



Train the agent, not just the model

Sergio Paniego | @sergiopaniego 🤖 Hugging Face

October 2026



THE VOCABULARY

An agent is a model plus a harness



Claude Code



Codex



OpenCode



Pi



Mini-SWE-Agent



runs the loop

it calls the model again and again, and decides when to stop



owns the tools

it decides which tools exist, and it is the one that runs every call



writes the context

system prompt, history, compaction: everything the model reads



reads the reply

in its own format, and rejects it if it does not fit

Same weights, different harness: LFM2.5–2.6B solves **62%** of the tasks in Mini-SWE-Agent and **33%** in Claude Code.



THE MAP

How an agent gets trained in 2026

Post-training is where a pretrained model becomes an agent.

WHAT THE MODEL LEARNS FROM

WHERE IT ACTS WHILE IT LEARNS

imitate

SFT on agent traces

a demonstrator: the agent's past sessions

SFTTrainer

compress

Distillation

a teacher: its probabilities on every token

DistillationTrainer

reward

RL with verifiable rewards

a check you can run: did the tests pass?

GRPOTrainer

act

RL in environments

a sandbox with tools, in a loop the trainer runs

GRPOTrainer + OpenEnv

harness

RL through the harness

the real agent, running its own loop

AsyncGRPOTrainer + OpenEnv

The first three change the training signal, the last two move training into a loop where the model acts. **Mix and match: combine them, change the order, and skip what your data cannot support.**



THE STACK

Two open projects for every method



TRAINS THE MODEL

Hugging Face's post-training library. One trainer per method, with the same shape for all of them.

SFTTrainer

DistillationTrainer

GRPOTrainer

AsyncGRPOTrainer

hf.co/docs/trl



OpenEnv

SERVES THE ENVIRONMENTS

One interface for RL environments, reset and step behind an HTTP server, each one a Space you can pull from the Hub.

coding_env

Harbor tasks

browsers

games

simulators

and real coding agents inside them:



hf.co/docs/openenv

 Both run on PyTorch.



IMITATE

SFT on agent traces

PRE-RECORDED TRACE · ONE WHOLE AGENT SESSION, SAVED AS DATA

static dataset · nothing runs while training

1 · USER

fix the failing test
in utils.py

2 · AGENT

read("utils.py")

3 · TOOL

def parse(s): ...

4 · AGENT

edit("utils.py", ...)

5 · TOOL

pytest: 1 failed

6 · AGENT

edit("utils.py", ...)

7 · TOOL

pytest: 3 passed

8 · AGENT

Fixed the off-by-one
in parse()

☐ in the loss ☐ context only

Messages, tool calls and tool results, in the chat format the model already reads.

With **assistant_only_loss** the model learns the agent's turns and only reads the rest.

Claude Code, Codex and Pi traces can be shared on the Hub, with a trace viewer.

The model only gets as good as the traces it copies, and good traces rarely show how to recover from a mistake.



IMITATE

SFT in *TRL*

The traces become chat messages, with their tool calls, and **SFTTrainer** trains on them.

```
sft.py PYTHON

from huggingface_hub import snapshot_download
from trl import SFTConfig, SFTTrainer

# raw Pi sessions (JSONL), as the agent recorded them
path = snapshot_download("badlogicgames/pi-mono", repo_type="dataset")

# Pi's own format -> chat messages, keeping tool_calls and tool turns
dataset = sessions_to_messages(path)

trainer = SFTTrainer(
    model="Qwen/Qwen3-0.6B",
    train_dataset=dataset,
    args=SFTConfig(assistant_only_loss=True),
)
trainer.train()
```

Only because pi-mono is raw Pi JSONL. Traces already in chat format load with **load_dataset** and skip this.

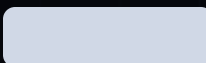


COMPRESS

Distillation: learn from a teacher

return a _ next token? (illustrative)

SFT TARGET

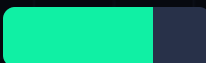
+  1.00

-  0.00

*  0.00

//  0.00

TEACHER

+  0.72

-  0.15

*  0.09

//  0.04

ON-POLICY: THE TEACHER GRADES THE STUDENT'S TOKENS

def	add	(a,	b):	return	a	-	b
0.98	0.97	0.95	0.96	0.93	0.97	0.04	0.91

The teacher puts 0.04 on -, so the student is corrected at the exact token where it went wrong.

A teacher gives a probability for every candidate token, where SFT gives only the right one.

On-policy, the student learns on text it wrote itself, so it sees its own mistakes.

The teacher: a bigger model, RL experts merged into one (MOPD), or the model itself with a hint (self-distillation).

Teacher and student need the same tokenizer, and the student is limited by how good the teacher is.



COMPRESS

Distillation in *TRL*

The student generates from the prompts and the teacher scores every token. On-policy by default.

distill.py

PYTHON

```
from trl import DistillationTrainer

trainer = DistillationTrainer(
    model="Qwen/Qwen3-0.6B",           # student
    teacher_model="Qwen/Qwen3-4B",    # same tokenizer
    train_dataset=prompts,            # prompts only
)
trainer.train()
```



REWARD

GRPO: Learn from a check you can run

ONE TASK

```
write  
is_palindrome(s)  
+ 3 hidden tests
```

THE MODEL TRIES 8 TIMES

#1 ✓✓✓

1.00

#2 ✓××

0.33

#3 ×××

0.00

#4 ✓✓✓

1.00

#5 ✓✓×

0.67

#6 ×××

0.00

#7 ✓✓✓

1.00

#8 ✓××

0.33

COMPARE

0.54

group average

above it:
better than
usual. Below:
worse.

UPDATE THE MODEL

↑ more likely

#1 #4 #5 #7

↓ less likely

#2 #3 #6 #8

EACH ✓ IS ONE HIDDEN TEST PASSED, AND THE FRACTION PASSED IS THE REWARD · ILLUSTRATIVE NUMBERS

The group is its own baseline, so there is no separate value model to train.

Tests, a compiler or a known answer all work as the reward.

Nobody shows the model the answer, so no demonstrator caps the result.



REWARD

GRPO in *TRL*

The reward is a plain Python function over the completions.

grpo.py

PYTHON

```
from trl import GRPOConfig, GRPOTrainer

def tests_pass(completions, test_list, **kwargs):
    # fraction of the task's asserts the code passes
    return [run_asserts(c[0]["content"], t)
            for c, t in zip(completions, test_list)]

trainer = GRPOTrainer(
    model="Qwen/Qwen3-0.6B",
    reward_funcs=tests_pass,
    train_dataset=mbpp,
    args=GRPOConfig(num_generations=8), # the group
)
trainer.train()
```



ACT

RL inside an environment

GRPO TRAINER RUNS THIS LOOP · ONE SANDBOX PER ATTEMPT · ILLUSTRATIVE ATTEMPT

reset()

a fresh sandbox
with the task
and its hidden
tests

→

model

calls
run_python

→

sandbox

a
NameError

→

model

fixes it,
runs it
again

→

sandbox

the right
output

→

model

answers:
done

→

get_reward()

runs the hidden
tests: 5 of 6
pass → 0.83

<

>

☐ model turns, trained on ☐ the OpenEnv sandbox answers

OpenEnv gives every environment one contract
(reset, step), served as a Hub Space.

The reward moves into the sandbox, and comes
after many turns of tool use.



ACT

Environments in + OpenEnv

Same `GRPOTrainer` as before. The class's methods become tools, `get_reward` is the reward.

● ● ● grpo_env.py

PYTHON

```
from trl import GRPOConfig, GRPOTrainer
from coding_env import CodeAction, CodingEnv # OpenEnv client

class MBPPEnv:
    def reset(self, tests=None, **kwargs): ... # fresh sandbox
    def get_reward(self) -> float: ... # hidden tests
    def run_python(self, code: str) -> str: # a tool
        r = self.client.step(CodeAction(code=code))
        return r.observation.stdout + r.observation.stderr

trainer = GRPOTrainer(
    model="Qwen/Qwen3-1.7B",
    train_dataset=mbpp_tasks,
    environment_factory=MBPPEnv,
    args=GRPOConfig(num_generations=8),
)
trainer.train()
```



HARNESSES

Frontier labs now train coding models across several real harnesses

 Kimi K3	builds its harnesses from modules, trains across them
 Qwen3-Coder-Next	agentic coding data from six harnesses
 MiMo-V2.6	a pool of task-adapted mini-harnesses
 Poolside Laguna	1.3B tokens of OpenHands, OpenCode, Mini-SWE-Agent
 Liquid LFM2.5	a harness picked at random per task





HARNESS

Who drives the loop: the trainer or the harness

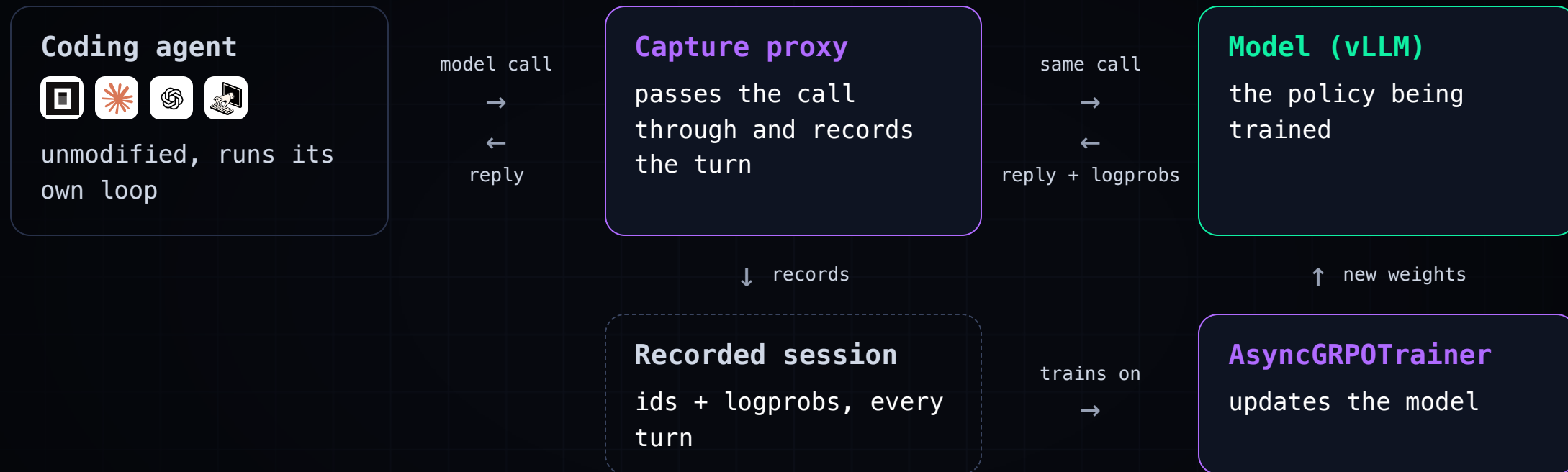


The environment we just saw was the white box: our trainer called the model and the tools. A harness is the black box. It calls the model itself.



HARNES

The capture proxy



The harness talks to what it thinks is a normal model API. The proxy forwards each call to vLLM and records the token ids and logprobs of every turn, which is what GRPO needs to train.



HARNESSES

In + OpenEnv

multi_harness.py

PYTHON

```
from harbor_env import HarborEnv # OpenEnv client
from trl.experimental.async_grpo import AsyncGRPOTrainer
from trl.experimental.async_grpo.openenv_harness import HarnessRolloutWorker

env = HarborEnv.from_hub(
    "FineEnvs/smoldataenv-multi-harness-harbor", # tasks and sandboxes
    harness=random.choice(HARNESSES), # picked per rollout
)

worker = HarnessRolloutWorker(
    harness_session_factory=env.session_factory,
    harness_adapter=None, # the agent owns its loop
    rollout_reward_fn=reward, # correctness and tool calls
)

AsyncGRPOTrainer(model=..., rollout_worker=worker).train() # keeps generating while it trains
```

The harnesses come from Harbor, and OpenEnv runs them. Swapping the harness is **one argument**.



HARNESSES

We tried it: multi-harness RL

One small model trained through four coding agents.

model	 LFM2.5-2.6B, which Liquid AI trained in other harnesses (Hermes Agent, OpenClaw)
harnesses	 OpenCode  Claude Code  Codex  Mini-SWE-Agent unmodified
tasks	SmolDataEnvs: data-analysis questions from real Kaggle notebooks, each with a grader. 1,000 to train, 250 held out
reward	1 for a correct answer, plus at most 0.1 for using fewer tool calls
two runs	OpenCode alone, and the four harnesses rotated across GRPO groups
recipe	AsyncGRPOTrainer, 1,000 steps, two H100s, one E2B sandbox per rollout



LFM2.5-2.6B, TRAINED THROUGH FOUR CODING AGENTS

42% → 54%

BEFORE

AFTER

of 250 held-out data-analysis tasks solved, averaged over

 OpenCode  Claude Code  Codex  Mini-SWE-Agent

31%

fewer tool calls, on tasks both solved

4 / 4

harnesses improved

Trained in OpenCode alone it reaches 58% there, and gains much less everywhere else.

Source: [The ultimate guide to multi-harness RL](#)



RECAP

Which one to reach for

You have traces of the agent you want

SFT

You have a stronger model with your tokenizer

Distillation

You can check the answer

GRPO

The task takes many steps with tools

GRPO + OpenEnv

Users run it inside a harness

Train through the harness

They combine. DeepSeek-V4 runs SFT then GRPO per domain, then merges the experts with on-policy distillation. Our multi-harness run went straight to RL from an already post-trained model.



OVER TO YOU

Go deeper

TRAINING AGENTS, LIVE

[#1 SFT on agent traces](#)

[#2 Distillation](#)

[#3 RL with GRPO](#)

[#4 RL environments](#)

[The whole playlist](#)

READ

[Distillation in 2026](#)

[Agentic RL in 2026](#)

[RL environments in 2026](#)

[Multi-harness RL](#)

[Agent glossary](#)

RUN

[SFT on tool calls](#)

[Distillation](#)

[GRPO + OpenEnv](#)

[AsyncGRPO + Harbor](#)

[Multi-harness models](#)

CODE

[TRL](#)

[OpenEnv](#)

[TRL docs](#)

[OpenEnv docs](#)

[Environments on the Hub](#)

The deck is at hf.co/spaces/sergiopaniego/train-the-agent-slides

Sergio Paniego [@sergiopaniego](#) Hugging Face



THESE SLIDES