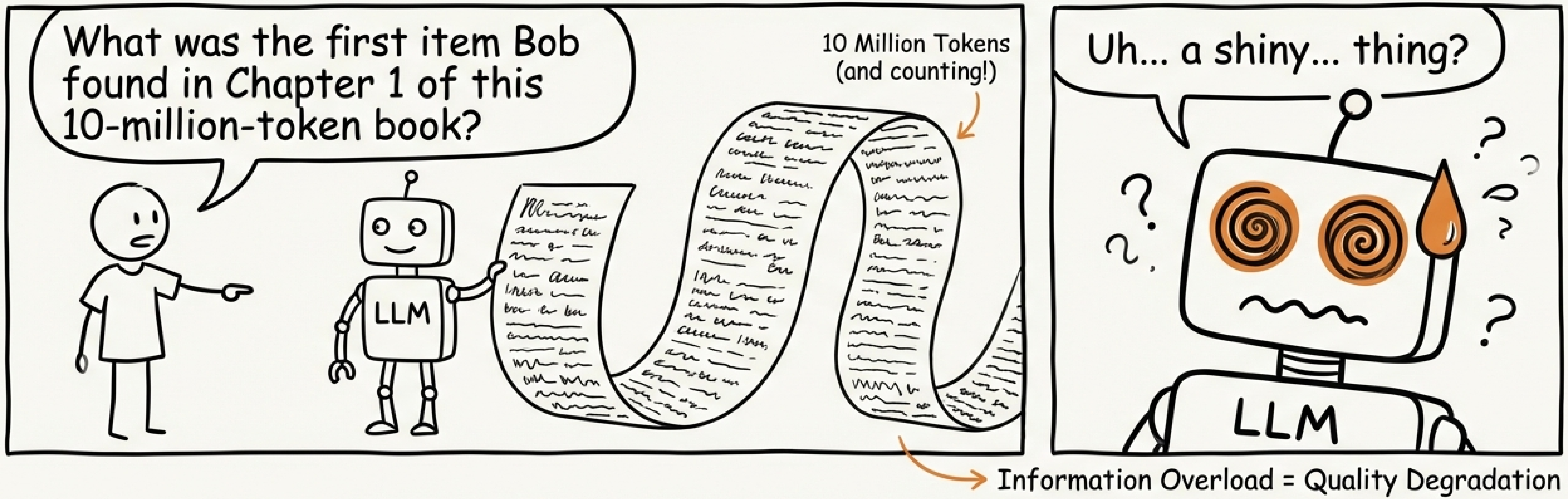
LLMs with a Library Card: An Introduction to Recursive Language Models

Based on the paper “Recursive Language Models” by Alex L. Zhang, Tim Kraska, and Omar Khattab (MIT CSAIL).

arXiv: 2512.24601v1

Modern LLMs suffer from "Context Rot." The longer the text, the worse they get.

As context gets longer, the quality of even frontier models like GPT-5 degrades quickly. This is a critical barrier for long-horizon tasks that require processing tens or hundreds of millions of tokens.



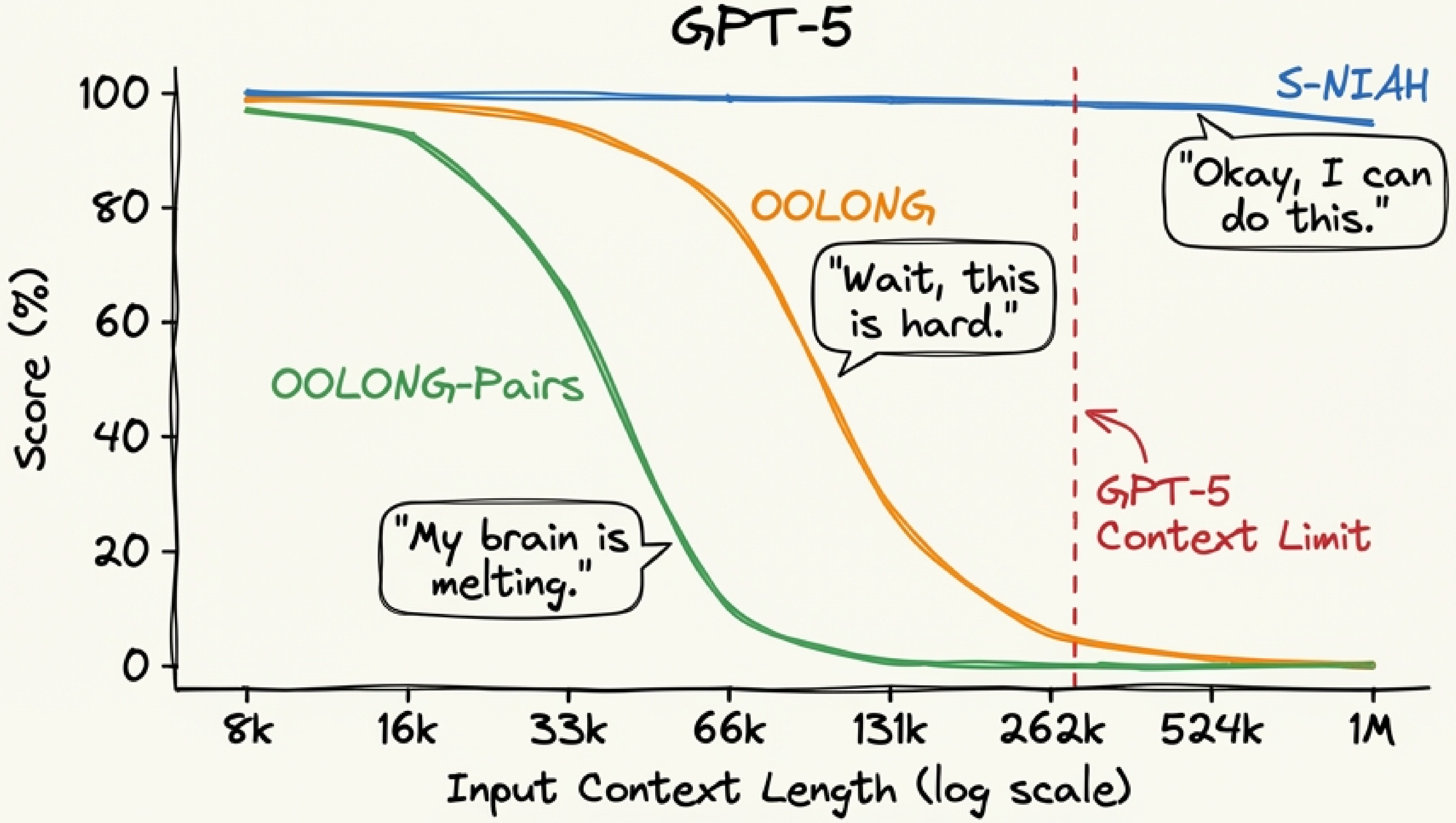
What was the first item Bob found in Chapter 1 of this 10-million-token book?
10 Million Tokens (and counting!)
LLM
Uh... a shiny... thing?
LLM
Information Overload = Quality Degradation

Fig. 1: The "Context Rot" Phenomenon in Action. Memory fade for long inputs.

NotebookLM

Performance doesn't just fade; it falls off a cliff.

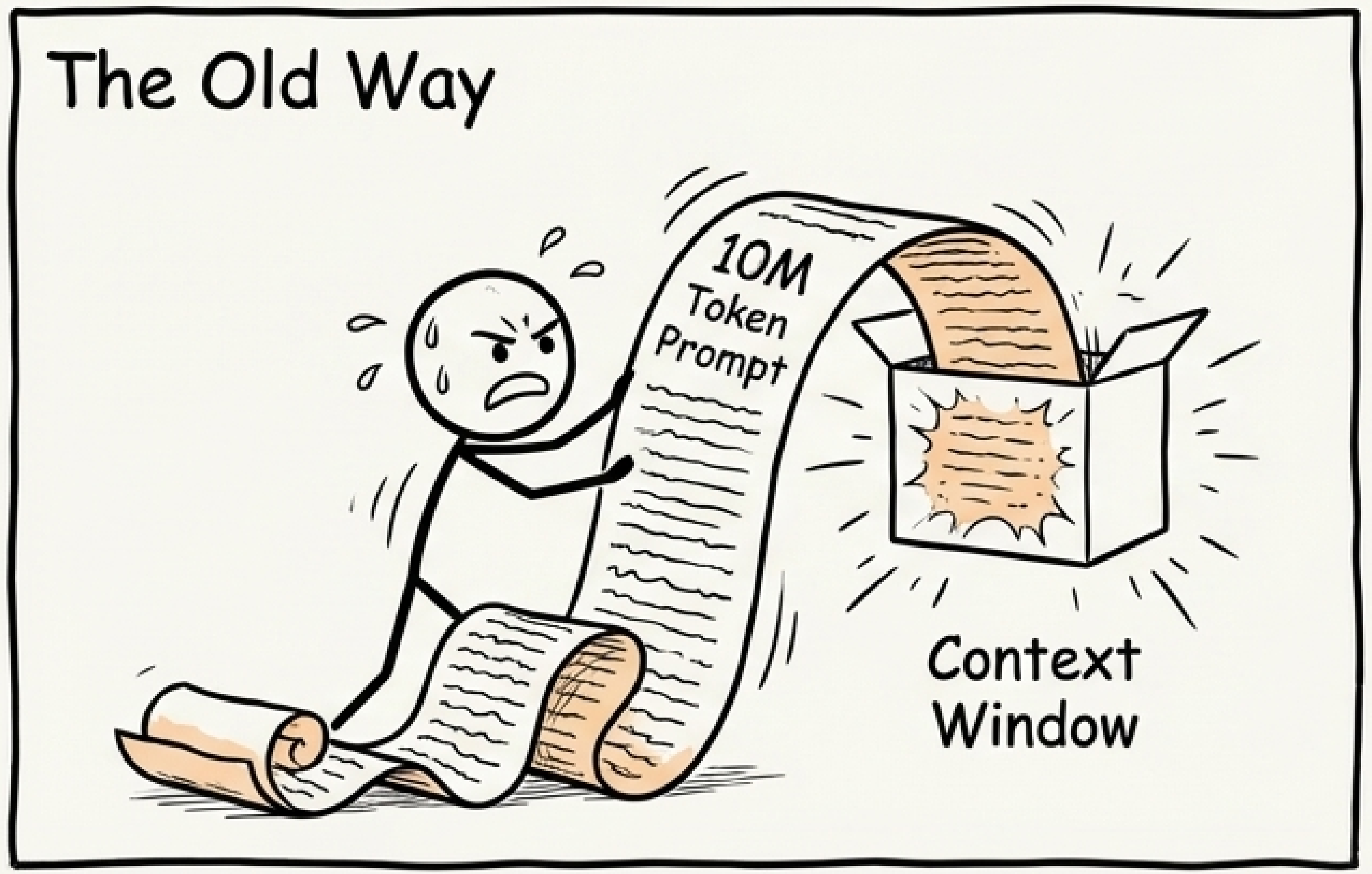
We tested GPT-5 on three long-context tasks of increasing complexity. The model's effective context window shrinks dramatically as the task gets harder.

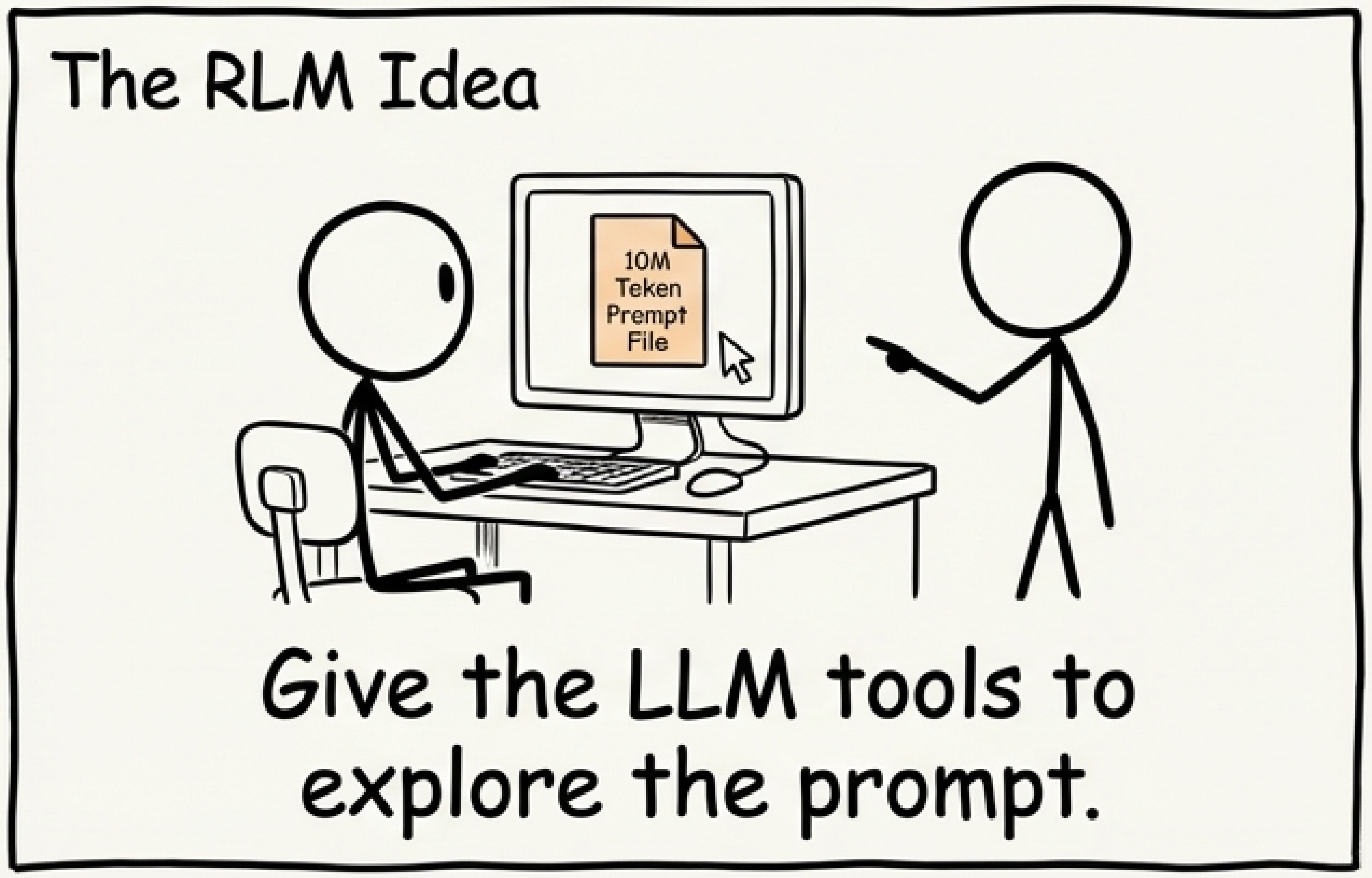


What if we stopped trying to cram everything into the LLM's brain?

The key insight: long prompts shouldn't be fed into the Transformer directly. Instead, treat the prompt as part of an external environment the LLM can symbolically interact

We were inspired by out-of-core algorithms, where systems with small memory process huge datasets by cleverly managing data fetching.



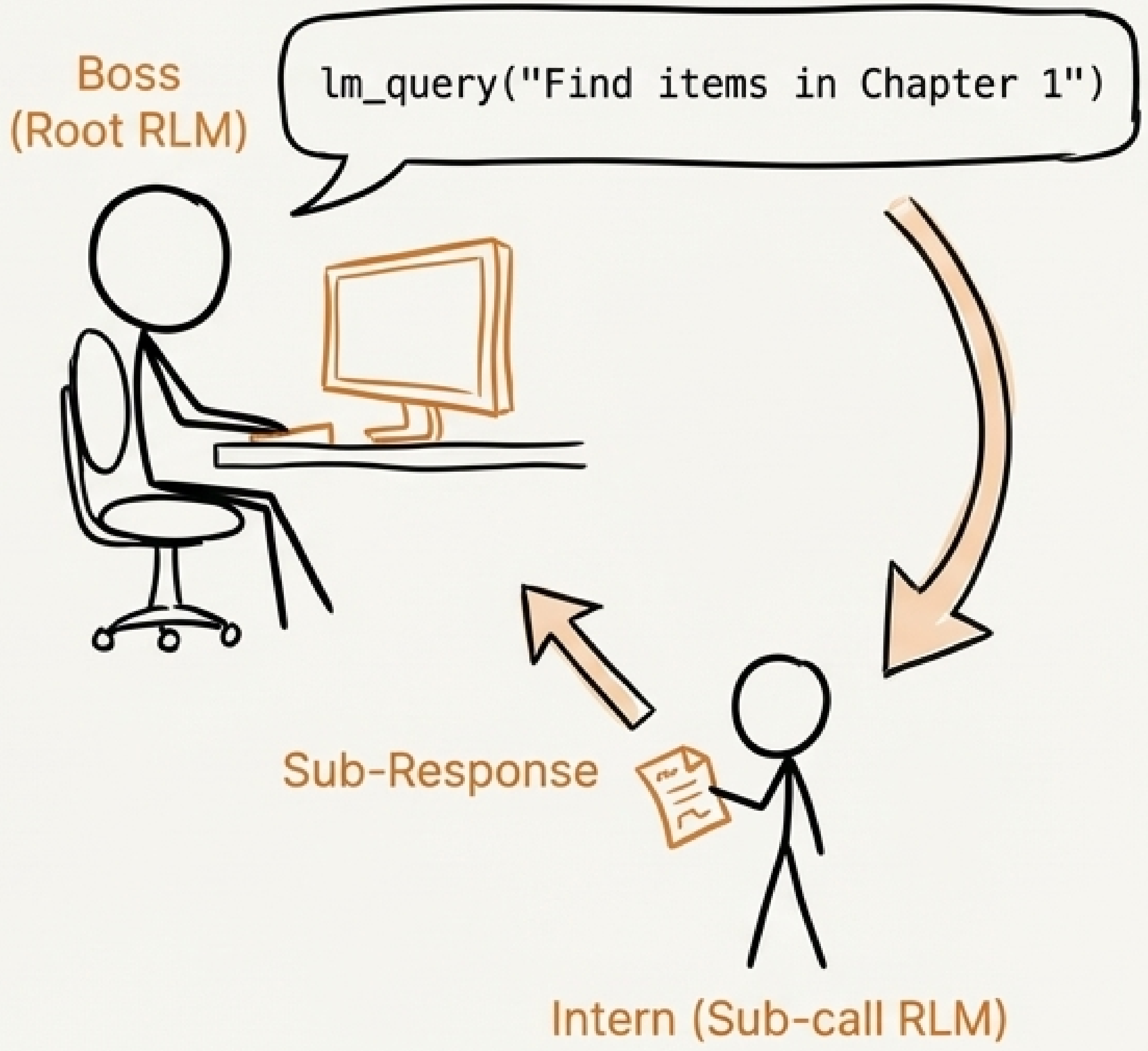


RLMs use a programming environment to explore the prompt and delegate tasks

The RLM places the entire prompt into a Python REPL environment as a variable.

From there, the LLM writes code to:

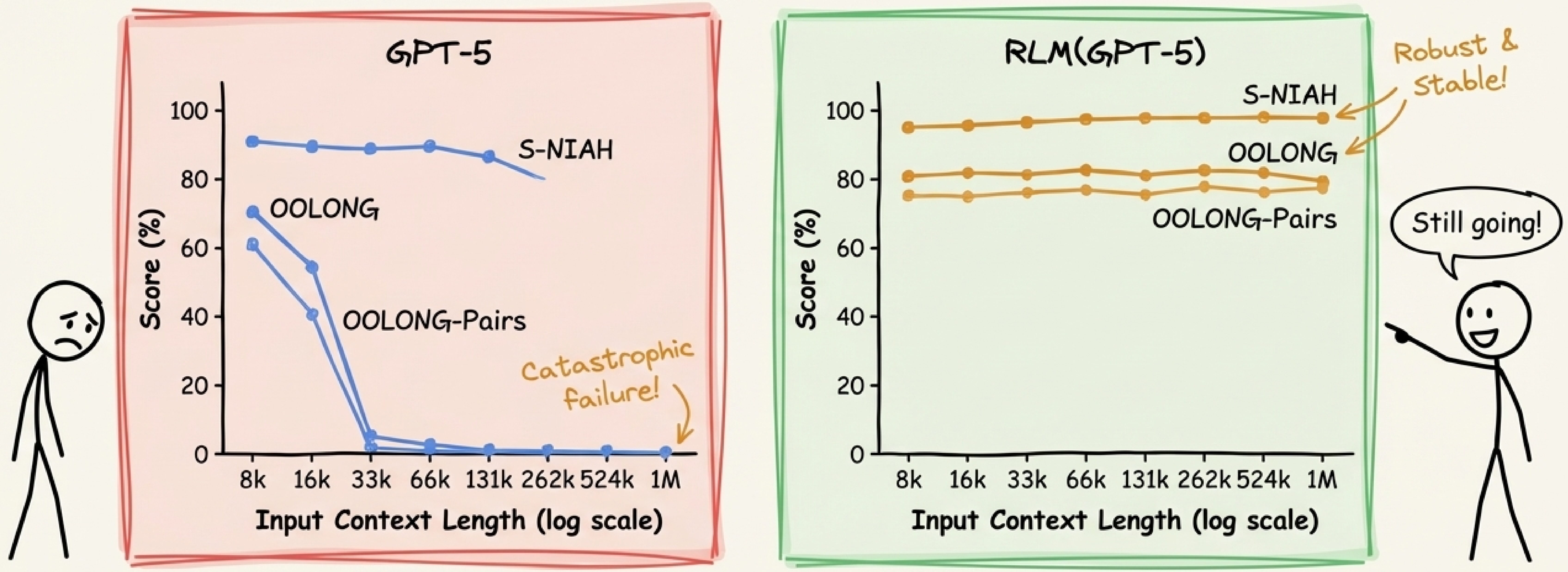
1. Peek into the prompt (`print(prompt[:100])`).
2. Decompose it into smaller pieces.
3. Recursively call itself (`lm_query(...)`) to solve sub-tasks on those pieces.



NotebookLM

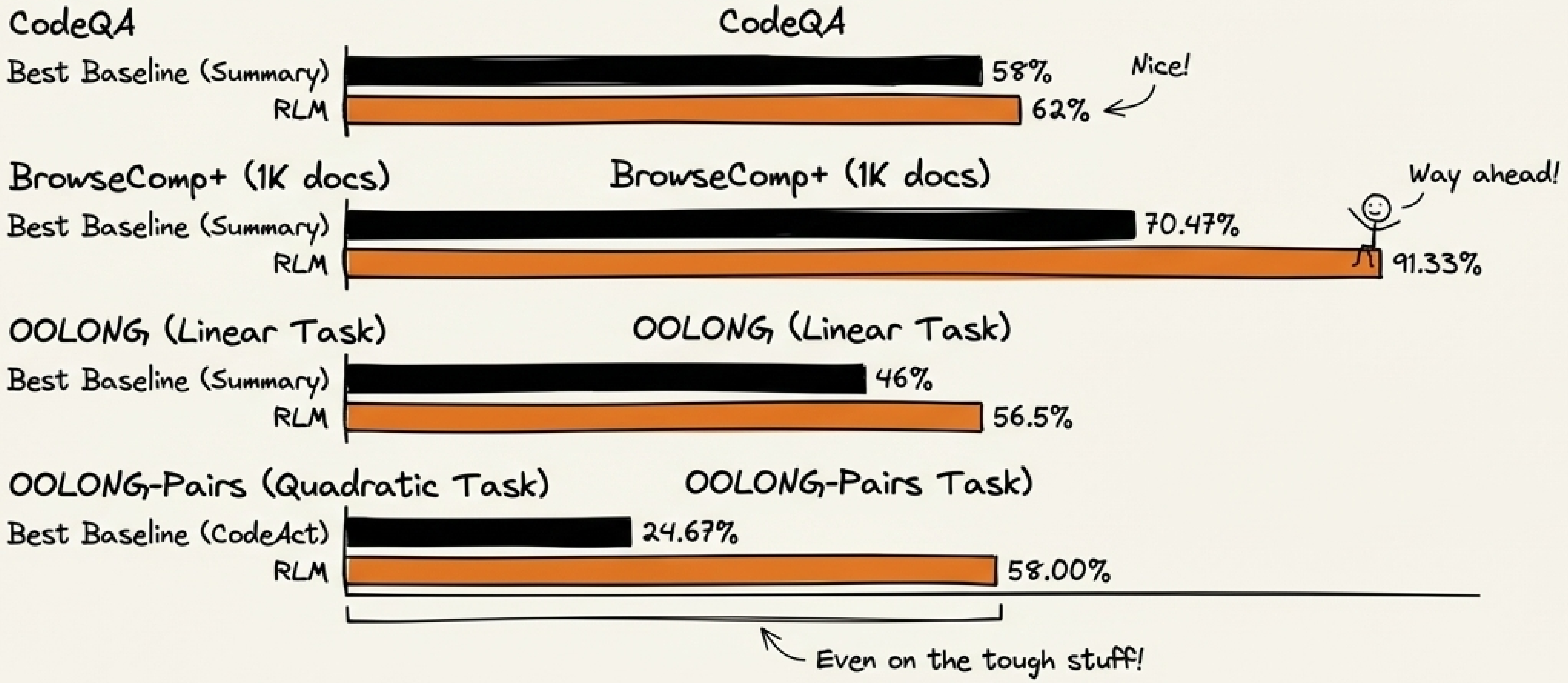
The result: RLMs maintain performance where base models catastrophically fail.

We ran the same scaling experiment on RLM(GPT-5). While the base model's performance collapsed, the RLM's performance remained stable and robust across all context lengths and complexities, even handling inputs up to two orders of magnitude beyond the base model's context window.



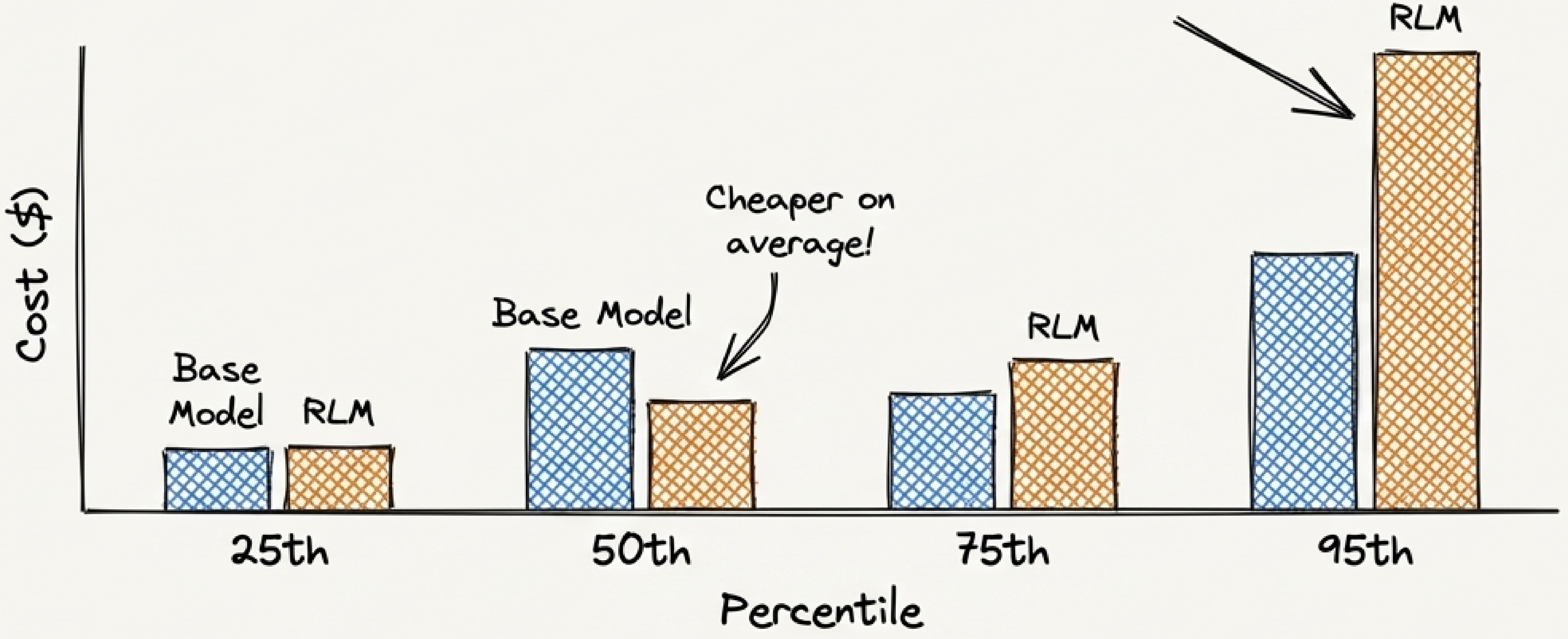
RLMs dominate across a diverse set of complex, long-context tasks.

We compared RLM(GPT-5) against strong baselines (including summary agents and CodeAct agents) on benchmarks for code understanding, multi-document QA, and information aggregation. RLMs consistently and significantly outperform all other approaches.



This power is surprisingly affordable—often comparable to, or even cheaper than, a base model call.

Because RLMs can selectively view context instead of ingesting everything, their median cost is often lower than calling the base model directly. However, costs are high-variance, as the RLM can sometimes get curious and run many steps to find an answer.

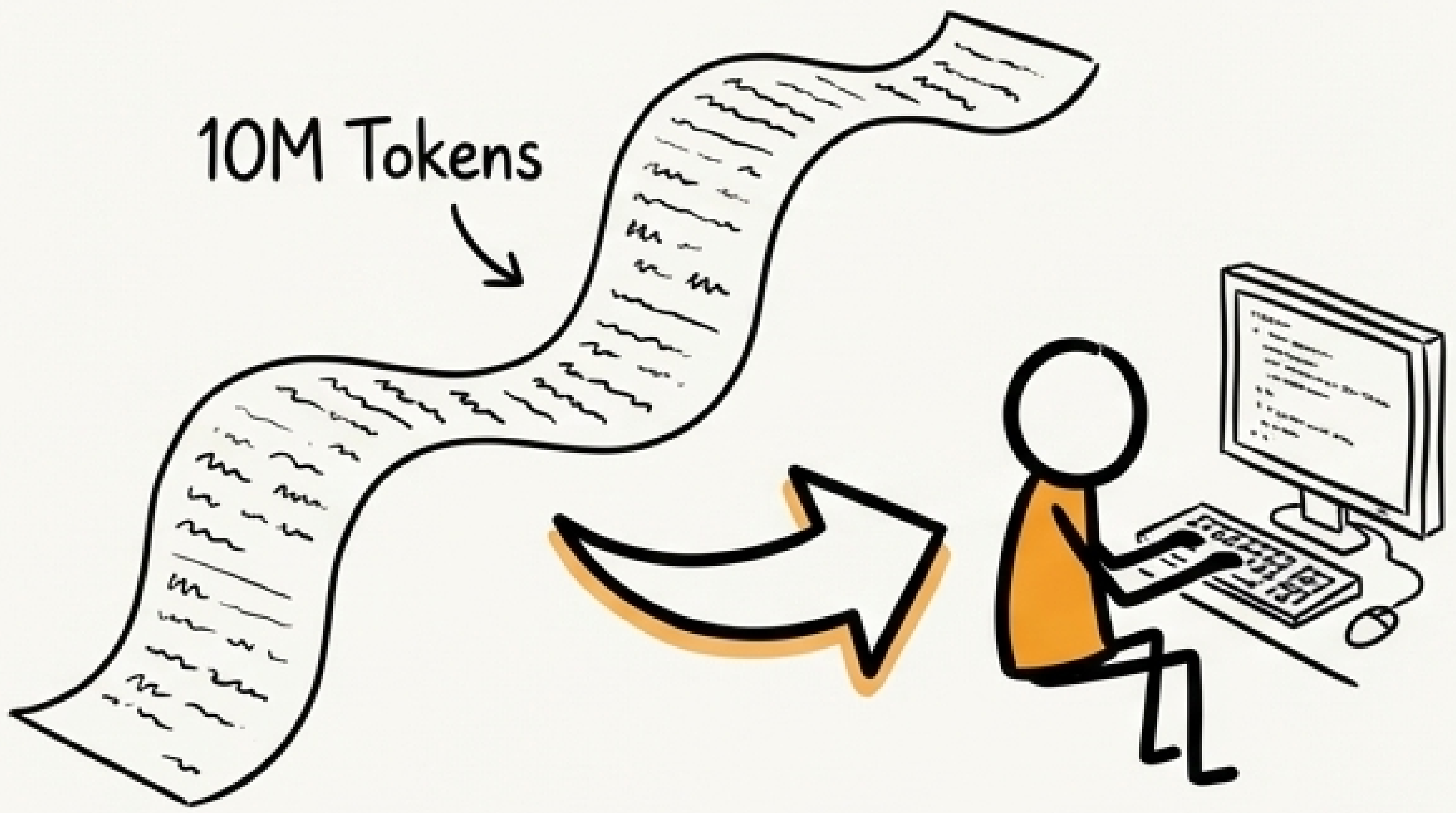


Though sometimes the RLM decides to spend the afternoon exploring the data and runs up a bit of a tab.

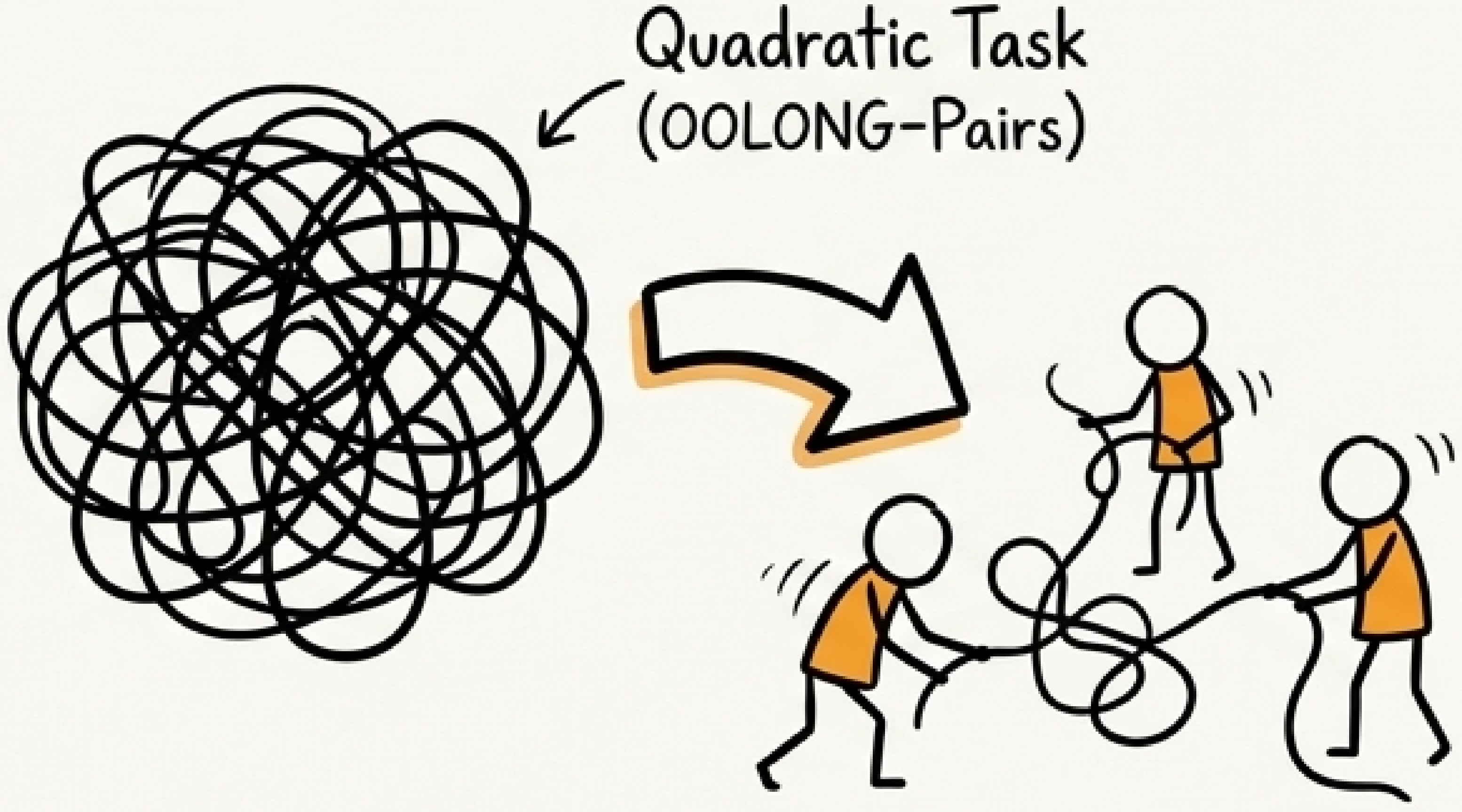
NotebookLM

The REPL handles length; recursive calls conquer complexity.

The two key features of RLMs solve two different problems. The REPL environment is what allows the model to access inputs of arbitrary length. Recursive sub-calling is what allows it to decompose and solve information-dense problems that would otherwise be intractable.



The REPL lets us *access* all of this.



Recursion lets us *untangle* this.

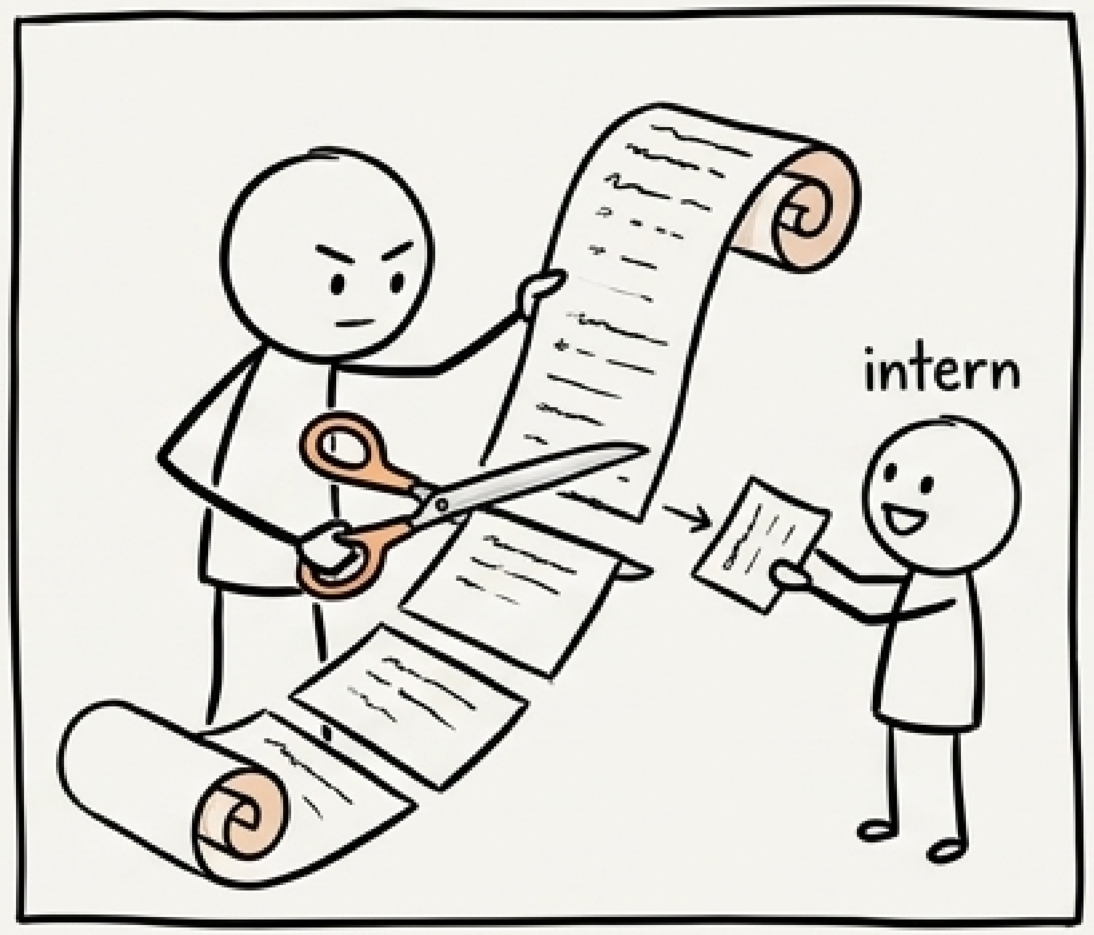
NotebookLM

Without being told, RLMs learn to think like a research assistant.

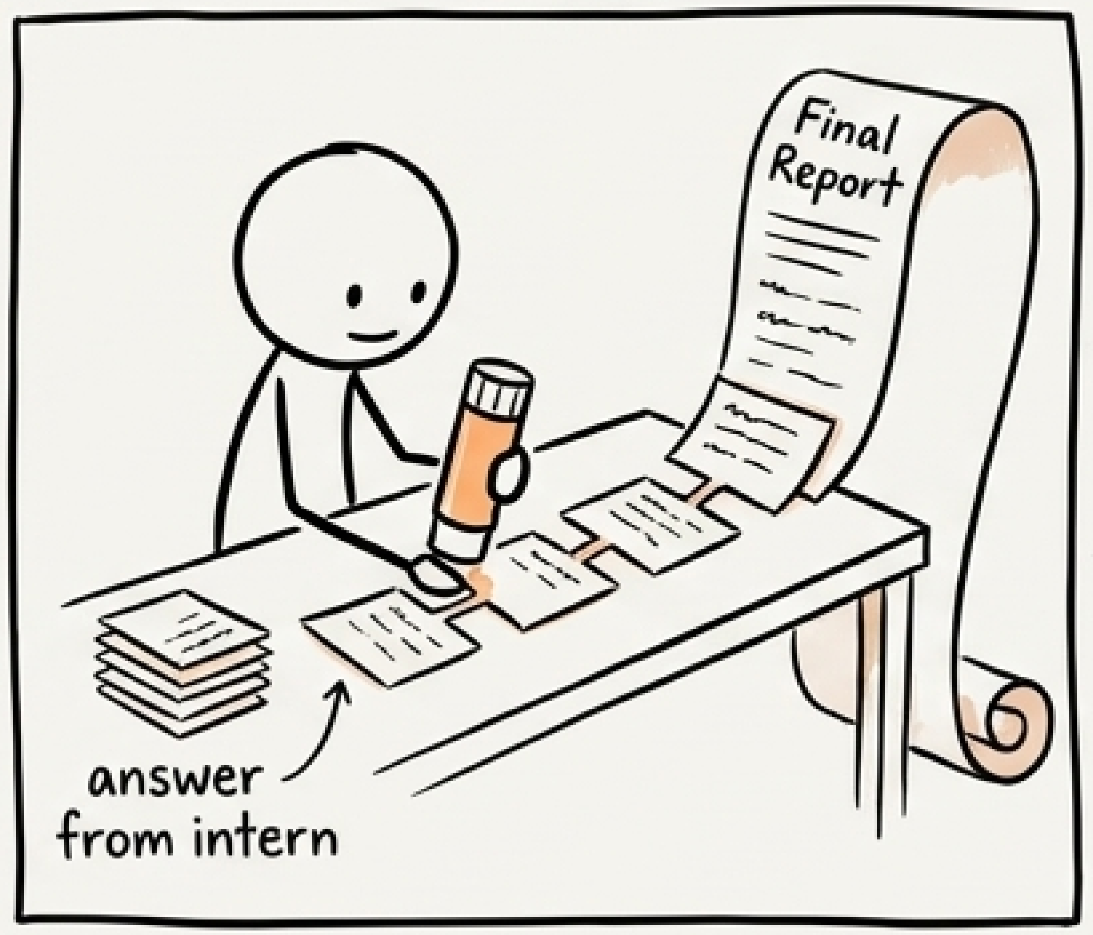
We observed common, sophisticated patterns in RLM trajectories. The model learns to manage its context and decompose problems effectively.

Filtering

Chunking

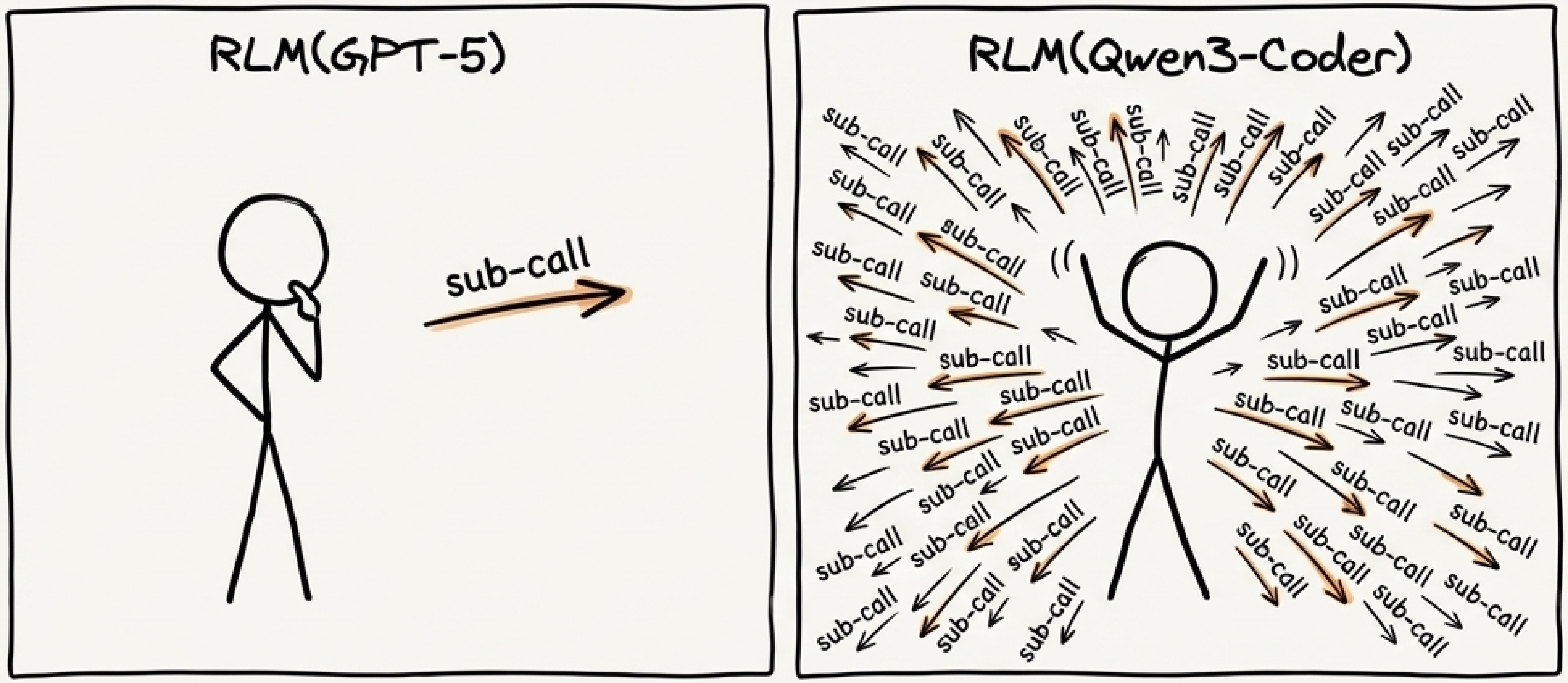


Stitching



The base model's personality shines through.

Different base models exhibit different behaviors as RLMs. RLM(GPT-5) tends to be conservative and plans carefully, while RLM(Qwen3-Coder) is far more liberal with its use of sub-calls.



We literally had to add a line to the RLM prompt for Qwen3-Coder warning it "not to use too many sub-calls."

It's not perfect... yet.

New Issue

🐞 `[BUG] Slow due to sequential calls.`

Our current implementation uses blocking calls. This can be significantly improved with asynchrony.

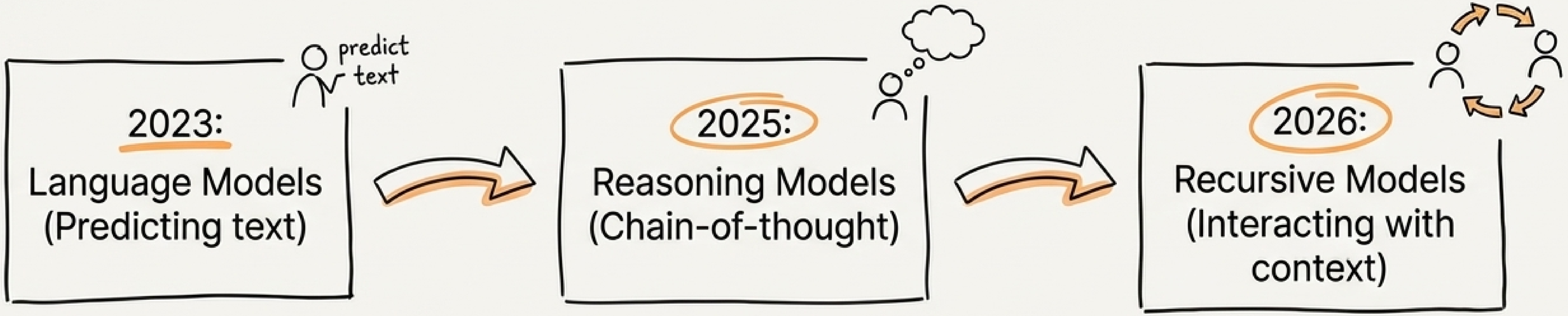
🐞 `[BUG] Sometimes gets confused about when it’s ‘done’.`

The model occasionally tries to output its plan as the final answer instead of executing it.

🐞 `[FEATURE] Train models specifically to be RLMs.`

Current models are inefficient decision makers. Training them on RLM trajectories could provide huge performance gains.

“Much like the switch in 2025 from language models to reasoning models, we think 2026 will be all about the switch to Recursive Language Models.”


predict text
2023:
Language Models (Predicting text)
2025:
Reasoning Models (Chain-of-thought)
2026:
Recursive Models (Interacting with context)

The Takeaway: LLMs with tools are fundamentally more powerful.

- ✓ **The Problem:** LLMs suffer from “Context Rot,” failing on long and complex inputs.
- ✓ **The Solution:** RLM treats the prompt as an external environment, allowing the LLM to explore it with code and recursive self-calls.
- ✓ **The Result:** Orders-of-magnitude scaling in context, superior performance on complex tasks, and comparable (or lower) cost.

Read the Full Paper

Recursive Language Models
Alex L. Zhang, Tim Kraska, Omar Khattab
MIT Computer Science & Artificial Intelligence Laboratory

arxiv.org/pdf/2512.24601