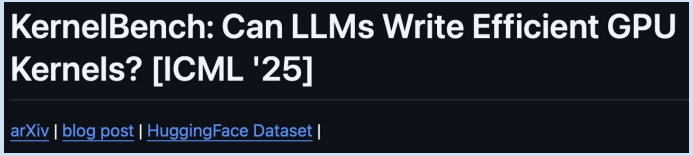
Rethinking the role of harnesses in LMs

Alex Zhang
PhD student in CSAIL advised by Omar Khattab / Tim Kraska

What I used to do, and how that got me to harnesses

“LM capabilities are a function of training data, whose size & quality is a function of what we care about solving / benchmarking.”

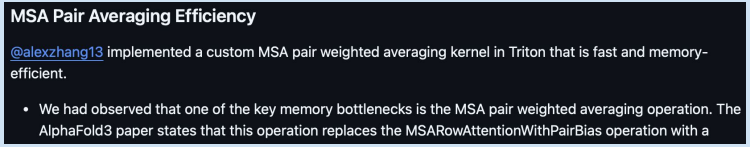




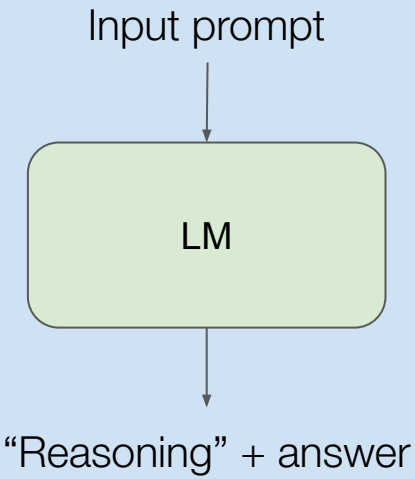


“LM systems and kernel design are iterative with clear objectives, but data constrained. How do we accelerate such tasks?”

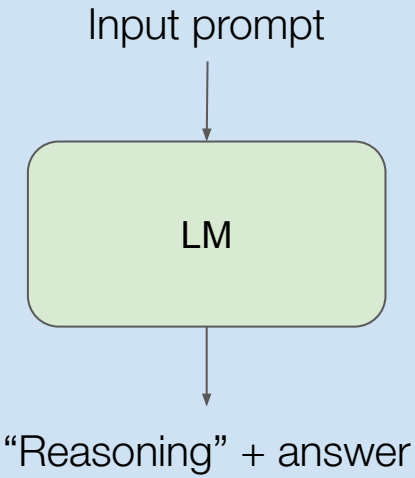


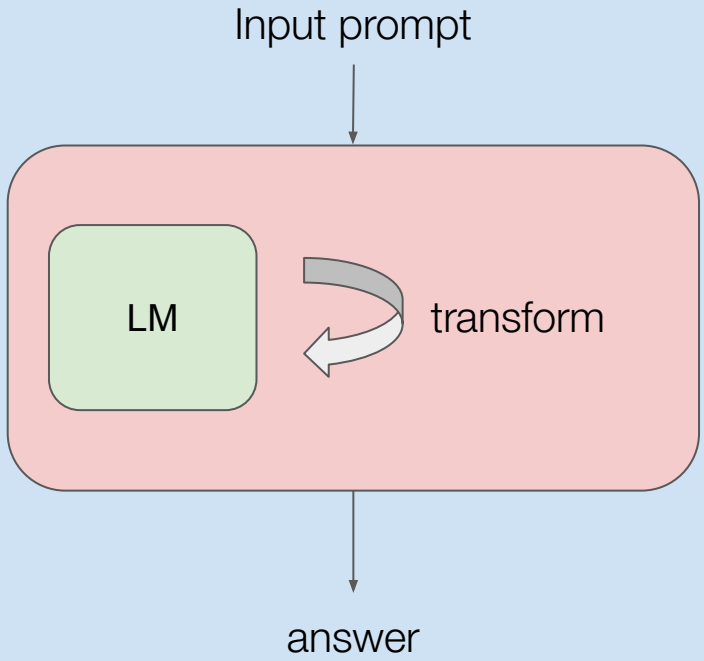


Typical language model workflow



Typical language model workflow

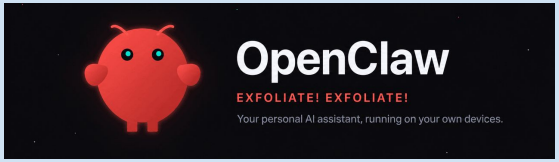




What does the “standard” agent look like?

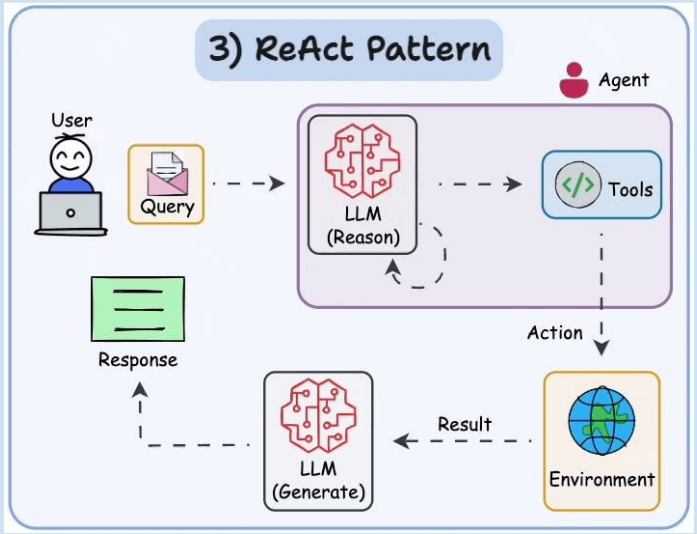








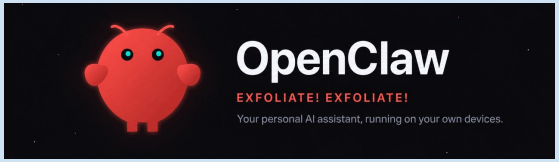




What does the “standard” agent look like?











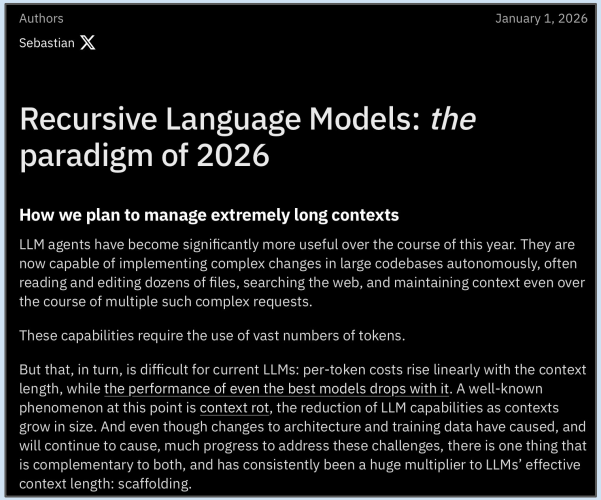

```python
def pi(prompt):
    history = [prompt]

    while True:
        prompt = render(history, tools)
        reply = llm_call(prompt)
        history.append(reply)

        if not reply.tool_calls:
            return reply.text

        results = parallel_map(run_tool, reply.tool_calls)
        history.extend(results)
```

Start of 2026: Context management in language models



Authors

Sebastian

January 1, 2026

Recursive Language Models: *the* paradigm of 2026

How we plan to manage extremely long contexts

LLM agents have become significantly more useful over the course of this year. They are now capable of implementing complex changes in large codebases autonomously, often reading and editing dozens of files, searching the web, and maintaining context even over the course of multiple such complex requests.

These capabilities require the use of vast numbers of tokens.

But that, in turn, is difficult for current LLMs: per-token costs rise linearly with the context length, while the performance of even the best models drops with it. A well-known phenomenon at this point is context rot, the reduction of LLM capabilities as contexts grow in size. And even though changes to architecture and training data have caused, and will continue to cause, much progress to address these challenges, there is one thing that is complementary to both, and has consistently been a huge multiplier to LLMs' effective context length: scaffolding.

Jan 1. Prime Intellect reveals they are working on RLMs.



Deedy @deedydas · Jan 6

Claude Code is one of the worst named products. It's not just for **code**.

Watch this Hermès ad.

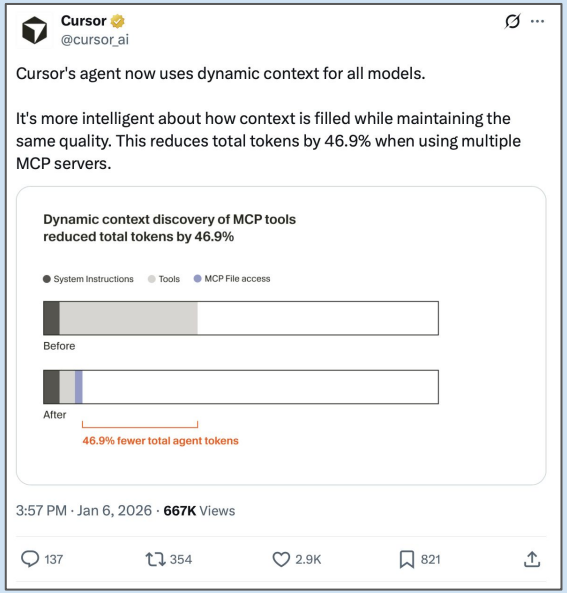
30 seconds. 8 shots. Voice. Music. Even text branding burned in with ffmpeg. Clopus 4.5 wrote a script, orchestrated ElevenLabs, Google Veo 3, downloaded music and made this from

Show more

0:23

281 331 4.8K 1M

Past few months: people are trying Claude Code for non-coding tasks.



Cursor

@cursor_ai

Cursor's agent now uses dynamic context for all models.

It's more intelligent about how context is filled while maintaining the same quality. This reduces total tokens by 46.9% when using multiple MCP servers.

Dynamic context discovery of MCP tools reduced total tokens by 46.9%

System Instructions Tools MCP File access

Before

After

46.9% fewer total agent tokens

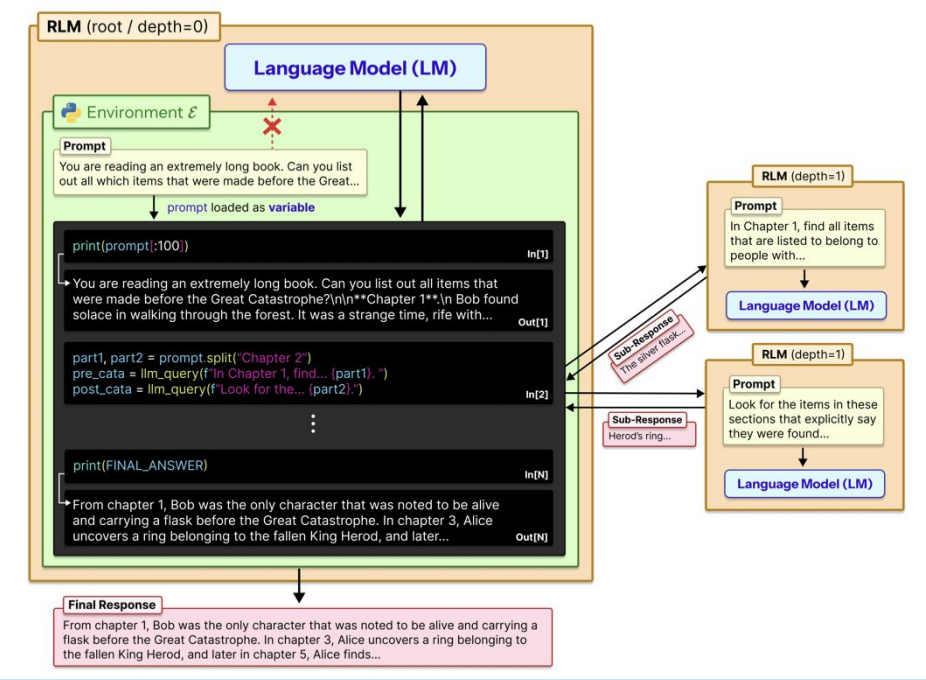
3:57 PM · Jan 6, 2026 · **667K** Views

137 354 2.9K 821

Jan 6. Cursor release dynamic context.

Recursive Language Models (2025)

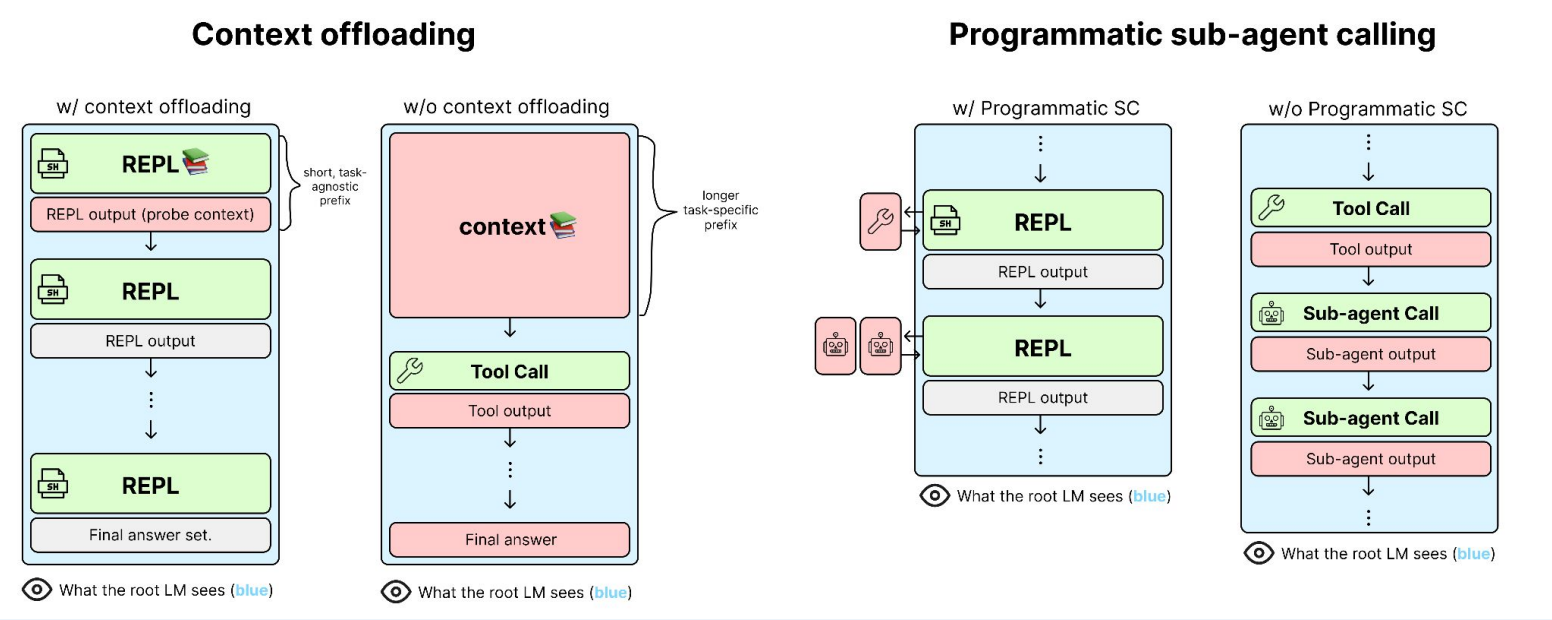
- Should language models be entirely neural?
- Do harnesses matter, and what abstractions are important in designing a harness?
- Long-context is a *silly* problem.



Recursive Language Models (Zhang, Kraska, Khattab, NeurIPS 2026)

Recursive Language Models (2025)

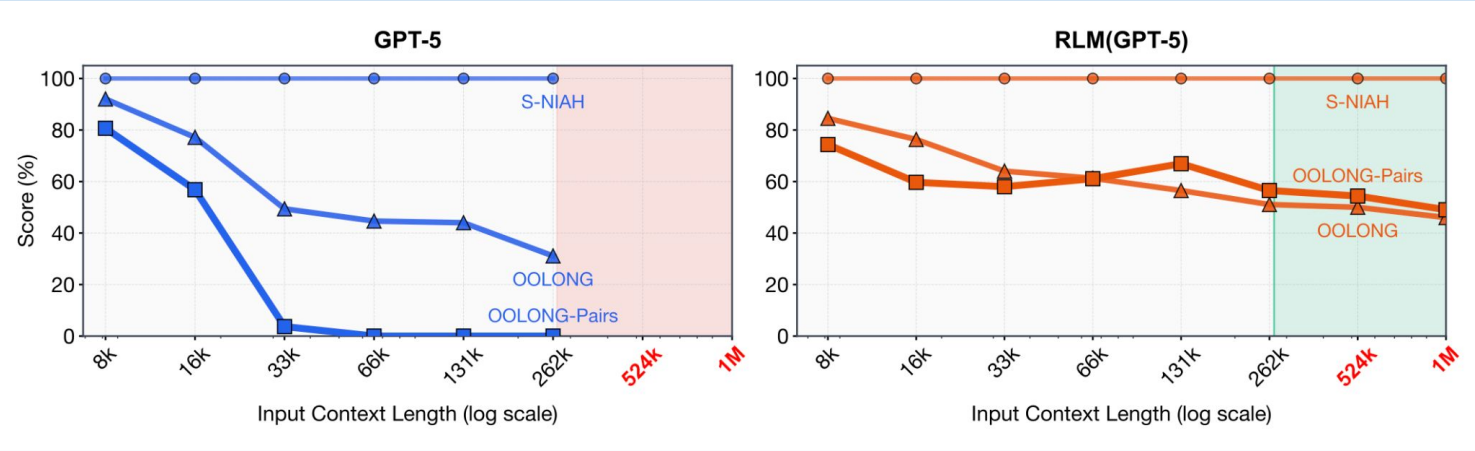
"LMs are constrained around their architecture. Harnesses are opinionated choices on top of a blackbox model, to get around that model's constraints."



Recursive Language Models (Zhang, Kraska, Khattab, NeurIPS 2026)

Recursive Language Models (2025)

"LMs are constrained around their architecture. Harnesses are opinionated choices on top of a blackbox model, to get around that model's constraints."



Recursive Language Models (Zhang, Kraska, Khattab, NeurIPS 2026)

RLMs are the new reasoning models

April 20, 2026

Reasoning models were the first clear proof that language model capability can scale with test-time compute. Recursive language models (RLMs) ask what the correct abstraction for spending that compute is.

The insight behind RLMs is obvious in hindsight: it is the direct marriage of two important axes of model capability — reasoning and tool use. This is more radical than it first sounds. RLMs collapse reasoning and tool use into a single inference abstraction: the model treats its own prompt as an environment it can inspect, slice, and recursively query. *Context itself becomes the object of computation.*

From Raymond Weitekamp (2026): https://raw.works/

Are “LM agents” analogous to an RLM reasoning chain?

mini-swe-agent / src / minisweagent / agents / default.py ↑ Top

Code Blame 155 lines (137 loc) · 6.21 KB Raw

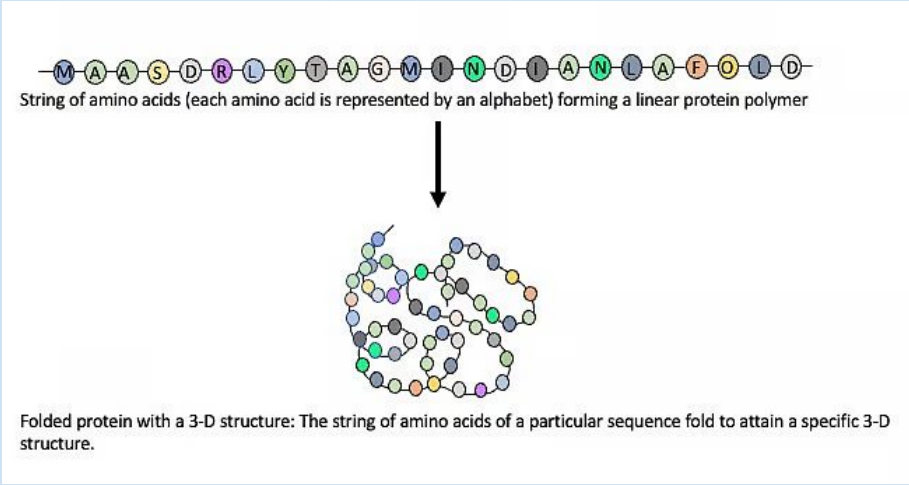
```python
class DefaultAgent:
    def step(self) -> list[dict]:
        """Query the LM, execute actions."""
        return self.execute_actions(self.query())

    def query(self) -> dict:
        """Query the model and return model messages. Override to add hooks."""
        if 0 < self.config.step_limit <= self.n_calls or 0 < self.config.cost_limit <= se
            raise LimitsExceeded(
                {
                    "role": "exit",
                    "content": "LimitsExceeded",
                    "extra": {"exit_status": "LimitsExceeded", "submission": ""},
                }
            )
        self.n_calls += 1
        message = self.model.query(self.messages)
        self.cost += message.get("extra", {}).get("cost", 0.0)
        self.add_messages(message)
        return message

    def execute_actions(self, message: dict) -> list[dict]:
        """Execute actions in message, add observation messages, return them."""
        outputs = [self.env.execute(action) for action in message.get("extra", {}).get("a
        return self.add_messages(*self.model.format_observation_messages(message, outputs
```

e.g. mini-swe-agent is ~100 lines of code!

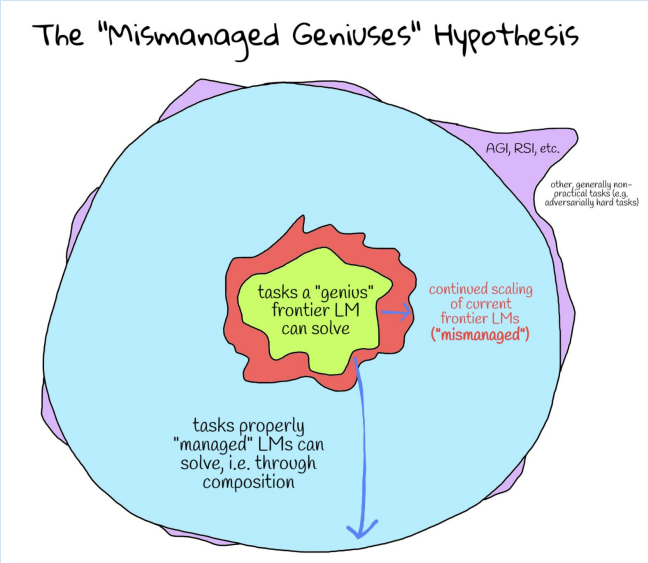
Suppose an LM is asked to produce an algorithm for *protein folding*, or some other big task.



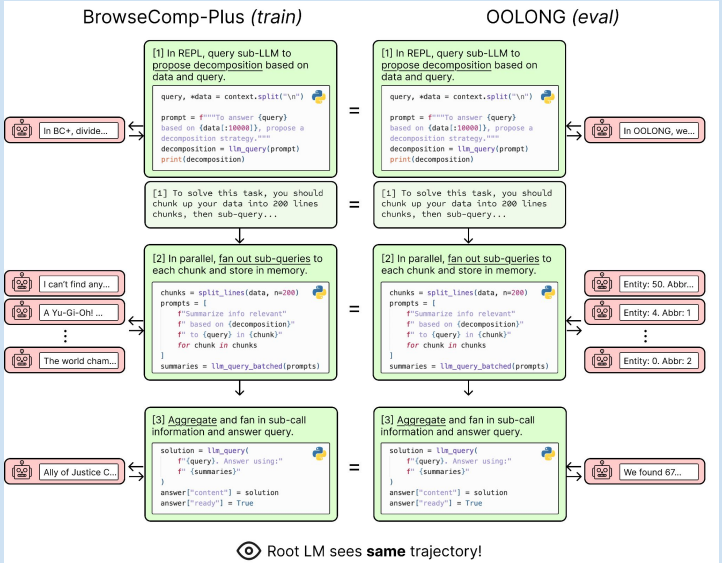
String of amino acids (each amino acid is represented by an alphabet) forming a linear protein polymer

Folded protein with a 3-D structure: The string of amino acids of a particular sequence fold to attain a specific 3-D structure.

What exactly does the harness do?



"Language models are sufficiently capable of composing themselves to solve *harder* tasks. This a property of tasks themselves."



"Harnesses can enable wider task transfer, by *implicitly* shaping what the model sees."

[The Mismanaged Geniuses Hypothesis](#) (Zhang, Li, Khattab, 2026)

Re-thinking the role of the harness.

Language Models will be Scaffolds

Created in February 25, 2026

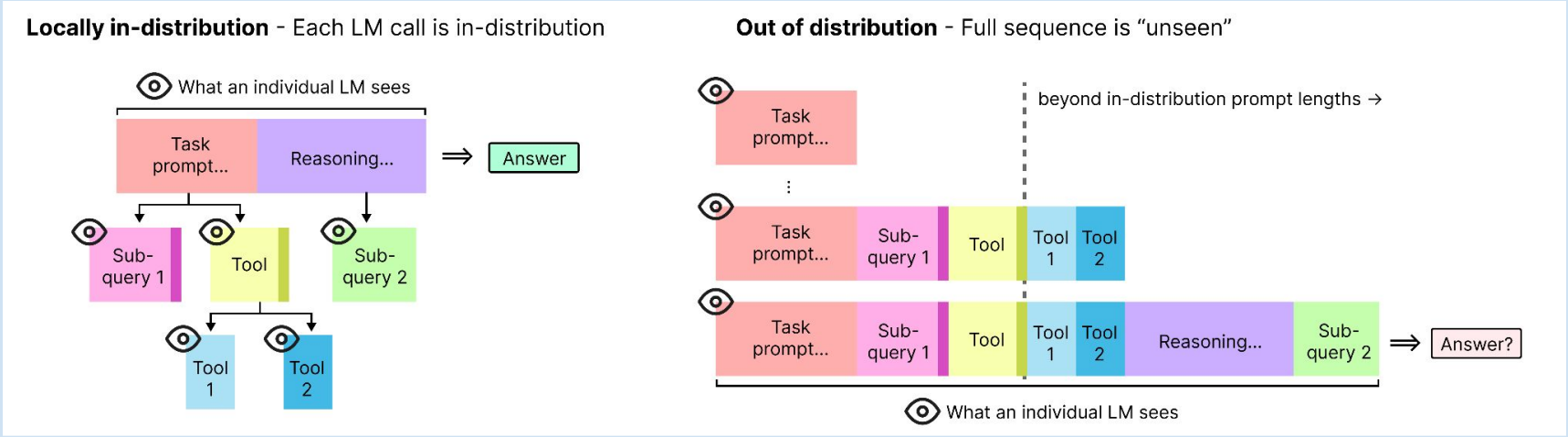
2026 · # scaffolds # language # models

Alex Zhang, Feb 25, 2026.

I have been somewhat convinced since before I started my PhD that the *language models we interact with in the (near) future* ***will be what we call scaffolds today***. For the earlier half of this decade, it was generally believed that any work other than improving the base "neural" language model (i.e. an end-to-end neural network model like Opus 4.6 or Qwen3-8B) was a contrarian take on Sutton's *Bitter Lesson* that would ultimately fall victim to scale. And this belief was genuinely a good bet since the invention of the Transformer in 2017: religiously following this bet is how companies like OpenAI and Anthropic have exploded in valuation since. The inability to follow this bet is also what has led to academia's weakening presence in AI, and the growing pull of industry labs for ambitious, young researchers.

For the second half of this decade, my intuition tells me to bet *differently*. I'm not saying that scaling is dead; scaling is quite literally the key to everything in a data-driven strategy like deep learning. It's more that language models are really good now: so good, that I theorize that existing **neural** language models are actually **severely underutilized**. I am implying that they are

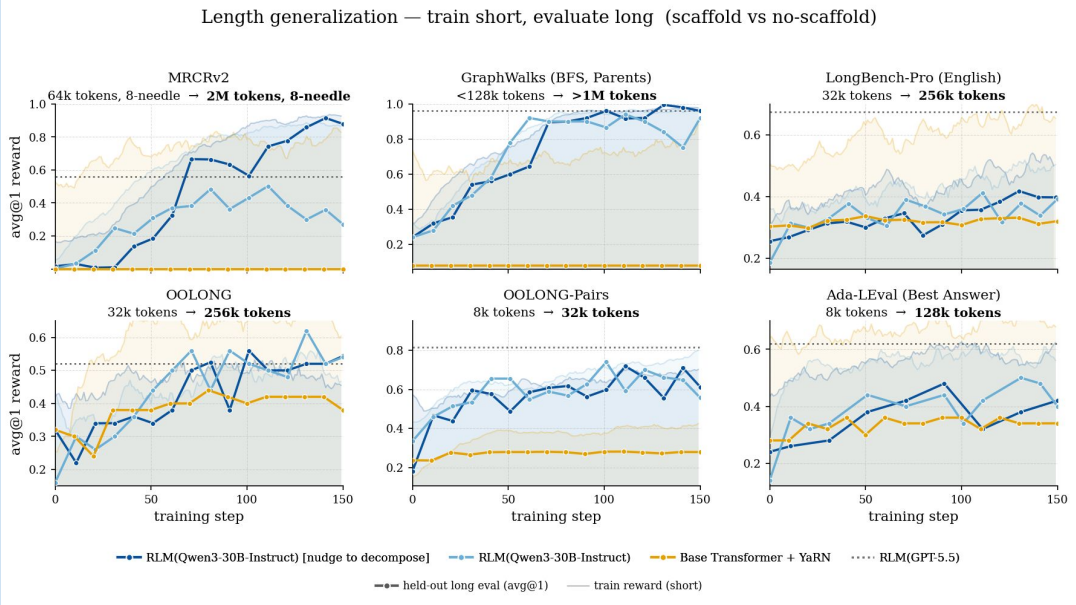
Harness Design for “Locally In-Distributioness”



[Language Model Harnesses are Compositional Generalizers](#) (Zhang, Khattab, 2026)

Removing generalization pressure from the LM.

Setup: Train only on **short** tasks, evaluate exclusively on ***longer*** variants.



[Language Model Harnesses are Compositional Generalizers](#) (Zhang, Khattab, 2026)

Relating generalization performance to a train trajectory.



`max`(re-executing top-25 closest train trajectories on eval contexts)

Task	RLM K=25	Depth-0 K=25
MRCR	0.000	0.000
GraphWalks	0.208	0.112
LongBench-Pro	0.171	0.203
OOLONG	0.680	0.640
OOLONG-Pairs	**0.730**	**0.490**
BestAnswer	0.000	0.000
Strategy-OOLONG	0.240	0.240

Language Model Harnesses are Compositional Generalizers (Zhang, Khattab, 2026)

Train r=1.00 · 8 turns · lengthgen-oolong-nohints/train@118/57

```
['answer', 'context_0', 'context', 'lines']
```

USER

Turn 3/20:

ASSISTANT

REPL · PYTHON

```
# Filter out the header/footer lines and keep
# We can identify real data lines by presence
data_lines = []
for line in lines:
    if "User: " in line and "Instance: " in li
        data_lines.append(line.strip())

print(f"Number of valid data lines: {len(data_
print(f"Sample of valid data lines:\n{data_lin
```

OBSERVATION

Eval r=1.00 · 8 turns · lengthgen-oolong-nohints/eval@131/23

REPL VARIABLES

```
['answer', 'context_0', 'context', 'lines']
```

USER

Turn 3/20:

ASSISTANT

REPL · PYTHON

```
# Remove header and footer lines (non-question
# We'll keep only lines that contain "User:" a
questions = []
for line in lines:
    if "User:" in line and "Instance:" in line
        questions.append(line.strip())

print(f"Number of valid question lines after f
print(f"Sample of 5 valid lines:\n{questions[:
```

OBSERVATION

Post-Training RLM Agents for End-to-End M&A Diligence

New research with Baseten shows how recursive language model harnesses and post-training can improve AI performance on complex M&A diligence.

by Niko Grupen, Julio Pereyra, Gabe Pereyra, Charlie O'Neill, Mudith Jayasek
Aaron Ellis-Bloor, Jonathon Liu, and Matthew Blau • Sep 8, 2026

We believe the next breakthrough in long-horizo
models to manage their own context.

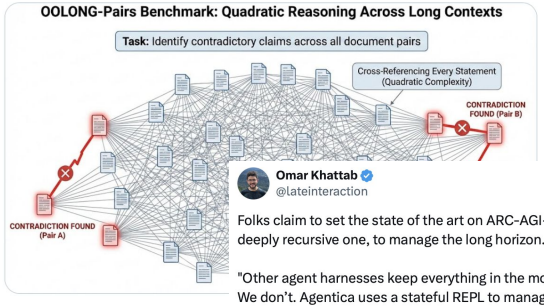
Introducing our new research direction on Recur

We are sharing our initial experiments

Google Cloud Tech
@GoogleCloudTech

Recursive Language Models (RLMs) let agents manage
delegating tasks recursively.

This Google Cloud Community Article explains why AD
choice for re-implementing the original RLM codebase
enterprise-ready format →goo.gle/4kjT12E



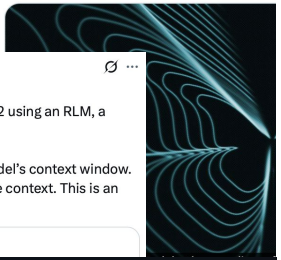
6:00 PM · Feb 7, 2026 · **932.8K** Viev

Omar Khattab
@lateinteraction

Folks claim to set the state of the art on ARC-AGI-2 using an RLM, a deeply recursive one, to manage the long horizon.

"Other agent harnesses keep everything in the model's context window. We don't. Agentica uses a stateful REPL to manage context. This is an RLM-style loop."

Agentica @agenticasdk · Feb 11



AUGUST 24, 2026 · A LAUDE / MIT COLLABORATION

Headlong: a microharness for persistent agents

Self-guided agents that think continuously

Introducing Headlong, an open source agent microharness featuring persistent agency. Your agent keeps thinking between external interactions in a self-guided loop inspired by human inner monologue. Headlong is a complete agent harness with a core of less than 10K lines of Bash, available on GitHub.

Asher
@ashe

I'm hiring a student researcher to work on RL and RLM-flavored things. DM me if interested

Aman @Amank1412 · 16h

Google Student Researcher Program 2026 is now OPEN!

Work on REAL AI/ML projects with:
• Google Research
• DeepMind...
Show more



2:12 PM · Feb 23, 2026 · **29.4K** Views

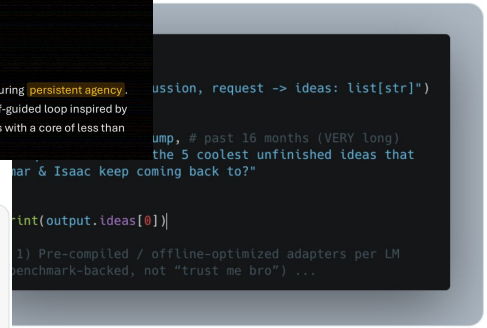
14 11 217 200

isaac
@isaacbmiller1

The dspy.RLM module is now released 👀

sage is plug-and-play with your existing

@lateinteraction and I figure out some



Jan 19, 2026 · **127.4K** Views

114 471 336

RLM Inference with Shared Program State

omes for free in programming systems using shared program state.

Γ CSAIL), **Logan Weber** (MIT CSAIL), **Tian Jin** (MIT CSAIL), **Michael Carbin** (MIT CSAIL)

of work have revealed revealed the benefits of marrying LLMs with programming uch as a Python REPL. Recursive Language Models (RLMs)[1] automates the work of prompt contexts using by enabling LLMs to self-decompose context prompts.Also, our **shared program state**[2] enables LLMs to modify the program state of a host program illed interface. The common pattern of using an LLM for automated state management s using shared program state can easily use RLMs.

In this blog post, we show how to enable the RLM inference strategy in only a few lines of code in a program with shared program state.

PrimeIntellect-ai/prime-agent Public

A self-improving RLM agent for coding workflows and long-running autonomous tasks.

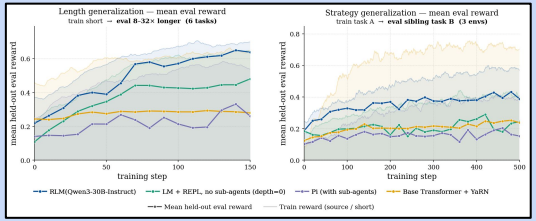
TypeScript ☆ 20.6k 2.3k

What I’m particularly interested in *now*

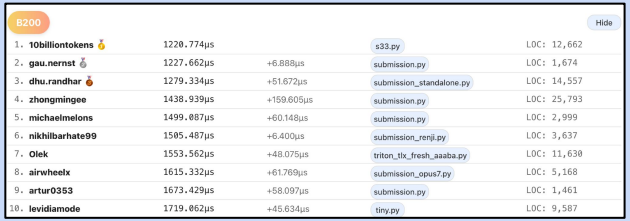
Micro-level view (*harness design choices*)

Base LM — Compaction — Compaction + RLM + “model shape” — ? — Compaction + RLM — RLM

Macro-level view **(scaling w.r.t. harnesses)**



how harnesses affect scaling on data / envs

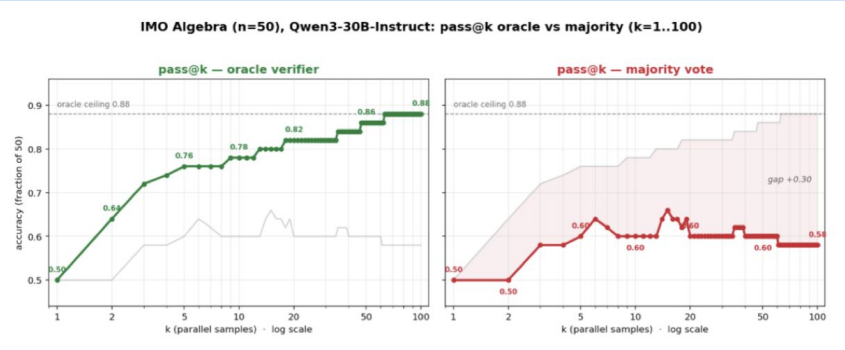


scaling laws for harnesses (token spend)

Are harnesses characterizable?

If you wanted some kind of predictable (e.g. inference scaling) laws for harnesses, how would you parameterize the class of harnesses we care about searching over?

Minimally, *as models get better*, what is the role of the harness and how do we characterize & study this?



with verification, does this just reduce to LLM monkeys?

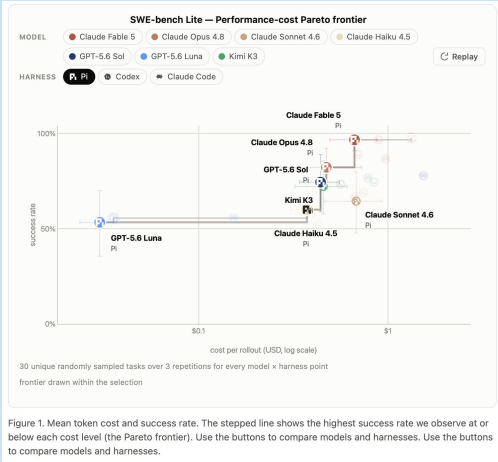


Figure 1. Mean token cost and success rate. The stepped line shows the highest success rate we observe at or below each cost level (the Pareto frontier). Use the buttons to compare models and harnesses. Use the buttons to compare models and harnesses.

What ultimately matters in a harness w.r.t. the model?
HarnessTax (Pan et al., 2026)

Blurring model-harness co-design.

Are there other tradeoffs with *different* input / output types that work better with harnesses? Do some harness changes become architectural changes?

```python
def pi(prompt):
    history = [prompt]

    while True:
        prompt = render(history, tools)
        reply = llm_call(prompt)
        history.append(reply)

        if not reply.tool_calls:
            return reply.text

        results = parallel_map(run_tool, reply.tool_calls)
        history.extend(results)
```

Should a language model always be, *densely*, string -> string?

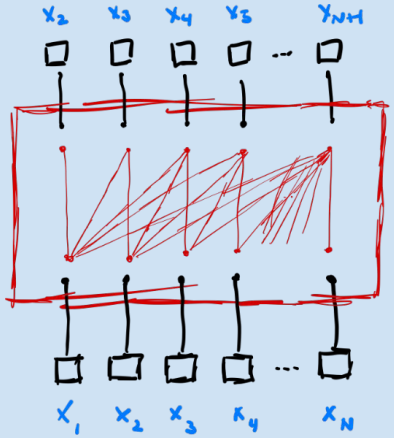


Jev (TypeSafe AI, 2026)

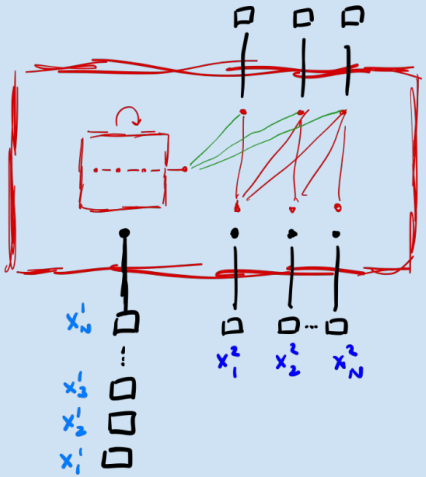
An neat recent example of constraining the output space, so an LM is string -> Dict[Literal, *noul*].

Blurring model-harness co-design: model shape

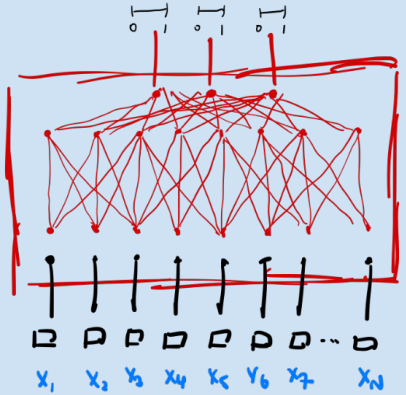
"Can language models have more *control* over what they see, and varying levels of granularity?"



decoder-only



half recurrent, half dense



non-constrained output

https://alexzhang13.github.io/blog/2026/shape/