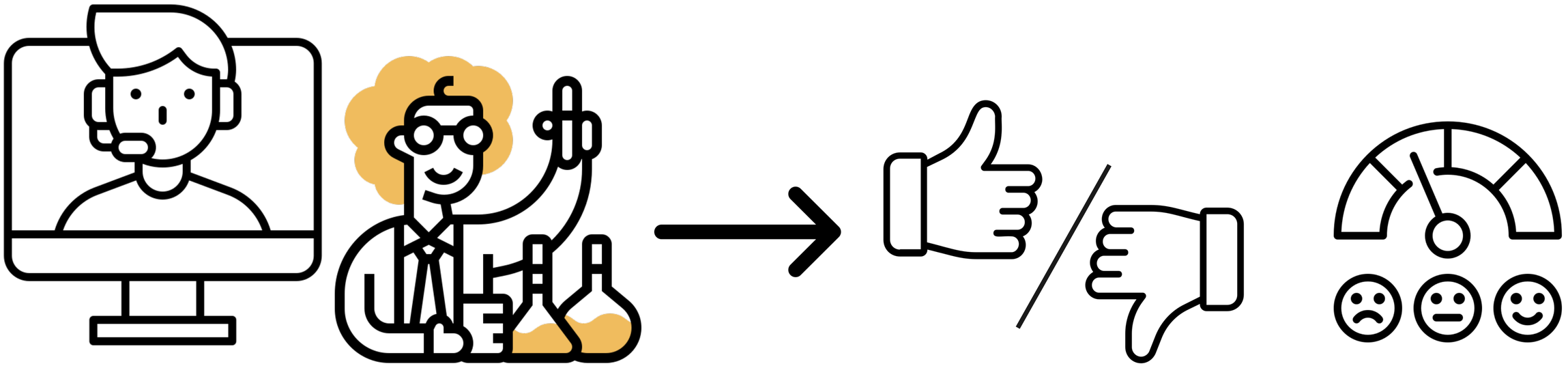
Meta-Harness: End-to-End Optimization of Model Harnesses

Yoonho Lee

Stanford Artificial Intelligence Laboratory
IRIS Lab

Learning From Experience



hours/days of work

sparse feedback

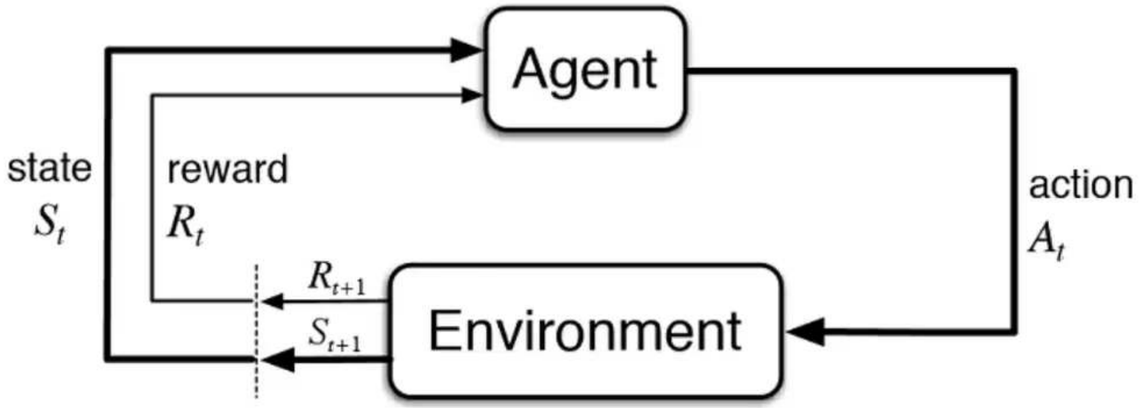
- This information channel **doesn't grow with task complexity**
- ...and this will only get worse as we deploy on harder tasks

Towards *Reflective* Learning

Supervised learning

+ Rich learning signal!

- Capped by data collector

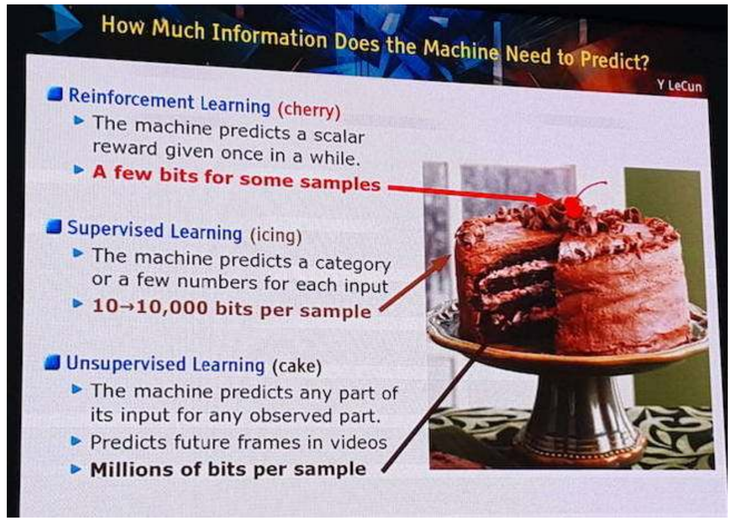


Reinforcement learning

+ Learns from experience!

- “sucking supervision through a straw”

- In practice, pure trial-and-error

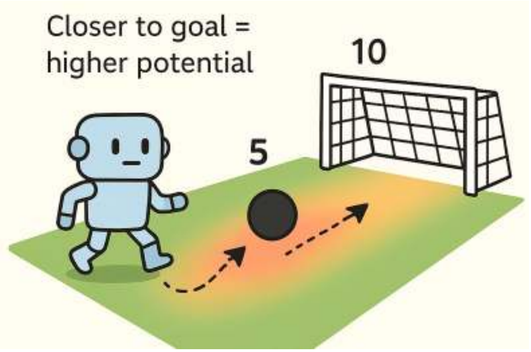


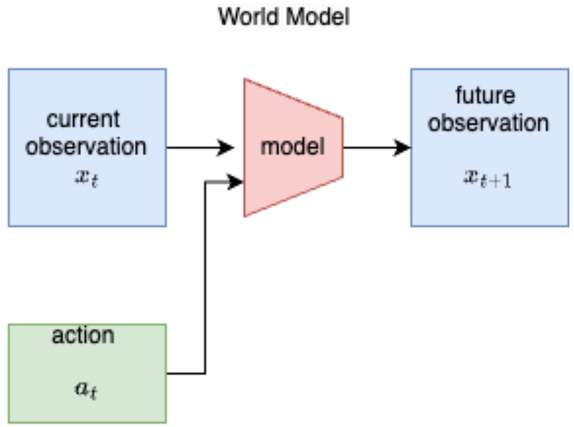
Reflective learning?

+ Learns from experience, *without* discarding info?

Towards *Reflective* Learning — Related Ideas

- Reward shaping / dense rewards
  - Not very scalable to harder tasks. The “reflection” is just *manually pulled back* into reward design
- World model / model-based RL
  - Learns from non-reward signals (states or latents)
  - “reflection” is single-thread planning
- Value functions
  - Collapses <u>all</u> futures into a *single number*
  - **Perfect V/Q** is a sufficient statistic for optimality, but discards a lot of useful info + is estimated





$$v_\pi(s) = \mathbb{E}_\pi\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s\right]$$

Value function — Expected discounted return — Starting at state s

Meta-Harness: End-to-End Optimization of Model Harnesses

Yoonho Lee
Stanford

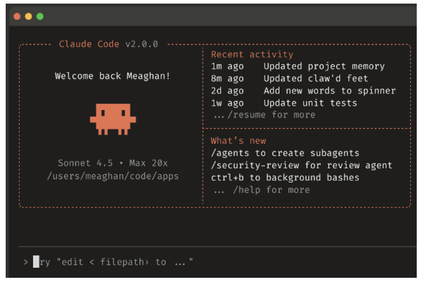
Roshen Nair
Stanford

Qizheng Zhang
Stanford

Kangwook Lee
KRAFTON

Omar Khattab
MIT

Chelsea Finn
Stanford

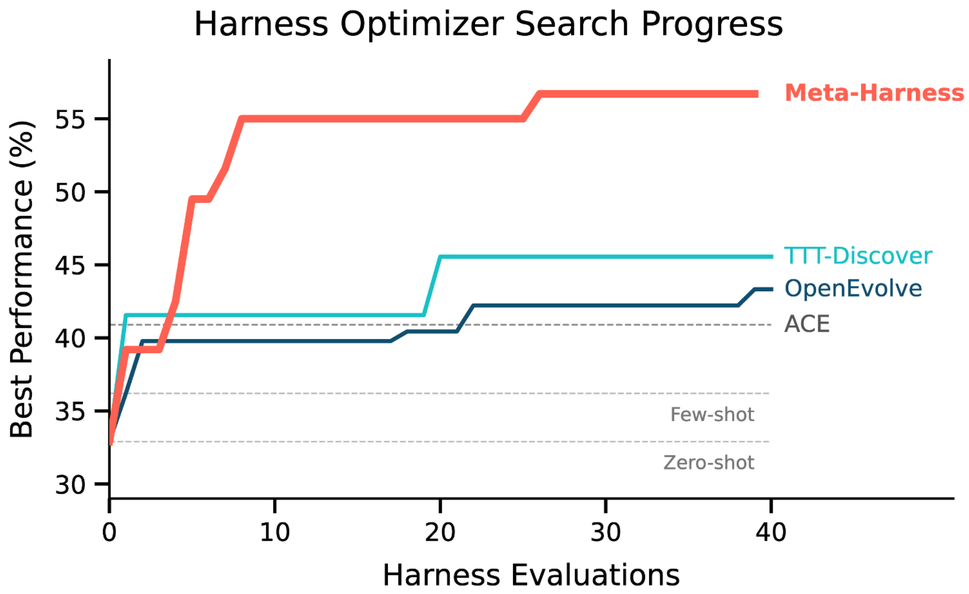


- Model Harness: the code layer around the raw LLM
  - Claude Code, Codex CLI, Gemini CLI, OpenHands, Terminus 2, Pi…
  - ***Very*** important for performance
    - TerminalBench-2 w/ Opus-4.6: 58% -> 81%
    - GAIA L3 w/ Sonnet-4.5: 15% -> 65%
- We're proposing a *harness for optimizing harnesses*

What properties should a ***Meta- Harness*** *have?*

Harness Engineering as a Coding/Optimization Problem

- A harness is just a specific type of code (e.g., written in Python)
- Benchmark scores tell you how good a given harness is
- We have text optimizers (AlphaEvolve, GEPA, TTT-Discover…) that can hill-climb text -> score mappings

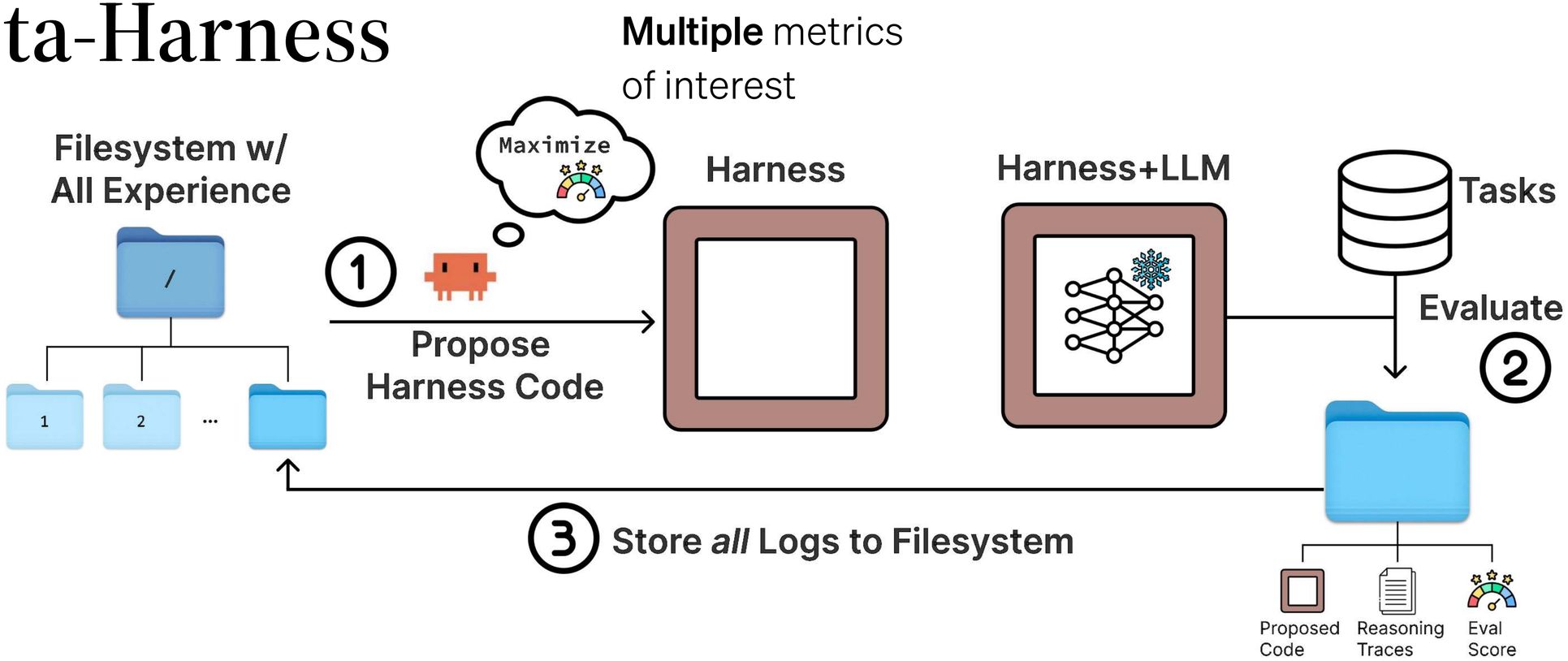


Prior methods were designed for tightly scoped settings:

- sphere packing
- prompts
- kernels

Each harness rollout is very long (~1 MTok), and you need to look across multiple rollouts

Meta-Harness



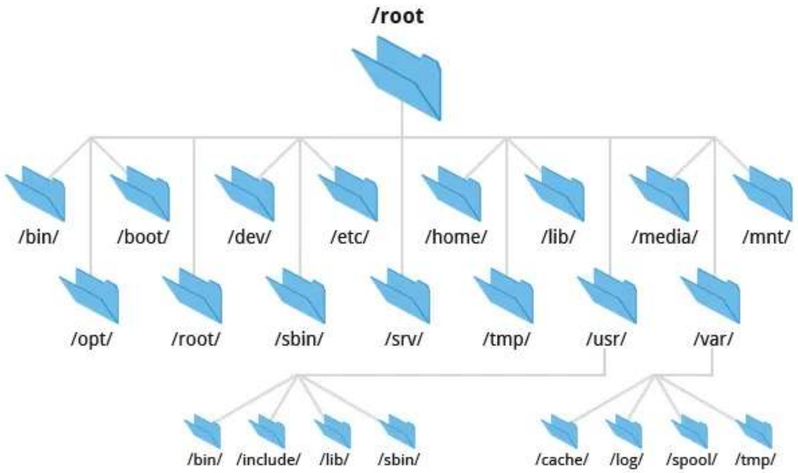
Very minimal; one core design choice

- Store ***all*** experience in a single filesystem
  - Use a *coding agent* for proposal

No: parent selection/exploration, crossover/mutation, separate feedback module, special data structure, separate memory/retrieval stack.

Filesystem as a Primitive for Storing Experience

- Very close to the LLM training distribution
  - Very mature tools: ls, cd, mv, grep, cat...
- Effectively no storage limits: no need for upfront compression
- Human-readable and editable
- Composable, i.e., you can put dirs in dirs
- Great versioning infra (git)

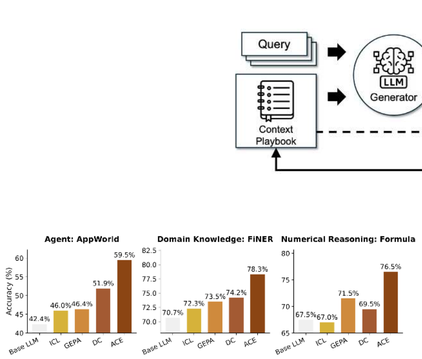


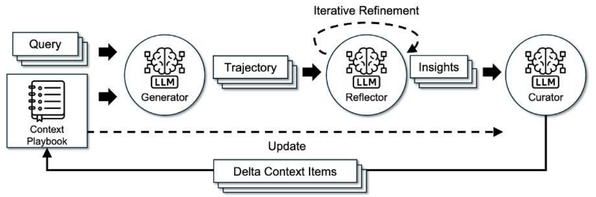
Example: Linux filesystem

Exp 1: Text Classification

Setting: agent learns from (query, prediction, answer) tuples.
Evaluated on held-out examples

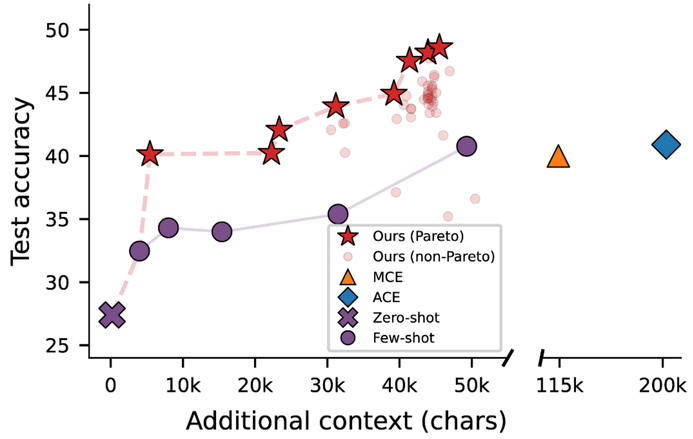
Harness	Datasets			Avg.	
	USPTO	S2D	Law	Acc	Ctx ↓
Zero-Shot	12.0	63.2	7.0	27.4	0
Few-Shot (8)	14.0	67.9	21.0	34.3	2.0
Few-Shot (32)	13.0	72.2	21.0	35.4	7.9
Few-Shot (all)	15.0	78.3	29.0	40.8	12.3
MCE [52]†	14.0	83.0	23.0	40.0	28.5
ACE [59]†	**16.0**	77.8	29.0	40.9	50.8
`Meta-Harness`	14.0	**86.8**	**45.0**	**48.6**	11.4





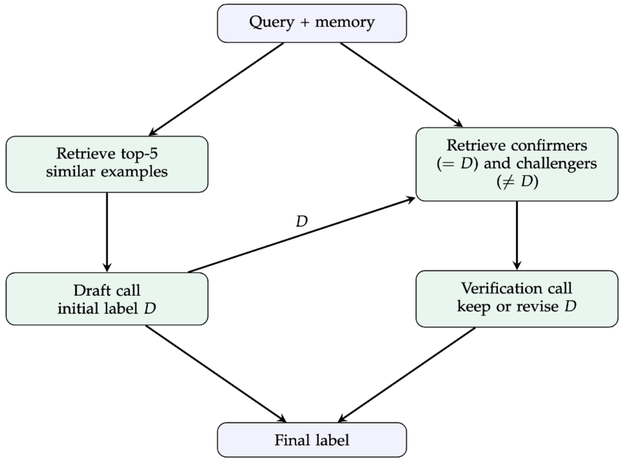
Method	GT Labels	FiNER (Acc↑)	Formula (Acc↑)	Average
DeepSeek-V3.1 as Base LLM				
Base LLM		70.7	67.5	69.1
Offline Adaptation				
ICL	✓	72.3+1.6	67.0−0.5	69.6+0.5
MIPROv2	✓	72.4+1.7	69.5+2.0	70.9+1.8
GEPA	✓	73.5+2.8	71.5+4.0	72.5+3.4
ACE	✓	**78.3**+7.6	**85.5**+18.0	**81.9**+12.8
ACE	✗	71.1+0.4	83.0+15.5	77.1+8.0
Online Adaptation				
DC (CU)	✓	74.2+3.5	69.5+2.0	71.8+2.7
DC (CU)	✗	68.3−2.4	62.5−5.0	65.4−3.7
ACE	✓	**76.7**+6.0	76.5+9.0	**76.6**+7.5
ACE	✗	67.3−3.4	**78.5**+11.0	72.9+3.8

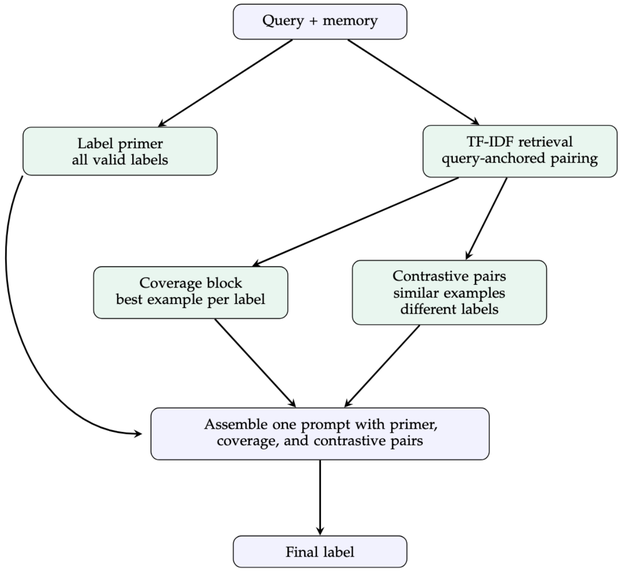
Zhang et al, ICLR 2026



Meta-Harness autonomously discovered a harness that outperforms the prior state of the art by 7.7 points at >4x token efficiency!

Exp 1: Text Classification



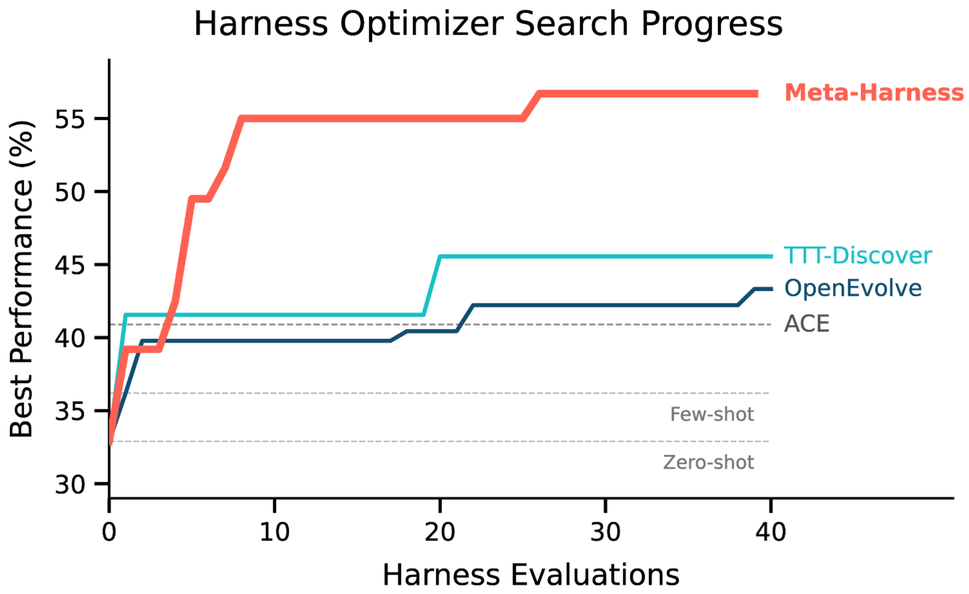


- Retrieve 5 → initial draft → verification
- Prime with all valid labels → per-label relevant + contrastive examples
- Split memory into successes and mistakes, retrieve seperately
- Store + retrieve reasoning traces for correct predictions
- Retrieval with TF-IDF w/ RRF + word-level Jaccard + char-level 4-gram similarity
- ...

Exp 1: Text Classification

Harness	SciC	FiNER	Amz5	FPB	GoEmo	Bank77	News	SciT	TwHate	Avg Acc	Ctx ↓
Zero-shot	32.7	56.0	52.7	90.0	42.0	80.7	84.7	89.3	75.3	67.0	-
Few-shot (8)	34.0	63.0	54.0	90.0	44.0	82.7	84.7	**91.3**	76.7	68.9	2.2
Few-shot (32)	38.7	62.0	53.3	90.7	43.3	**86.0**	85.3	90.7	76.7	69.6	5.2
Few-shot (all)	35.3	61.0	50.0	93.3	42.7	80.7	84.0	90.0	76.7	68.2	7.4
ACE [59]	40.7	**74.0**	48.0	**96.7**	44.0	83.3	86.0	90.7	68.7	70.2	11.7
`Meta-Harness`	**53.3**	67.0	**60.0**	94.0	**46.0**	82.7	**86.7**	**91.3**	**77.3**	**73.1**	7.3

Gains transfer to 6/9 unseen classification tasks



Compared to prior text optimizers:

- Reaches their final perf after a few steps
- >10% higher final acc

Exp 2: Agentic Coding

Setting: TerminalBench-2. 89 coding tasks that require multiple steps of execution (up to hours)

Very big + expensive: one full eval is >10 MTok + >$1000 in Opus API costs

> Couldn't find a reasonable way to even try previous text optimizers

The same discovered harness improves performance on Opus and Haiku!

Harness	Auto	Pass (%)
Claude Opus 4.6		
Claude Code	×	58.0
Terminus 2	×	62.9
Mux	×	66.5
Droid	×	69.9
TongAgents	×	71.9
MAYA-V2	×	72.1
Terminus-KIRA	×	74.7
Capy	×	75.3
ForgeCode	×	81.8
Meta-Harness	✓	**76.4**
Claude Haiku 4.5		
OpenHands	×	13.9
Claude Code	×	27.5
Terminus 2	×	28.3
Mini-SWE-Agent	×	29.8
Terminus-KIRA	×	33.7
Goose	×	35.5
Meta-Harness	✓	**37.6**

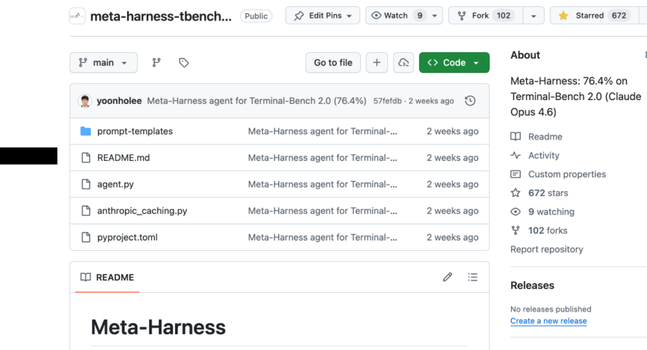
Exp 2: Agentic Coding

Showing 142 entries

Clear filters

Search leaderboard | Select agents | Select models | Select… | Verified only

Rank	Agent	Model	Date	Agent Org	Model Org	Accuracy
1	NexAU-AHE	GPT-5.5	2026-05-14	china-qijizhifeng	OpenAI	**84.7%** ± 2.1
2	LemonHarness	Multiple	2026-05-14	LR AILab of Lenovo CTO Org	Multiple	**84.5%** ± 2.6
3	Capy	GPT-5.5	2026-05-14	Capy	OpenAI	**83.1%** ± 2.1
4 ✓	Codex CLI	GPT-5.5	2026-04-23	OpenAI	OpenAI	**82.2%** ± 2.2
5	Polaris	Multiple	2026-05-14	PolarisOps	Multiple	**82.2%** ± 2.8
6	WOZCODE	Claude Opus 4.7	2026-05-14	WOZCODE	Anthropic	**80.2%** ± 2.1
7	TongAgents	Gemini 3.1 Pro	2026-03-13	BIGAI	Google	**80.2%** ± 2.6
8	LemonHarness	Multiple	2026-05-14	LR AILab of Lenovo CTO Org	Multiple	**79.9%** ± 3.0
9	SageAgent	GPT-5.3-Codex	2026-03-13	OpenSage	OpenAI	**78.4%** ± 2.2
10	Droid	GPT-5.3-Codex	2026-02-24	Factory	OpenAI	**77.3%** ± 2.2
11	Meta-Harness	Claude Opus 4.6	2026-05-14	Stanford IRIS	Anthropic	**76.4%** ± 2.4
12	CodeBrain-1.5	GPT-5.3-Codex	2026-02-10	Feeling AI	OpenAI	**75.8%** ± 2.0



Exp 2: Agentic Coding

Iterations 1–2: promising bugfixes are confounded by prompt edits. The first two iterations both bundle plausible structural fixes with prompt-template modifications, and both regress sharply from the 64.4% Terminus-KIRA baseline. Iteration 1 targets observation corruption from leaked terminal markers and adds a loop breaker:

```
Hypothesis: __CMDEND__ marker fragments leak into LLM observations on long-running
tasks, causing the model to get confused and enter infinite no-tool-call loops.
Stripping these markers + adding a loop breaker will recover wasted steps.
```

Iteration 3: the proposer identifies the confound. By iteration 3, the proposer explicitly infers that the regressions are not primarily due to the structural bugfixes themselves:

```
Prior attempts: evo_marker_fix (58.9%, -5.6pp), evo_single_confirm (57.8%, -6.7pp)
--- both regressed. Root cause of regressions: Prompt template changes (cleanup
directives) caused the agent to delete necessary state before task completion. The
structural bugfixes were confounded with harmful prompt changes. evo_strip_only
isolates the two proven structural fixes.
```

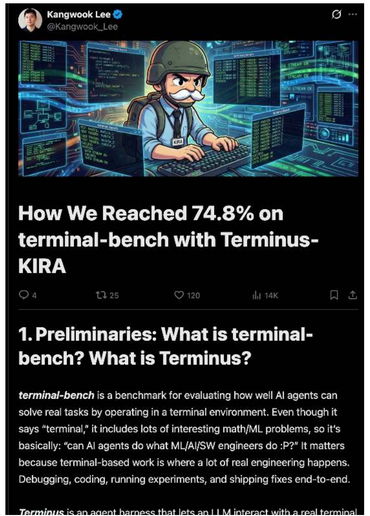
Iteration 7: the winning candidate. After six consecutive regressions, the proposer shifts strategy from modifying the control loop to adding information before the loop begins:

```
All 6 prior iterations regressed from the 64.4% baseline because they modified the
completion flow, prompt template, or observation processing. evo_env_bootstrap takes
a different approach --- purely additive. It gathers an environment snapshot via a
single shell command before the first LLM call and appends it to the initial prompt.
No other methods are changed. This should eliminate 3--5 wasted exploration turns on
dependency-heavy tasks without risking regression on already-passing tasks.
```

The proposer...

- Reads a median of 82 files before proposing a harness.
- Based on what it read, explicitly reasons through *why its previous attempts failed.*

Surprisingly close to how human engineers hill-climb this benchmark!



This post is about how we diagnosed seven distinct failure modes, fixed them systematically, and reached **78.4% SOTA** with `gemini-3.1-pro-preview` — and why those fixes generalized across models instead of overfitting to a single provider.

Failure Mode 1: Same model, very different performance

Our agent was built for interactive use. It asks clarifying questions when requirements are ambiguous, confirms architectural decisions before proceeding, and checks in with the user when it is uncertain about scope. This is exactly the right behavior in a chat interface.

In a benchmark environment, it is catastrophic.

TermBench tasks are graded on completion. There is no user to answer clarification requests. Every turn spent asking a question is a turn not spent solving the problem. Our agent was failing tasks not because it lacked the intelligence to solve them, but because it was waiting for a human who was never coming.

Fix: We introduced a strict **Non-Interactive Mode** — a separate runtime profile activated during evaluation:

Conclusion & Discussion

- **Meta-Harness** is a simple iterative loop built around the idea of *storing all past experience in a filesystem and letting the model decide what to view*
  - A new instantiation of meta-learning ideas: take important hand-designed parts of learning → optimize them end-to-end
- We autonomously improved carefully designed state-of-the-art harnesses
- In my view, this is a step towards true continual learning in text space
- Promising directions:
  - The evals are the bottleneck. Refine or propose new evals based on experience.
  - Co-evolve weights & harness: test hypotheses in text space, distill into weights
  - End-to-end training for important primitives (idea gen, prompt editing…)