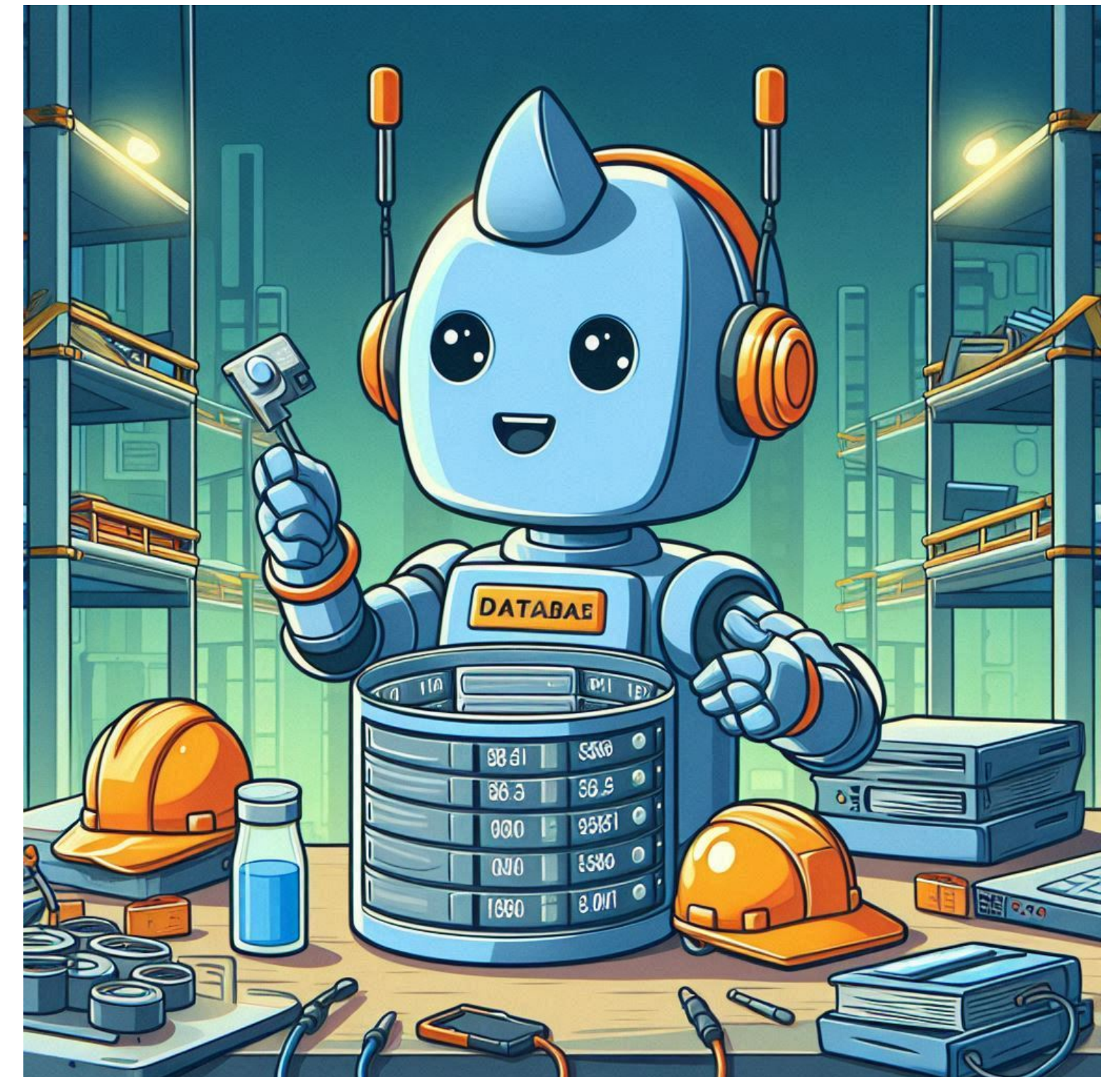


A DBMS without (human) programmers

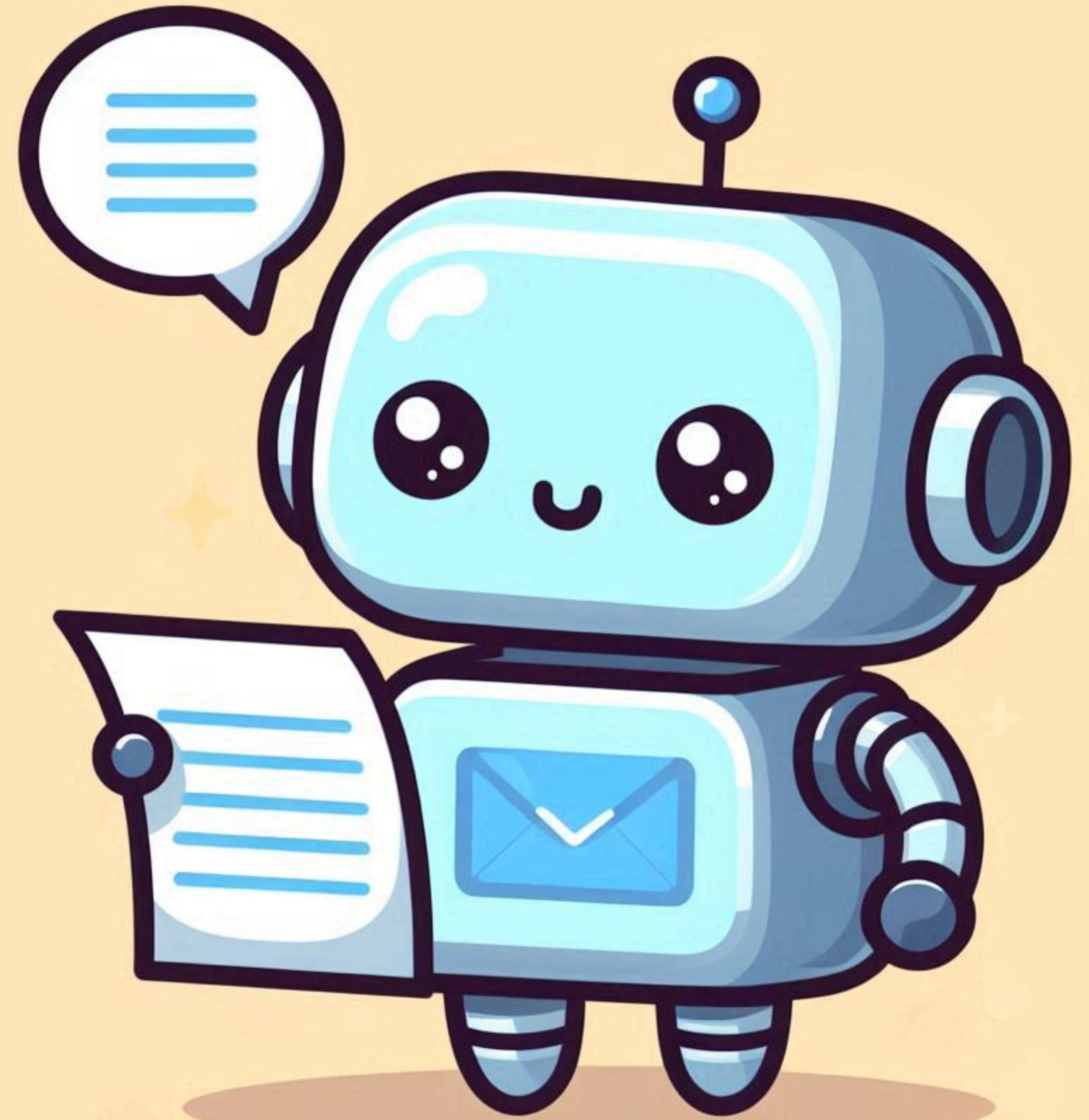
By machines, for machines

Michael Cahill, University of Sydney, September 2024



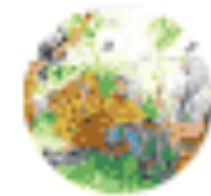
AI code generation

- Prompts and responses can be natural language, code, or a mixture of the two.
- Each system has a **limited amount of context**.
- If the prompt is larger than the context size, only a sliding window is accessible to the model as it generates the response.
- **Results are non-deterministic.**
- Training sets are huge and some include open source DBMS implementations.



Auto-correct on steroids?

Andrej Karpathy, ex-director of AI at Tesla:



Andrej Karpathy ✓

@karpathy

...

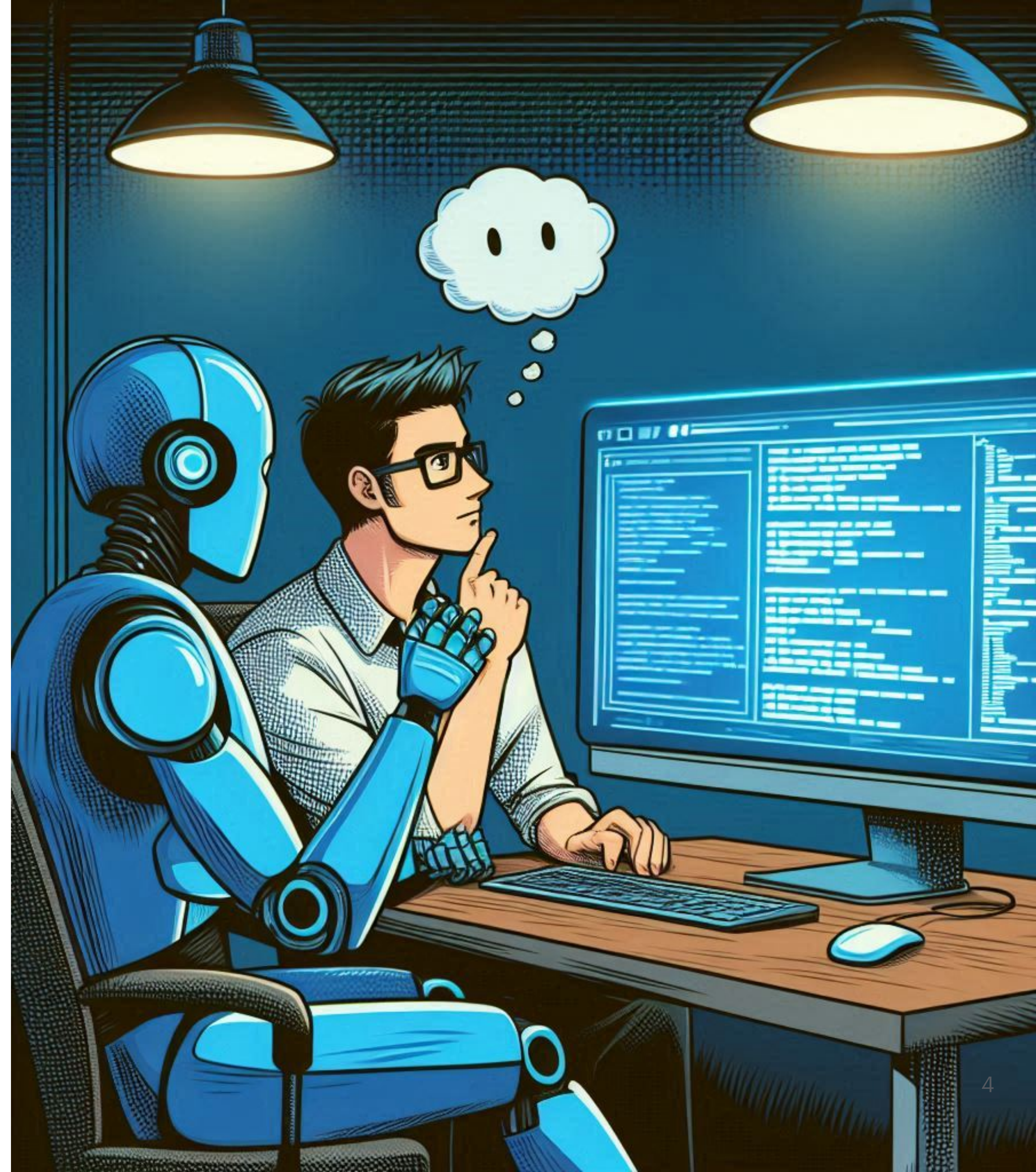
automating software engineering

In my mind, automating software engineering will look similar to automating driving. E.g. in self-driving the progression of increasing autonomy and higher abstraction looks something like:

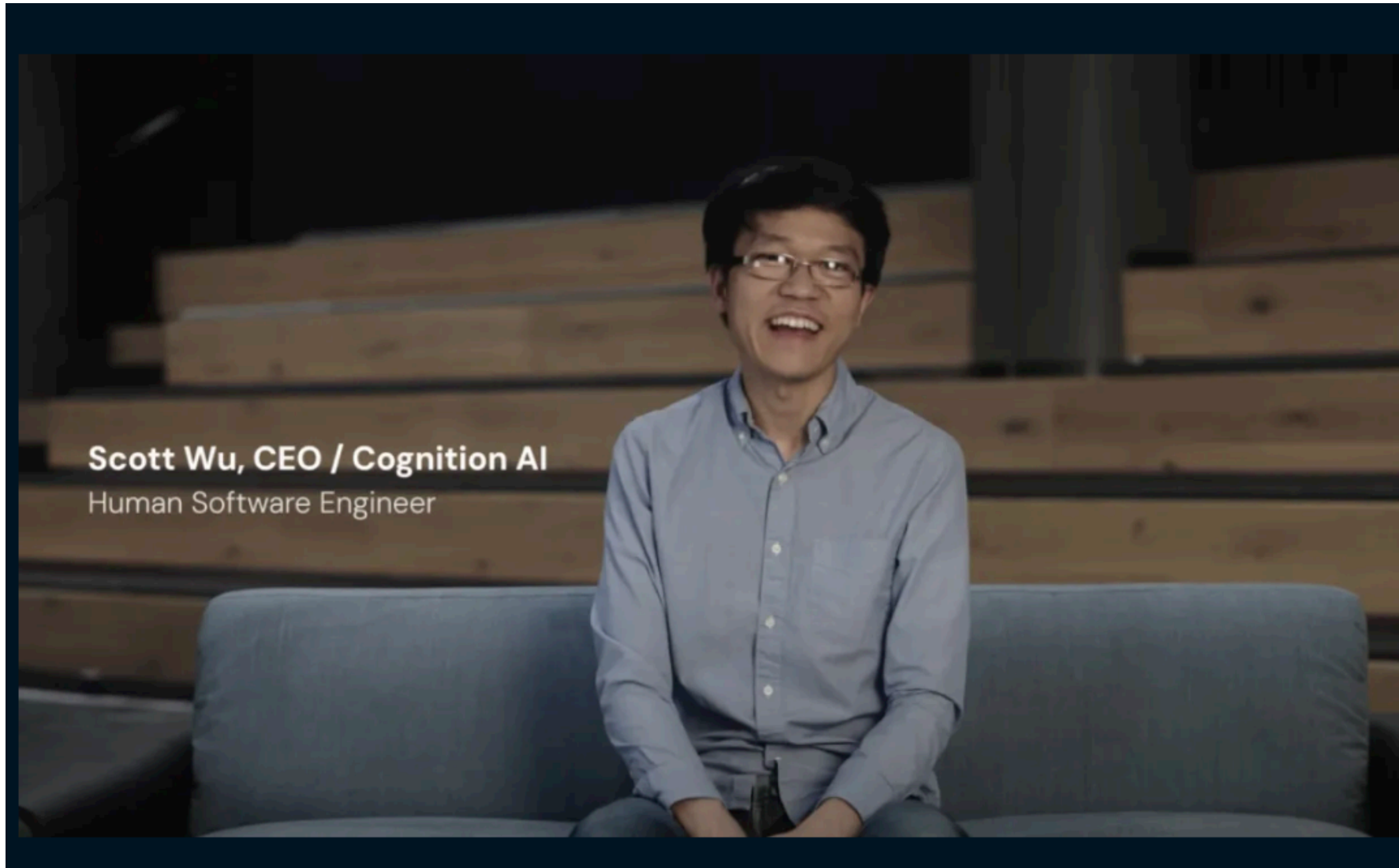
1. first the human performs all driving actions manually
2. then the AI helps keep the lane
3. then it slows for the car ahead
4. then it also does lane changes and takes forks
5. then it also stops at signs/lights and takes turns
6. eventually you take a feature complete solution and grind on the quality until you achieve full self-driving.

Today: pair programming

- Add a chat interface to development environments
- Programmers can ask “how do I?” questions
- The IDE provides context to the model (usually from the active source file)
- Responses inform the programmer
- Often: copy / paste



AI programmers for hire



Introducing Devin, the first AI software engineer

March 12, 2024 • by Scott Wu

Setting a new state of the art on the SWE-bench coding benchmark. Meet Devin, the world's first fully autonomous AI software engineer.

On the other hand hand...

News / Technology

AI worse than humans in every way at summarising information, government trial finds

A test of AI for Australia's corporate regulator found that the technology might actually make more work for people, not less.

CAM WILSON SEP 03, 2024

 [Share](#)

Why try to build a DBMS with AI?

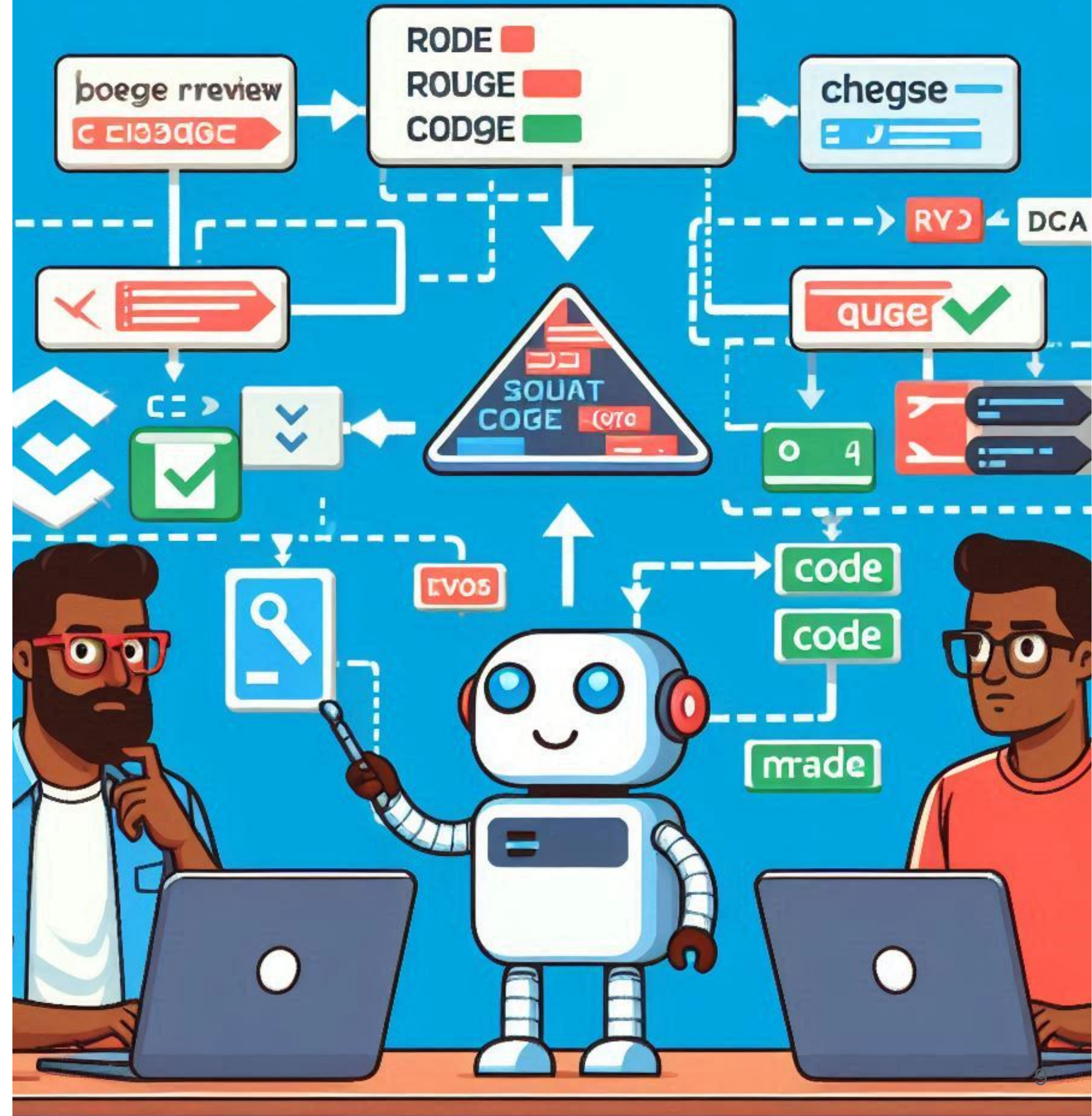
- Can non-experts build something that used to be the domain of a few?
 - Do LLMs lower the bar?
- Key question: what is good code?
 - Focusing on a domain where we have experience
 - There are objective tests for DBMS correctness and performance
 - Less so for the code itself
- New lens for thinking about software engineering

What do SWEs actually do?

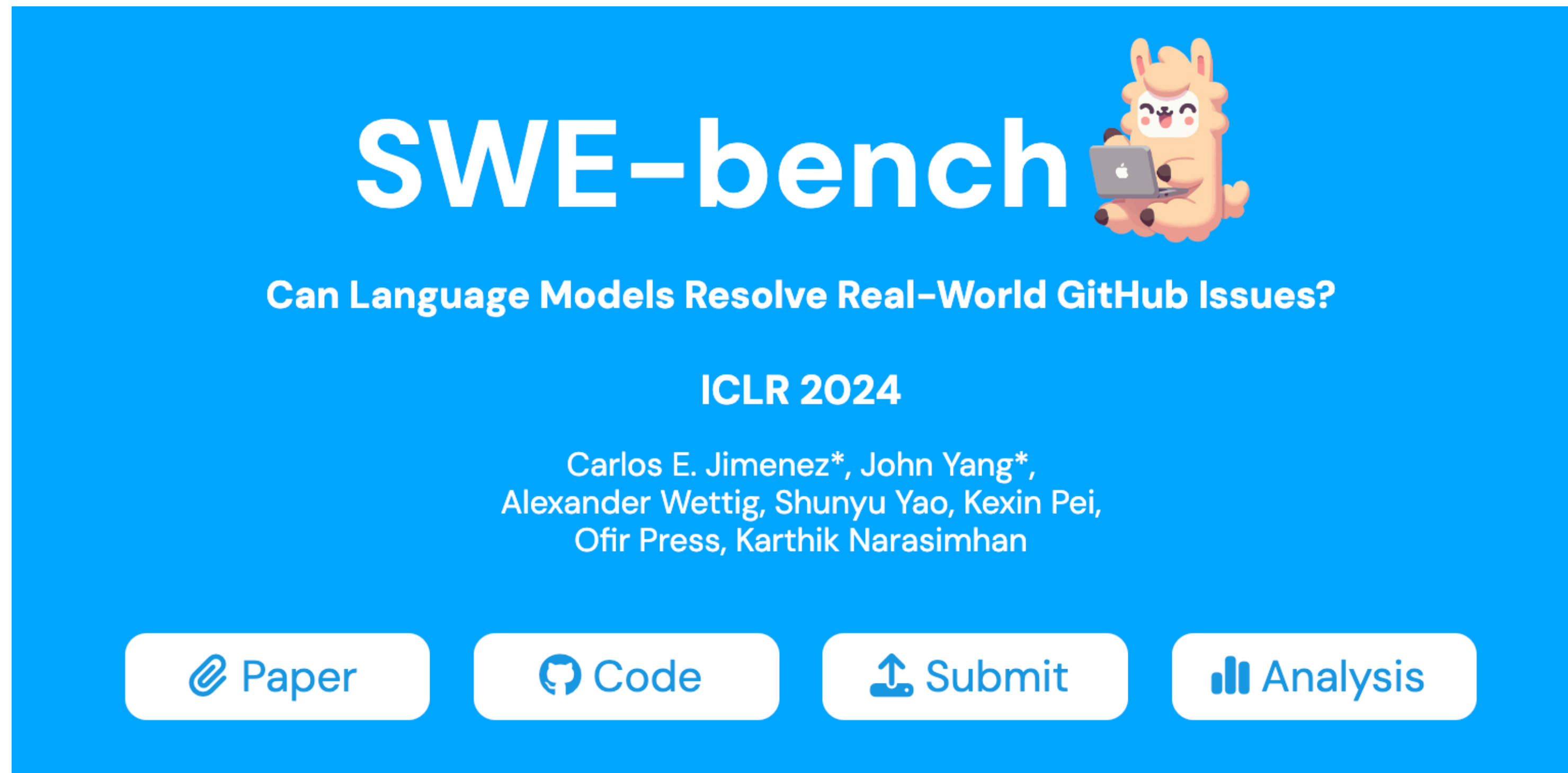
- A lot more than turning a specification into code!
- Code reviews
- Debugging
- Tuning
- Mapping new features onto a system
 - While maintaining compatibility and online upgrades
- Refactoring / redesigning / re-architecting


Tasks to prompts

- Code review:
 - 3 different angles + decider
- Debugging:
 - Generate a test that demonstrates the problem
 - Propose a code / prompt change
 - Repeat until all tests pass; and
 - Change accepted by code review



Can language models fix bugs?



SWE-bench 

Can Language Models Resolve Real-World GitHub Issues?

ICLR 2024

Carlos E. Jimenez*, John Yang*,
Alexander Wettig, Shunyu Yao, Kexin Pei,
Ofir Press, Karthik Narasimhan

[Paper](#) [Code](#) [Submit](#) [Analysis](#)

Our evaluations show that both state-of-the-art proprietary models and our fine-tuned model SWE-Llama can resolve only the simplest issues. The best-performing model, Claude 2, is able to solve a mere 1.96% of the issues. Advances on SWE-bench represent steps towards LMs that are more practical, intelligent, and autonomous.

Can language models fix bugs?

Leaderboard

Lite	Verified	Full			
Model	% Resolved	Date	Logs	Trajs	Site
🏆 Gru(2024-08-24)	45.20	2024-08-24	🔗	🔗	🔗
🥈 Honeycomb	40.60	2024-08-20	🔗	🔗	🔗
🥉 Amazon Q Developer Agent (v20240719-dev)	38.80	2024-07-21	🔗	🔗	🔗
AutoCodeRover (v20240620) + GPT 4o (2024-05-13)	38.40	2024-06-28	🔗	-	🔗
Factory Code Droid	37.00	2024-06-17	🔗	-	🔗

Current state

- Starting with a K/V store
 - Scripts call model to generate code
 - Template language for prompts
 - C for now just for header separation
- Generating tests and implementation
- Non-determinism, retries required
 - Human-in-the-loop for now

≡ system.prompt

```
1 You are an expert C programmer.  
2 You like to write clear, complete,  
3 portable code that is well commented.
```

≡ skiplist.prompt

```
1 Implement a skiplist for bytestring key/value  
2 pairs, where the keys are comparable with  
3 memcmp. The skiplist should support insert and  
4 remove operations.
```

≡ synthdb-test.prompt

```
1 Write some code to test the skiplist  
2 and cursor interfaces given below.  
  
8 The skiplist interface, called  
9 `skiplist.h` is given here:  
10 ```c  
11 #include raw 'skiplist.h'
```

Enforcing code constraints

- In any complex code base, humans use guard rails for sanity checking
 - e.g., “always check the return value of `malloc`”
 - or, “every latch acquire should have a matching release”
- How well can these be enforced by LLMs?
 - cf static analysis tools, false positives, etc.
 - much simpler to express than writing a tool to walk an AST

Haven't we seen this before?

1. Outsourcing

- ... with an immediate feedback loop

2. Code generation

- ... great for automating repetitive tasks, can be hard to generalize

3. Refining a formal specification into an implementation

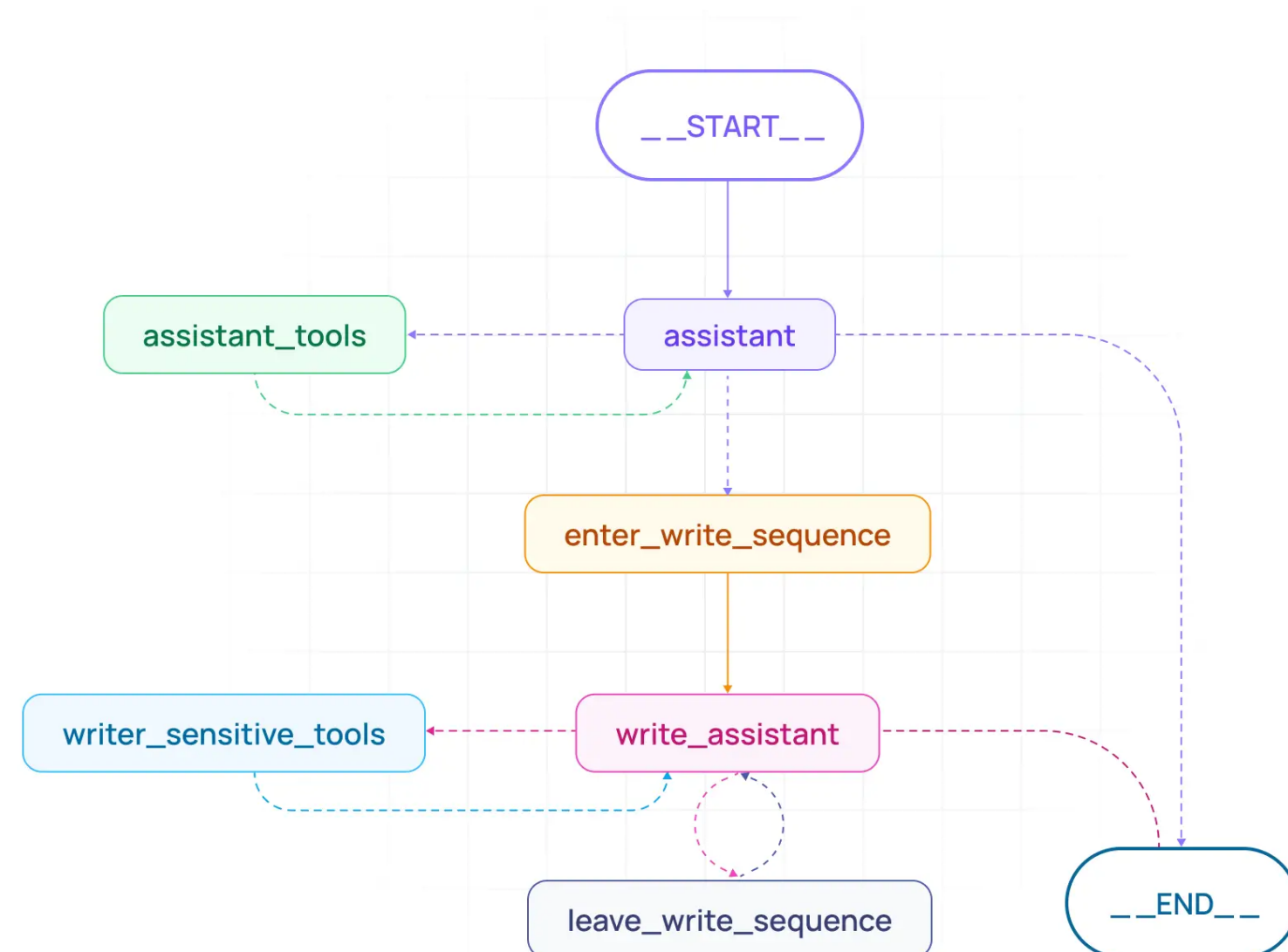
- ... natural language is informal but can be understood by more people
- ... here, we need to tame non-determinism

Taming non-determinism

- We have processes to cope with variation among programmers
- Code review process can propose alternatives and choose the best
- Test-driven development
 - But who makes sure the tests are valid and comprehensive?
- Get a(nother) model to explain generated code
 - Iterate until the explanation “matches” the prompt

Non-determinism and quality

- Building strong properties from fallible components is a classic database problem
- Lots of support from the community building on LLMs, e.g., LangGraph



Controllable cognitive architecture for any task

LangGraph's flexible API supports diverse control flows – single agent, multi-agent, hierarchical, sequential – and robustly handles realistic, complex scenarios.

Ensure reliability with easy-to-add moderation and quality loops that prevent agents from veering off course.

Open questions

- Can this produce a functioning DBMS?
- How does context size constrain code structure?
- Effort compared to human-built DBMS to:
 - Debug?
 - Extend / improve?
 - Reconfigure?
- “Clean room” possible?
 - existing DBMS -> description -> new DBMS
- If applications are also generated, what kinds of interfaces work best?

