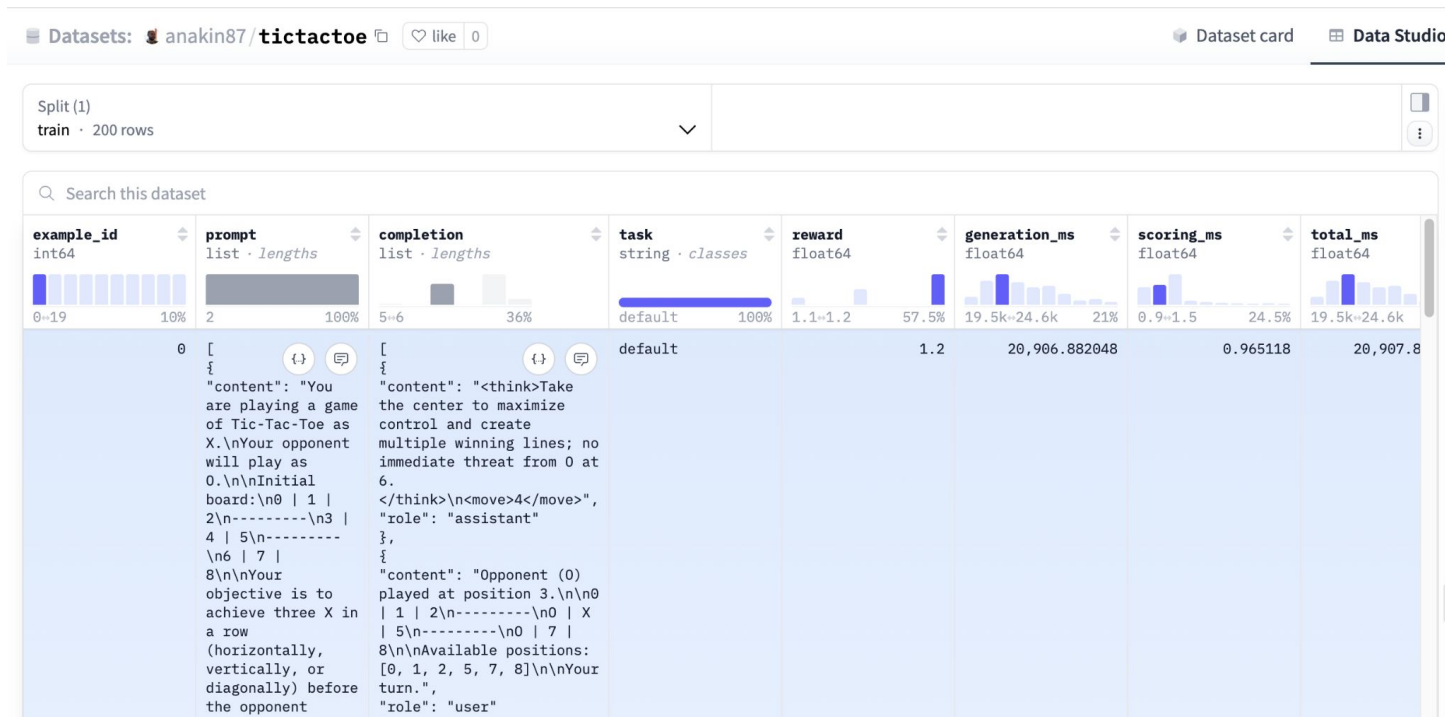


## Training overview

1. Synthetic data generation for Supervised Fine-Tuning
2. Supervised Fine-Tuning
3. (One or more phases of) Group-based Reinforcement Learning

# Synthetic data generation for SFT

```
prime eval run tictactoe -m openai/gpt-5-mini -n 200 -r 1 \
--save-to-hf-hub --hf-hub-dataset-name anakin87/tictactoe
```



# Supervised Fine-Tuning warm-up



prime-rl

```
max_steps = 30
```

```
[ckpt]
```

```
# Checkpoint at the end of training
```

```
[model]
```

```
name = "LiquidAI/LFM2-2.6B"
```

```
[data]
```

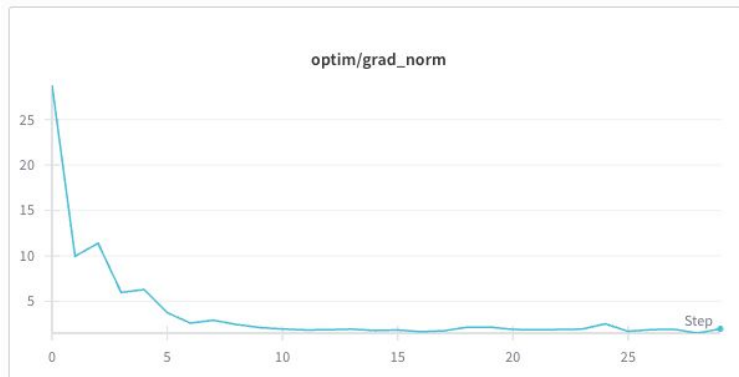
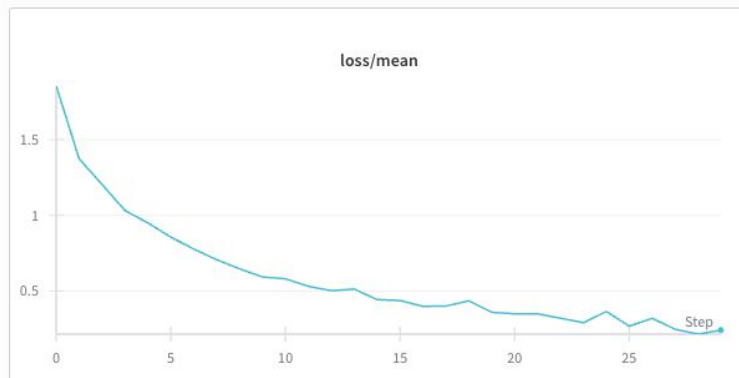
```
name = "anakin87/tictactoe-filtered"
```

```
seq_len = 700
```

```
batch_size = 32
```

```
[optim]
```

```
lr = 1e-5
```

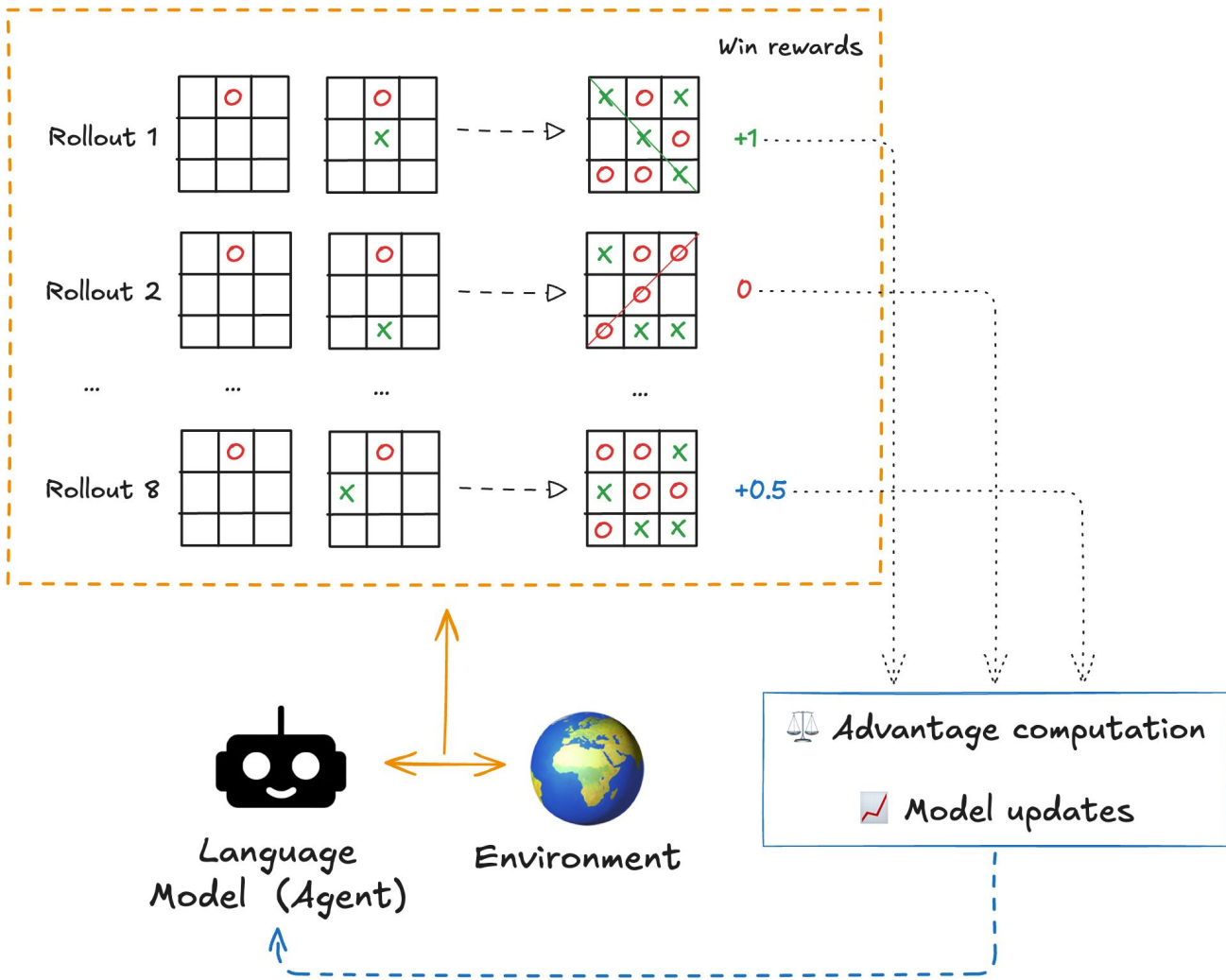


NVIDIA RTX Pro 6000 96GB

# SFT warm up - Evaluation

Model vs random opponent	% Wins	% Draws	% Losses	% Follows format	% Games w invalid moves
LiquidAI/LFM2-2.6B	40	11	49	27.8	40
LFM2-2.6B-ttt-sft	74	13	13	<b>99.8</b>	11

Model vs optimal opponent	% Wins	% Draws	% Losses	% Follows format	% Games w invalid moves
LiquidAI/LFM2-2.6B	0	11	89	24.7	43
LFM2-2.6B-ttt-sft	0	52	48	<b>99</b>	14



## GRPO recap

# RL training: configuration

```
model = "anakin87/LFM2-2.6B-ttt-sft"
```

```
[env]
```

```
id = "anakin87/tictactoe"
```

```
[env.args]
```

```
min_random_move_prob = 0.2
```

```
max_random_move_prob = 0.7
```

```
max_turns = 7
```

```
num_examples = 19200
```

```
num_groups = 32
```

```
[inference]
```

```
gpus = 1
```

```
[inference.args]
```

```
enforce_eager = true
```

```
[trainer]
```

```
gpus = 1
```

```
[trainer.args]
```

```
run_name = "tictactoe"
```

```
use_liger = false
```

```
# Batch configuration
```

```
micro_batch_size = 8
```

```
rollouts_per_example = 8
```

```
batch_size = 256
```

```
max_tokens = 128
```

```
max_seq_len = 768
```

```
max_steps = 600
```

```
learning_rate = 5e-5
```

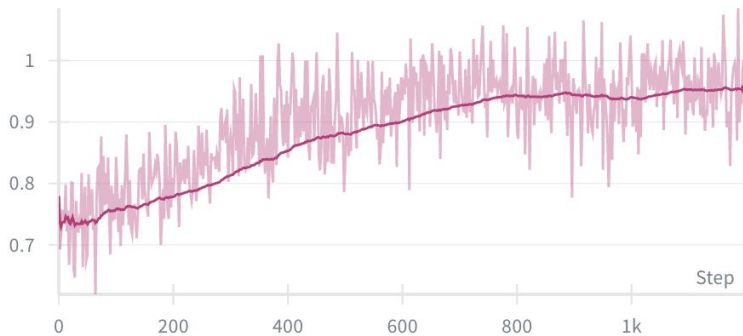


2x NVIDIA RTX Pro  
6000 96GB

# RL training plots

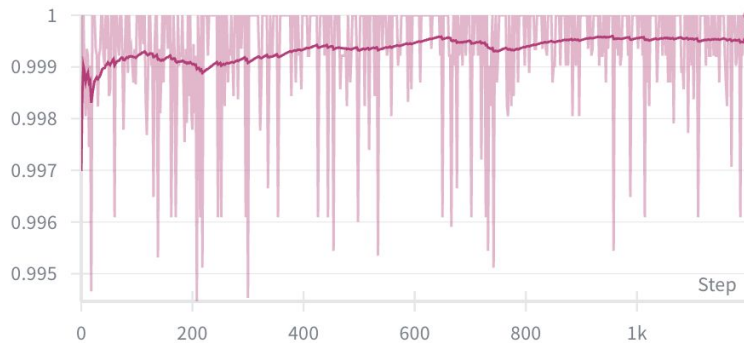
train/reward

— RL-1



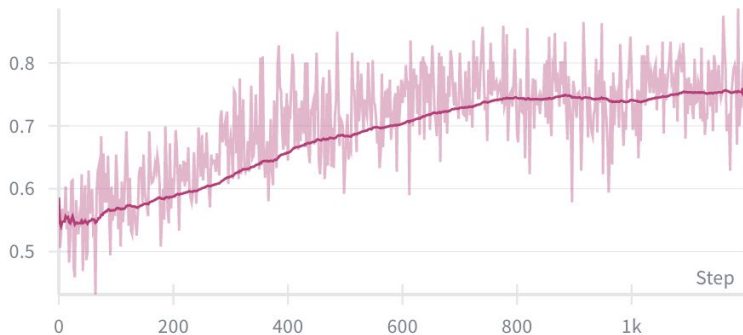
train/reward/format\_reward\_func

— RL-1



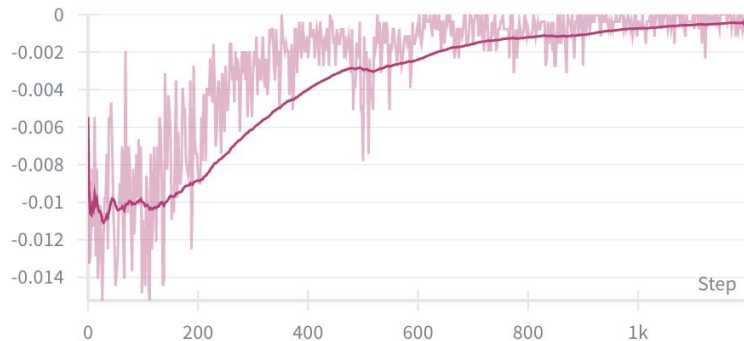
train/reward/win\_reward\_func

— RL-1



train/reward/invalid\_move\_penalty\_func

— RL-1



# RL Training - Evaluation

Model vs random opponent	% Wins	% Draws	% Losses	% Follows format	% Games w invalid moves
LFM2-2.6B-ttt-sft	74	13	13	99.8	11
LFM2-2.6B-ttt-rl	86	12	2	100	1

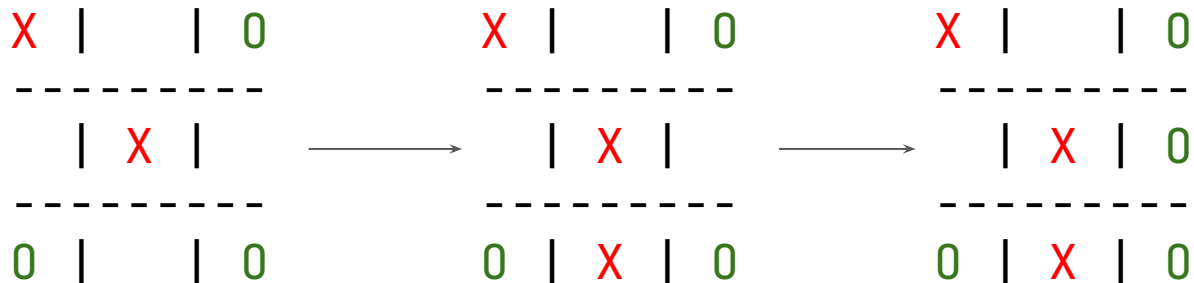
Model vs optimal opponent	% Wins	% Draws	% Losses	% Follows format	% Games w invalid moves
LFM2-2.6B-ttt-sft	0	52	48	99	14
LFM2-2.6B-ttt-rl	0	85	15	100	1



# Possible improvements

Our model (X) still loses games sometimes

Rollouts inspection -> it falls into fork traps



# RL training pt. 2: configuration

```
model = "anakin87/LFM2-2.6B-ttt-rl-merged"
```

```
[env]
```

```
id = "anakin87/tictactoe"
```

```
[env.args]
```

```
min_random_move_prob = 0.0
```

```
max_random_move_prob = 0.25
```

```
max_turns = 7
```

```
num_examples = 24000
```

```
num_groups = 60
```

```
[inference]
```

```
gpus = 1
```

```
[inference.args]
```

```
enforce_eager = true
```

```
[trainer]
```

```
gpus = 1
```

```
[trainer.args]
```

```
run_name = "tictactoe"
```

```
use_liger = false
```

```
# Batch configuration
```

```
run_name = "tictactoe"
```

```
use_liger = false
```

```
micro_batch_size = 24
```

```
rollouts_per_example = 8
```

```
batch_size = 480
```

```
lora_rank = 8
```

```
max_tokens = 64
```

```
max_seq_len = 384
```

```
max_steps = 400
```

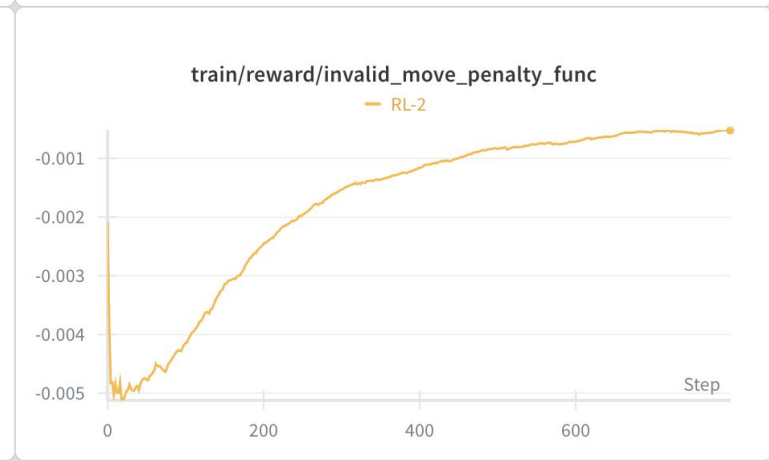
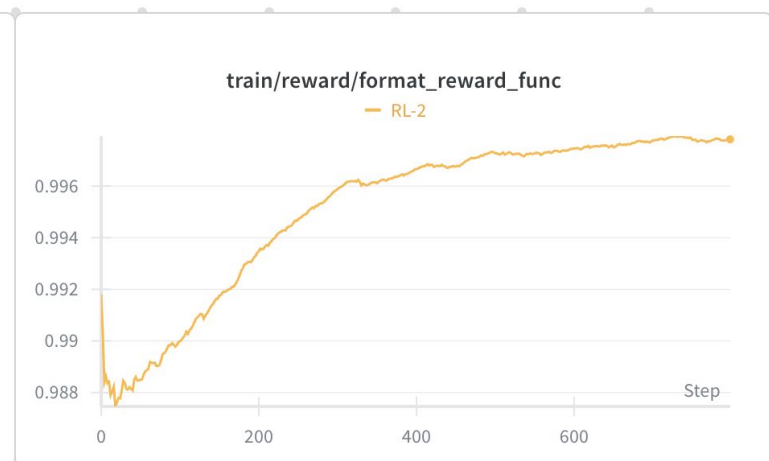
```
learning_rate = 3e-5
```

```
temperature = 1.25
```



2x NVIDIA H200  
141GB

# RL training pt. 2 plots

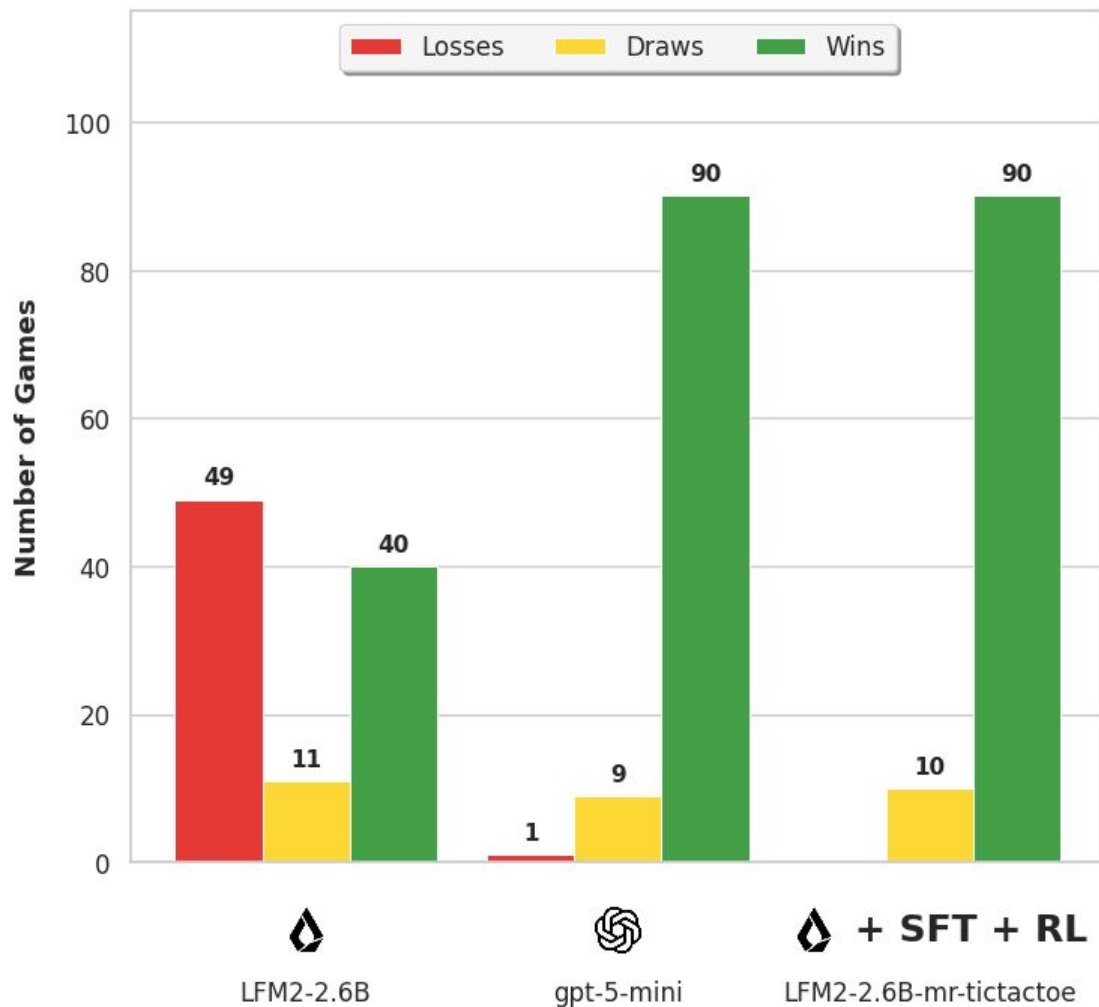


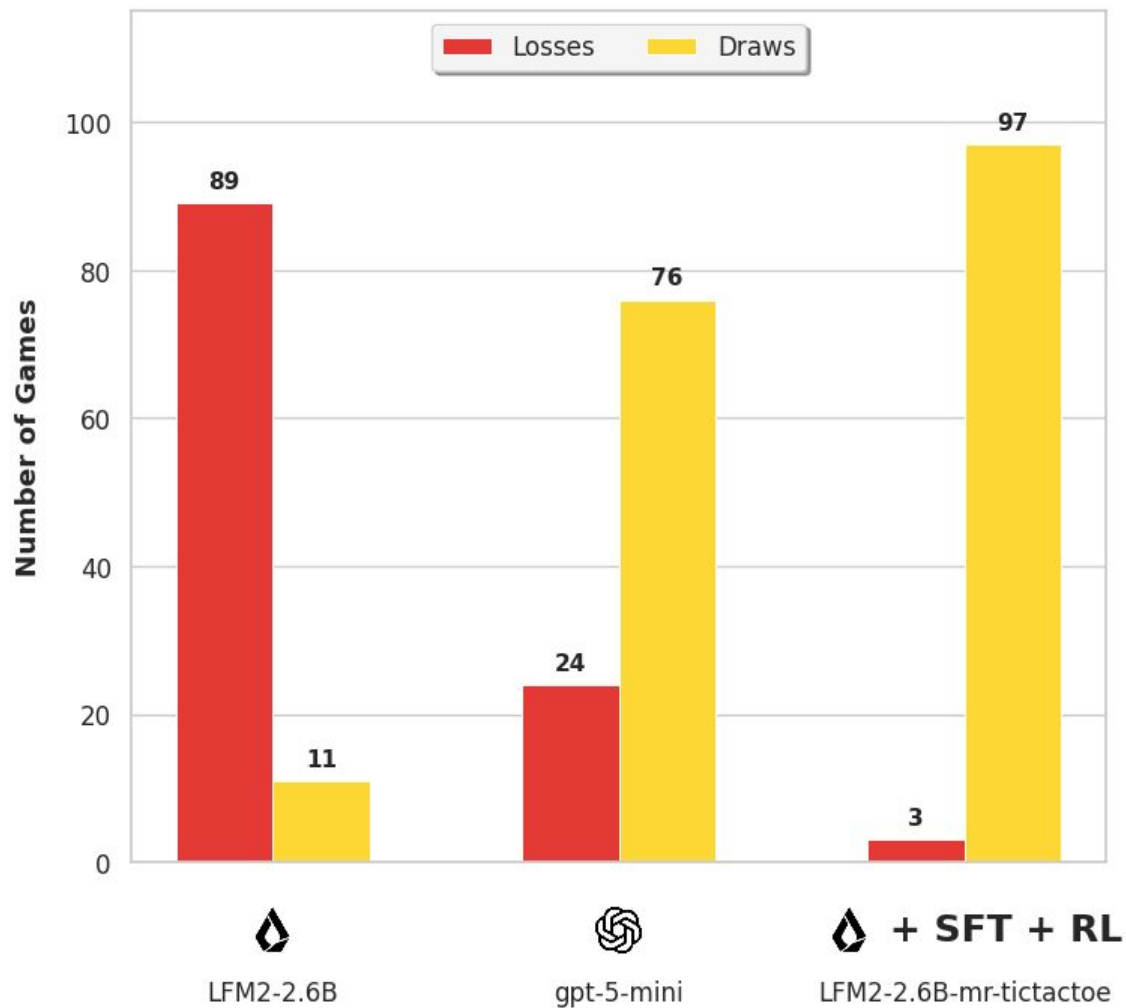
# RL Training pt. 2 - Evaluation

Model vs random opponent	% Wins	% Draws	% Losses	% Follows format	% Games w invalid moves
LFM2-2.6B-ttt-rl	86	12	2	100	1
LFM2-2.6B-mr-tictactoe	90	10	0	100	0

Model vs optimal opponent	% Wins	% Draws	% Losses	% Follows format	% Games w invalid moves
LFM2-2.6B-ttt-rl	0	85	15	100	1
LFM2-2.6B-mr-tictactoe	0	97	3	99.8	0

# Models vs random opponent





Models vs  
optimal  
opponent

# Conclusions

## What we did

- Mapped RL concepts to LLMs
- Explored Verifiers to easily build RL Environments
- Turned a 2.6B model into a Tic Tac Toe master

*Big models are powerful, but small models trained with RL in verifiable environments can beat them on specific tasks, at a fraction of the cost.*

# Lessons from failed runs

- Batch size is key
- Watch for hidden biases in environments
- Model choice matters
  - Consider starting from an instruct model
  - Very small models (<2B) might fail
- Always inspect outputs
- Try/validate the trained model beyond metrics
- Don't micromanage training, go for a walk



# What you can do next?

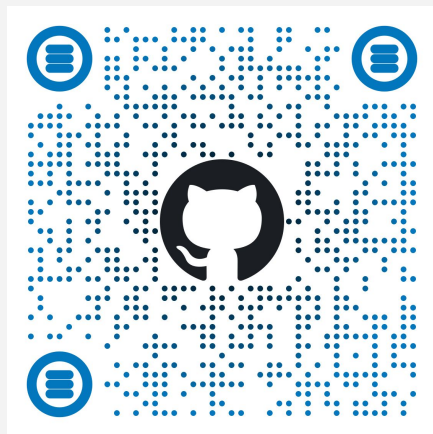
- Take a deep dive  
start with the [LLM RL Enviroments Lil Course](#)
- See what others are building:  
explore the [Enviroments Hub](#)
- Experiment yourself:  
train a small Language Model on 2-3 tools and try to beat  
a large proprietary model in that specific task  
([Wiki Search example](#))



# Let LLMs wander: Engineering RL Environments

*Stefano Fiorucci*

Talk Materials +  
Course



[anakin87](#)



[theanakin87](#)



[anakin87](#)



[stefano-fiorucci](#)