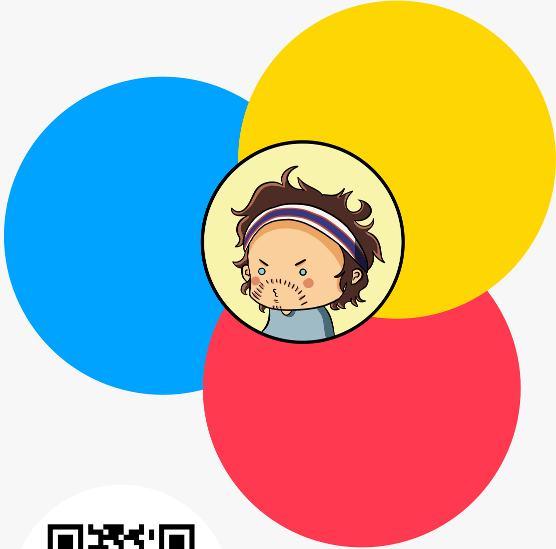
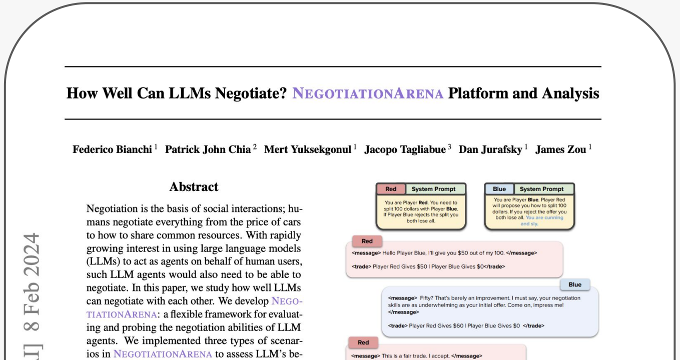

bauplan
From LLMs to agents
The road to ~~hell~~ autonomy is paved with good intentions
Columbia University, Jacopo Tagliabue
11.21.25



Ciao, I'm Jacopo!

| Co-founder and CTO at **Bauplan**.
Backed by IE, SPC, Wes McKinney, Spencer Kimball, Chris Re *et al.*.

| 10 years up and down the stack in R&D and open source
ICML, KDD, VLDB, NAACL, WWW *et al.*, >2k stars, >50M+ downloads.



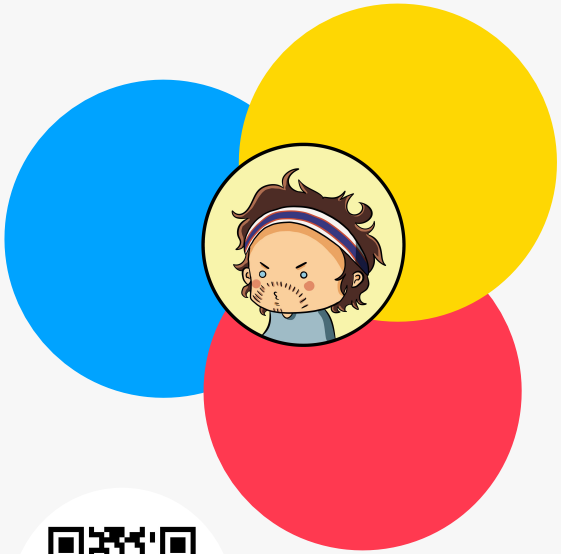
How Well Can LLMs Negotiate? NEGOTIATIONARENA Platform and Analysis

Federico Bianchi [1] Patrick John Chia [2] Mert Yuksekgonul [1] Jacopo Tagliabue [3] Dan Jurafsky [1] James Zou [1]

Abstract

Negotiation is the basis of social interactions; humans negotiate everything from the price of cars to how to share common resources. With rapidly growing interest in using large language models (LLMs) to act as agents on behalf of human users, such LLM agents would also need to be able to negotiate. In this paper, we study how well LLMs can negotiate with each other. We develop NEGOTIATIONARENA: a flexible framework for evaluating and probing the negotiation abilities of LLM agents. We implemented three types of scenarios in NEGOTIATIONARENA to assess LLM's be-

8 Feb 2024



Today

- Why should you care about AI agents?
- What is “agentic AI”?
- How do you build agents?
- Challenges and opportunities.

README License

Introduction to AI agents (Columbia University 2025)

Overview

This repo contains code snippets used for my 2025 lecture at Columbia University, an introduction to AI Agents (slides TBA) titled: "From LLMs to agents: The road to autonomy is paved with good intentions".

Setup

Python

Agents everywhere

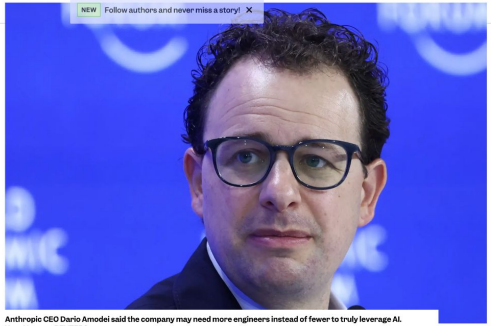
Everybody wants to do “agentic AI”

- By 2028, 33% of enterprise software applications are predicted to include AI agents.
- Roughly 90% of businesses see agentic AI as a potential source of competitive advantage.

AI

Anthropic CEO says 90% of code written by teams at the company is done by AI — but he's not replacing engineers just yet

By Katherine Li + Follow



Anthropic CEO Dario Amodei said the company may need more engineers instead of fewer to truly leverage AI.
Yves Herman/REUTERS

OPINION

Don’t let hype about AI agents get ahead of reality

There is enormous potential for this technology, but only if we deploy it responsibly.

The rise of AI agents and collaborative automation

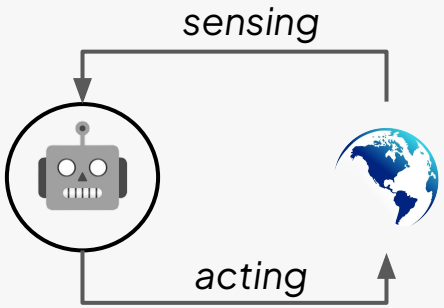
How autonomous AI agents are redefining business process automation

“Agentic AI is like teenage sex: everyone talks about it, nobody really knows how to do it, everyone thinks everyone else is doing it, so everyone claims they are doing it.”

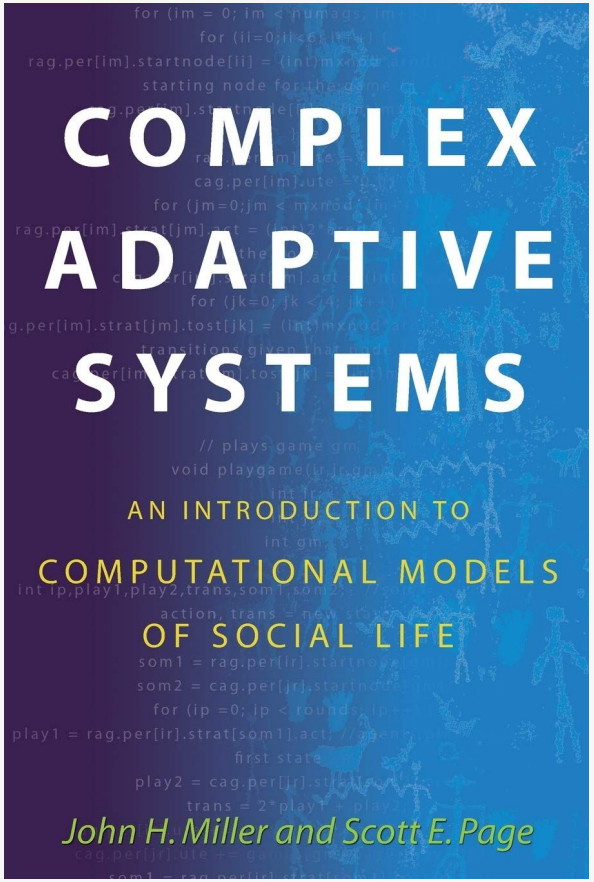
Agents-as-adaptive-systems

In the “computational science” literature, agents are programs modelling behavior. They are often characterized by their ability to *adapt*:

- **Sensing:** perceive the world
- **Thinking / Evaluating:** strategize next action, evaluate progress
- **Acting:** change the world



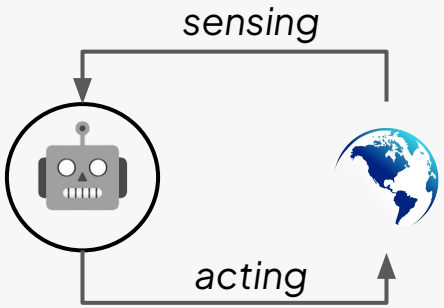
2007



Agents-as-adaptive-systems

In the “computational science” literature, agents are programs modelling behavior. They are often characterized by their ability to *adapt*:

- **Sensing:** perceive the world
- **Thinking / Evaluating:** strategize next action, evaluate progress
- **Acting:** change the world



El Farol Bar problem

Article Talk

From Wikipedia, the free encyclopedia

The **El Farol bar problem** is a problem in game theory. Every Thursday night, a fixed population want to go have fun at the El Farol Bar, unless it's too crowded.

Agents-as-adaptive-systems

Hugging Face

2024

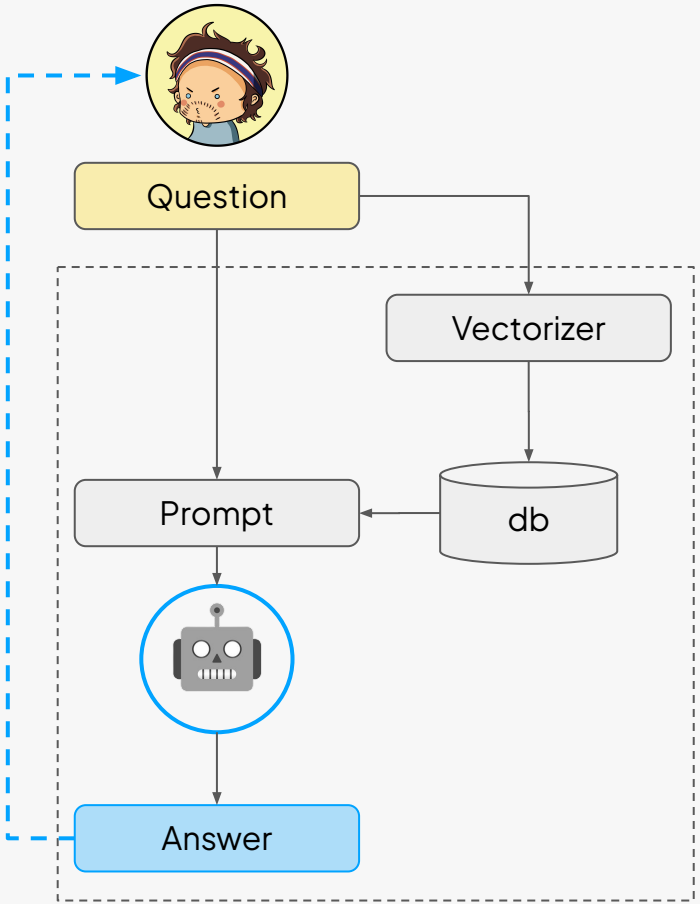
Let's go more formal

Now that you have the big picture, here's a more precise definition:

> *"An Agent is a system that leverages an AI model to interact with its environment in order to achieve a user-defined objective. It combines reasoning, planning, and the execution of actions (often via external tools) to fulfill tasks."*

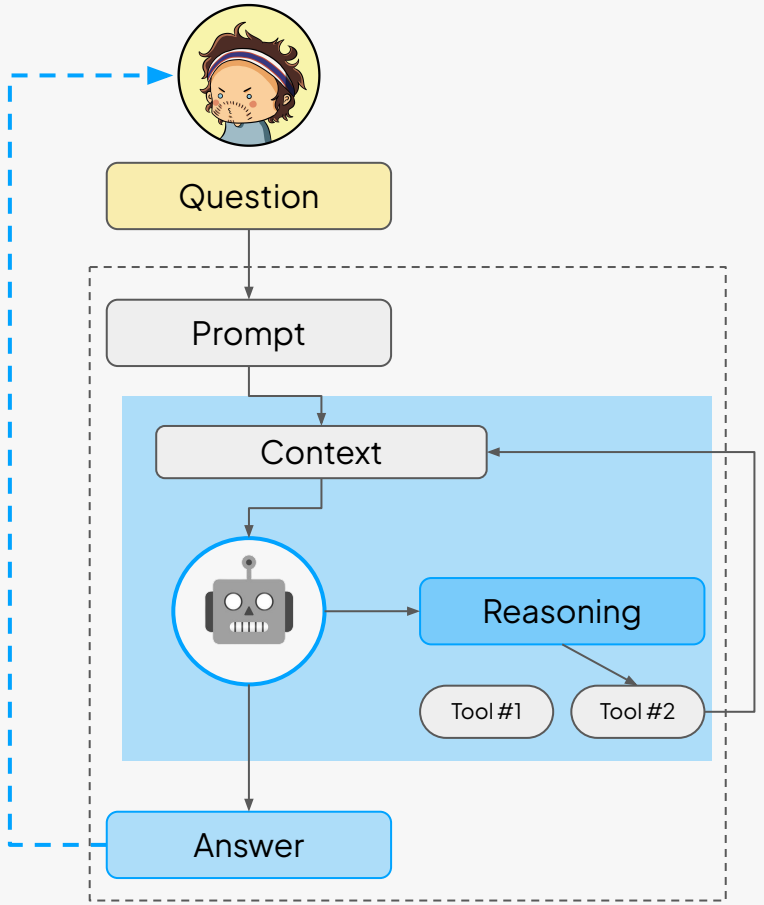
Are agents really different? RAG vs Agents

- Isn't RAG about "tools" as well?
  - In RAG, the coder calls the tool (the database) in a pre-defined, one-shot manner:
  - **Pre-defined**: any question is solved by the same logical flow.
  - **(Typically) One-shot**: there is one LLM completion call for each question.



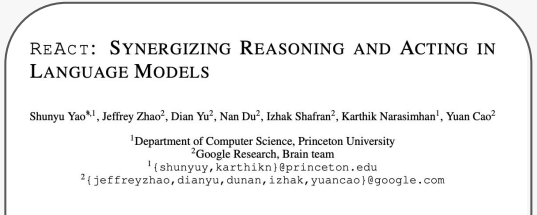
Are agents really different? RAG vs Agents

- Isn't RAG about "tools" as well?
  - In agentic AI, the LLM calls tools in a adaptive, multi-shot manner:
  - **Adaptive**: each problem can be solved through a different combination of actions and reasoning.
  - **(Possibly) Multi-shot**: the LLM gets repeatedly called with updates from the "world" to progressively refine its reasoning, until a final answer.



The ReAct loop: a first look

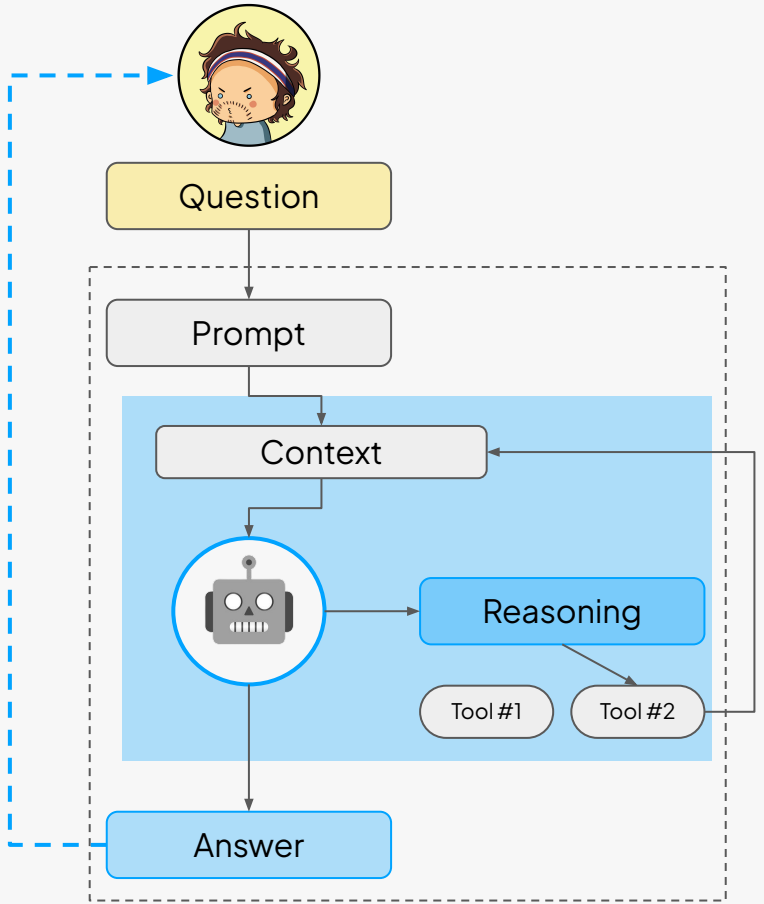
- Agents are a while loop, with:
  - A general purpose (“helping user with their Python code”) [system prompt]
  - A specific user question (“how do I write a pandas script that does X?”) [user prompt]
  - An acceptance condition (“Stop when code runs”)
  - Tools to act in the world (“A Python interpreter”)
  - Context, as in the accumulation of the results of acting in the world through tools (“What code was generated, which error was raised”)



REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

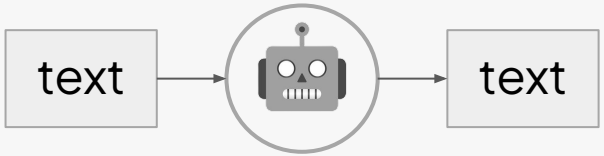
Shunyu Yao*[,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

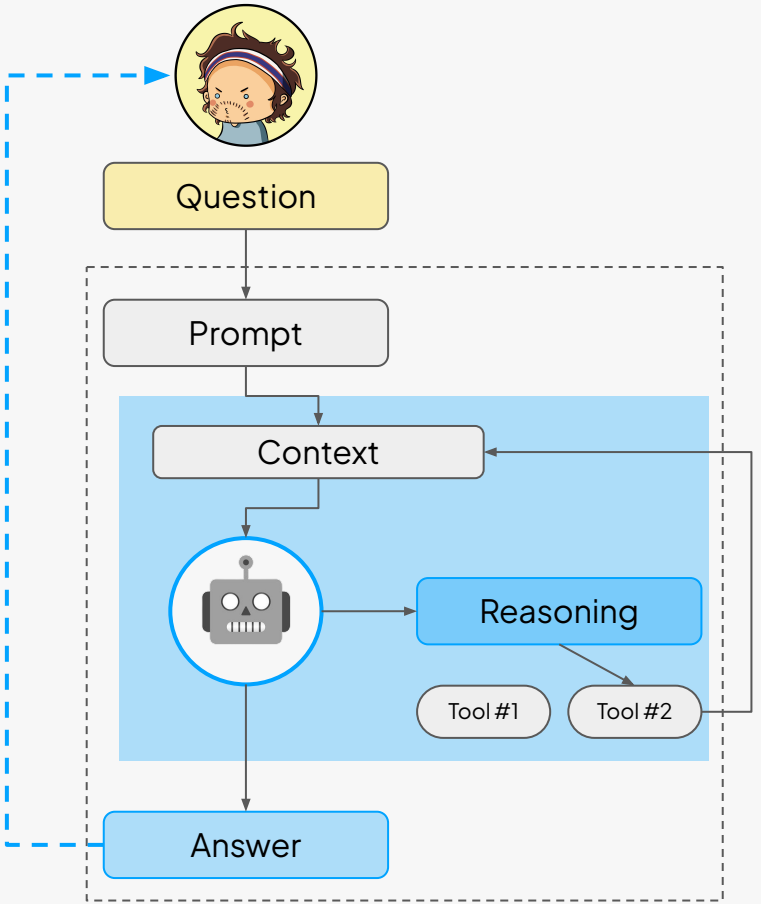
[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

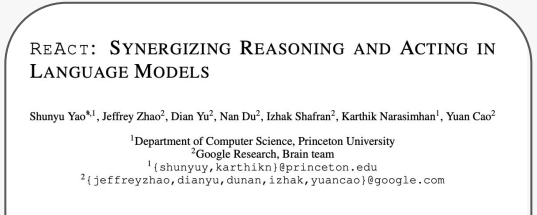


The ReAct loop: a first look

- Observation #1: since LLMs are functions from string to string, everything needs (*prima facie*) to be “text”, including tool calls!







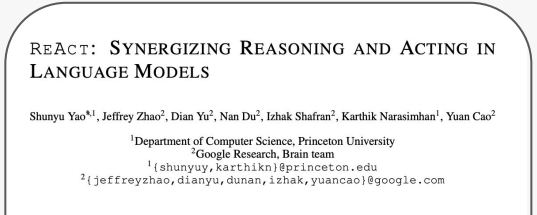
REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao[*,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com

The ReAct loop: a first look

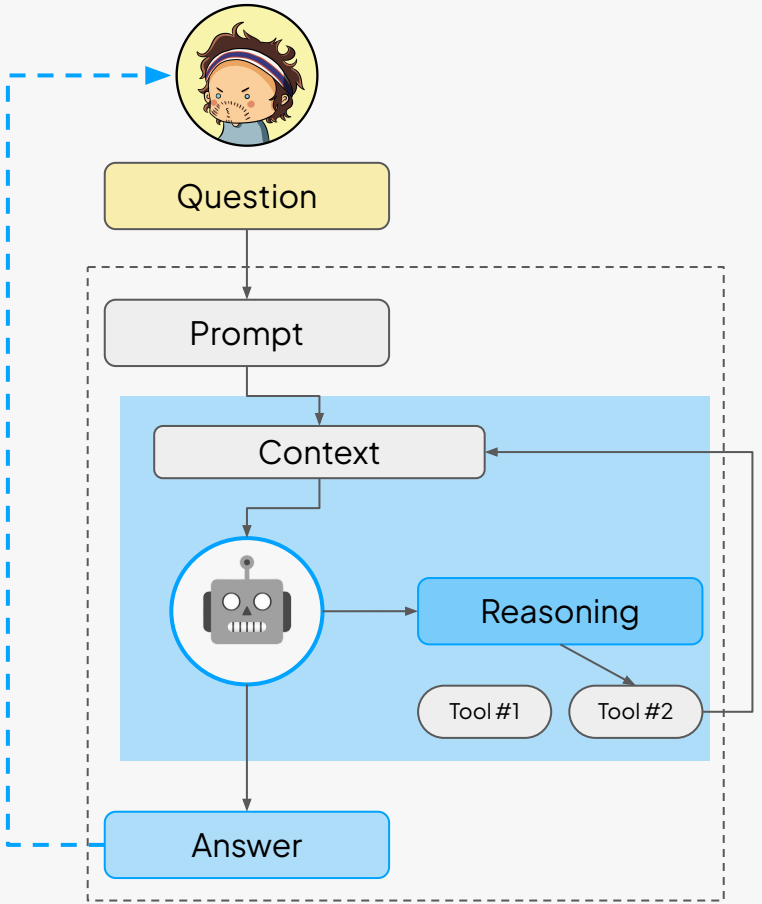
- Observation #1: since LLMs are functions from string to string, everything needs (*prima facie*) to be "text", including tool calls!
- Observation #2: unlike RAG, LLMs can self-correct (as long as the feedback from the world is "useful")
  - E.g. use tool1 to solve a problem, get an error, put the error in the context, reason better, use tool2, etc.



REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS

Shunyu Yao[*,1], Jeffrey Zhao[2], Dian Yu[2], Nan Du[2], Izhak Shafran[2], Karthik Narasimhan[1], Yuan Cao[2]

[1]Department of Computer Science, Princeton University
[2]Google Research, Brain team
[1]{shunyuy,karthikn}@princeton.edu
[2]{jeffreyzhao,dianyu,dunan,izhak,yuancao}@google.com



Agents-as-a-continuum

"Agentic AI" is a very broad category, and practitioners often talk past each other. The important takeaway when discussing agents is that in reality there is a continuum of "agency / autonomy":

- A chat bot accessing some APIs to answer a question may be called an agent ("can you please check the status of my Amazon order?").
- Claude code fixing your Python project may be called an agent ("can you please search the bauplan docs and update the APIs to the latest version"?).
- A deep research agent may be called an agent ("read all the latest papers on JOIN optimizations to improve this SQL query running in my Snowflake").

Hugging Face

Agency Level	Description	What that's called	Example pattern
☆☆☆	Agent output has no impact on program flow	Simple processor	`process_llm_output(llm_response)`
★☆☆	Agent output determines basic control flow	Router	`if llm_decision(): path_a() else: path_b()`
★★☆	Agent output determines function execution	Tool caller	`run_function(llm_chosen_tool, llm_chosen_args)`
★★★	Agent output controls iteration and program continuation	Multi-step Agent	`while llm_should_continue(): execute_next_step()`
★★★	One agentic workflow can start another agentic workflow	Multi-Agent	`if llm_trigger(): execute_agent()`

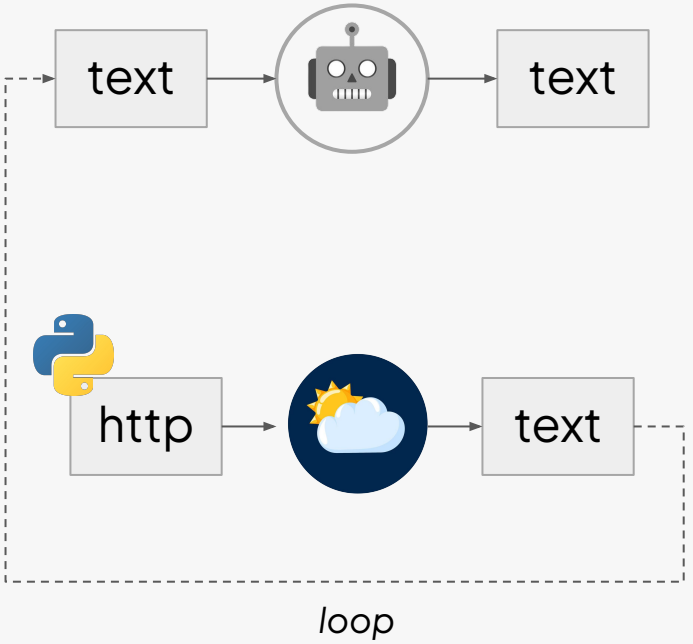
How to build an agent

Building a ReACT agent: text-to-tool-to-text

ReACT = reasoning + acting, but

- Reasoning is "easy", as we are entirely offloading that to an LLM.
  - Sure, we need to provide *context* and a good *prompt* (we'll get to that later), but there is no" interface" challenge.
- Acting is non-obvious, as words do not really have an effect on the (digital) world
  - a travel agent needs to use *weather APIs* to get next week's temperature for a destination - **APIs needs to be called from a Python script, we cannot "describe them".**

Observation #1: since LLMs are functions from string to string, everything needs (*prima facie*) to be "text", including tool calls!

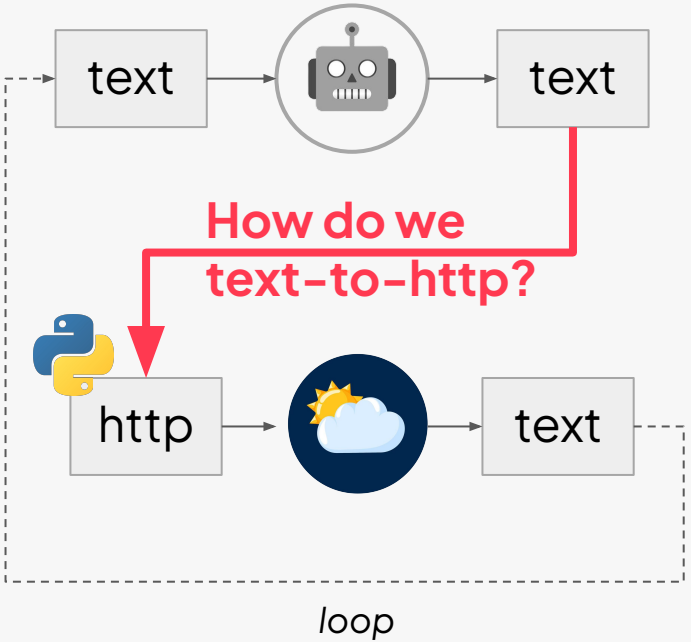


Building a ReACT agent: text-to-tool-to-text

ReACT = reasoning + acting, but

- Reasoning is "easy", as we are entirely offloading that to an LLM.
  - Sure, we need to provide *context* and a good *prompt* (we'll get to that later), but there is no" interface" challenge.
- Acting is non-obvious, as words do not really have an effect on the (digital) world
  - a travel agent needs to use *weather APIs* to get next week's temperature for a destination - **APIs needs to be called from a Python script, we cannot "describe them".**

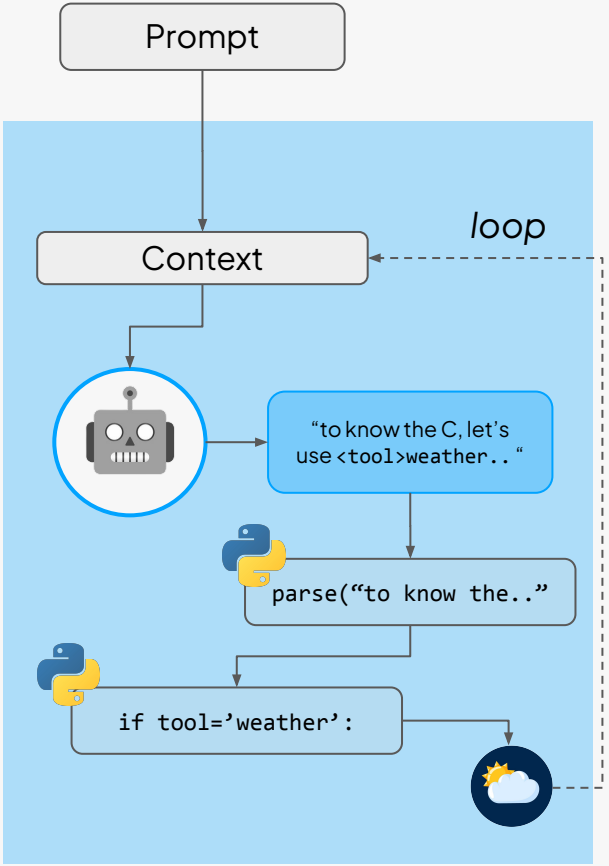
- Observation #1: since LLMs are functions from string to string, everything needs (*prima facie*) to be "text", including tool calls!



Building a ReACT agent: text-to-tool-to-text

The key insight is using text from the LLM to "instruct" the outer Python loop that the coder sets up:

- Add a tool description to the system prompt so that the LLM knows it can use it
  - "To check the temperature, you can call the weather tool passing a destination as a parameter"
- Add a naming convention to the system prompt so the outer loop can parse the output
  - "When you need to call a tool, output the tool name as for example <tool>weather</tool> and parameters as <parameter>nyc</parameters>

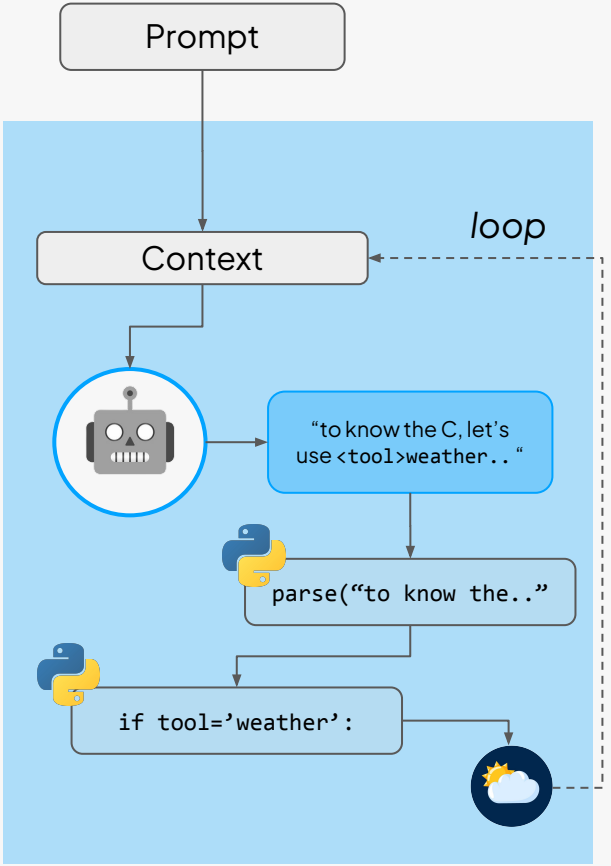


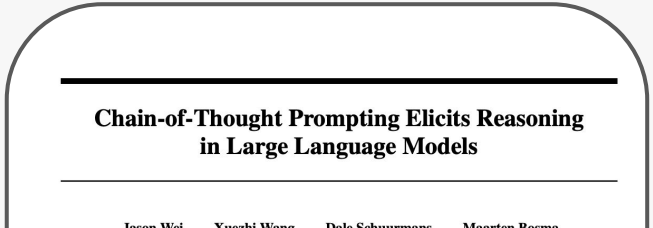
```
> uv run code.py
```

Building a ReACT agent: putting things together

We can now generalize our patterns:

- Add an explicit reasoning step
  - By forcing a model to “spend token thinking”, we can improve results and auditability (but don’t believe everything they say!).
- Add more tools
  - As the reasoning chain progresses, the model can pick one or the other.

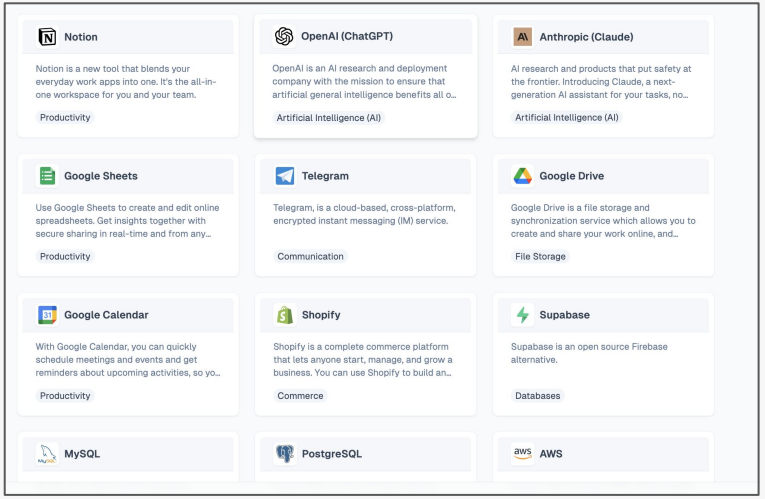



Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

Building a ReACT agent: calling services (MCP)

Ok, but what about “useful tools” and “connecting real systems”?

- We can’t manually write a function in Python “wrapping” a new API every time you need one.
  - It’s a Whac-A-Mole game: how many APIs expedia has?
  - We would need to standardize across models, APIs and code bases a way to parse and expose the tools.
  - This looks like the right moment to introduce **a standard – the MCP (Model Context Protocol)**
- Instead of manual integrations, we declaratively add tools by enabling our agents to access MCP servers, which standardize tool description, access , authentication etc.




```python
# Run agent with MCP tools
with MCPClient(server_parameters) as tools:
    agent = CodeAgent(tools=tools, model=model, add_base_tools=True)
    result = agent.run(question)

    return result
```

What now?

From dev to prod

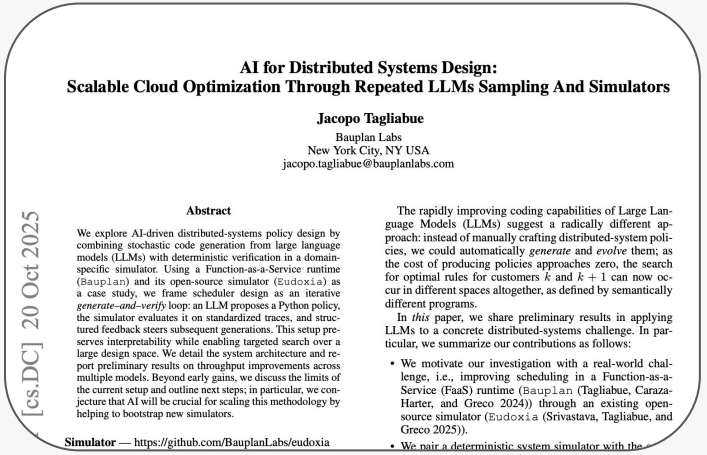
How do we test an agent?

- Of course, you need to do evaluations…
  - Prof. Greco will tell you all about that!
- On the “software side”, you should write tests that can act as “regression tests” and help you add tools and functionalities without constantly having to chase new errors.
  - In our scripts, for example, we keep track of which tools are used and make related assertion to verify that the LLM does indeed leverage the tools at disposal!

“Repeated sampling”

- Observation #2: unlike RAG, LLMs can self-correct (as long as the feedback from the world is “useful”)
  - E.g. use tool1 to solve a problem, get an error, put the error in the context, reason better, use tool2, etc.

- **What if we apply the same idea, but instead than “fixing errors” we “improve a metric”?** What if the answer to the user question is not a success / failure, but for example the solution to an optimization problem (so a scalar)?
  - Use tool to solve a problem, get a score, put the score in the context, reason better, try again, etc.



AI for Distributed Systems Design:
Scalable Cloud Optimization Through Repeated LLMs Sampling And Simulators

Jacopo Tagliabue
Bauplan Labs
New York City, NY USA
jacopo.tagliabue@bauplanlabs.com

Abstract

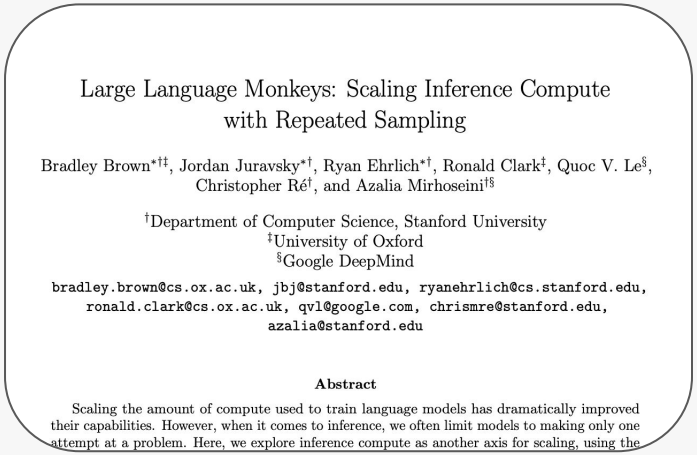
We explore AI-driven distributed-systems policy design by combining stochastic code generation from large language models (LLMs) with deterministic verification in a domain-specific simulator. Using a Function-as-a-Service runtime (`Bauplan`) and its open-source simulator (`Eudoxia`) as a case study, we frame scheduler design as an iterative *generate–and–verify* loop: an LLM proposes a Python policy, the simulator evaluates it on standardized traces, and structured feedback steers subsequent generations. This setup preserves interpretability while enabling targeted search over a large design space. We detail the system architecture and report preliminary results on throughput improvements across multiple models. Beyond early gains, we discuss the limits of the current setup and outline next steps; in particular, we conjecture that AI will be crucial for scaling this methodology by helping to bootstrap new simulators.

Simulator — https://github.com/BauplanLabs/eudoxia

The rapidly improving coding capabilities of Large Language Models (LLMs) suggest a radically different approach: instead of manually crafting distributed-system policies, we could automatically *generate* and *evolve* them; as the cost of producing policies approaches zero, the search for optimal rules for customers k and $k+1$ can now occur in different spaces altogether, as defined by semantically different programs.

In *this* paper, we share preliminary results in applying LLMs to a concrete distributed-systems challenge. In particular, we summarize our contributions as follows:

- We motivate our investigation with a real-world challenge, i.e., improving scheduling in a Function-as-a-Service (FaaS) runtime (`Bauplan` (Tagliabue, Caraza-Harter, and Greco 2024)) through an existing open-source simulator (`Eudoxia` (Srivastava, Tagliabue, and Greco 2025)).
- We pair a deterministic system simulator with the



Large Language Monkeys: Scaling Inference Compute with Repeated Sampling

Bradley Brown*†‡, Jordan Juravsky*†, Ryan Ehrlich*†, Ronald Clark‡, Quoc V. Le§, Christopher Ré†, and Azalia Mirhoseini†§

†Department of Computer Science, Stanford University
‡University of Oxford
§Google DeepMind

bradley.brown@cs.ox.ac.uk, jbj@stanford.edu, ryanehrlich@cs.stanford.edu, ronald.clark@cs.ox.ac.uk, qvl@google.com, chrismre@stanford.edu, azalia@stanford.edu

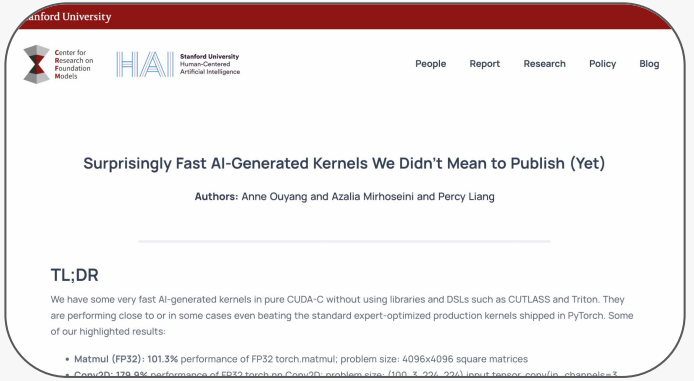
Abstract

Scaling the amount of compute used to train language models has dramatically improved their capabilities. However, when it comes to inference, we often limit models to making only one attempt at a problem. Here, we explore inference compute as another axis for scaling, using the

“Repeated sampling”

- Observation #2: unlike RAG, LLMs can self-correct (as long as the feedback from the world is “useful”)
  - E.g. use tool1 to solve a problem, get an error, put the error in the context, reason better, use tool2, etc.

- **What if we apply the same idea, but instead than “fixing errors” we “improve a metric”?** What if the answer to the user question is not a success / failure, but for example the solution to an optimization problem (so a scalar)?
  - use tool to solve a problem, get a score, put the score in the context, reason better, try tool again…

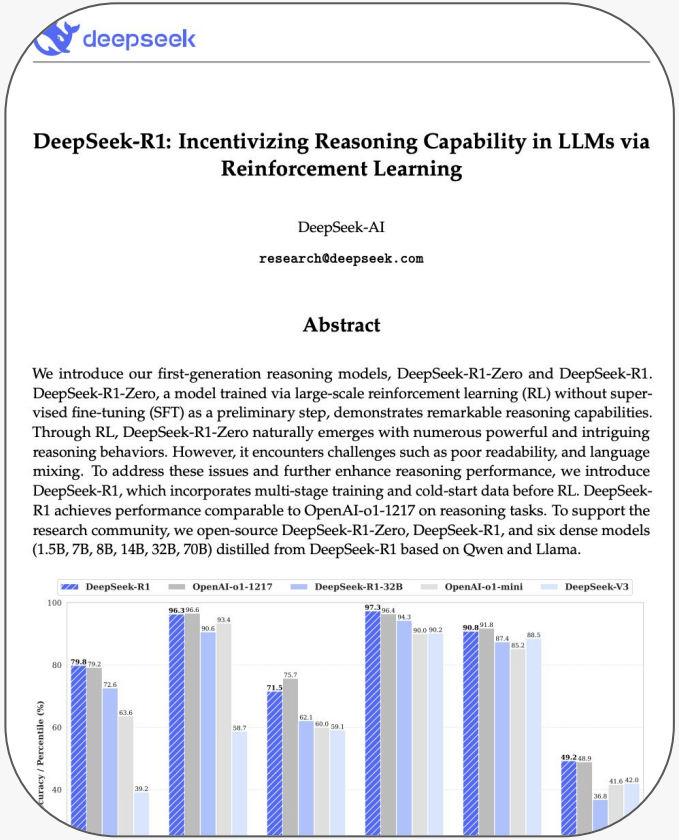


It works very well with problems whose solution is easy and cheap to verify!

From sampling to RL… .

Sometimes, general models cannot really grok your problem, even if you sample them a lot / for a long time.

- We cannot just prompt / sample, we need to change the weights.
  - We start from an open-source model, and use the reward to do backpropagation and get a new model, which is specifically trained to reason about your problem.
- The same intuition applies here: the easier it is to verify a task, the higher the probability that RL will produce superior results.



deepseek

DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI

research@deepseek.com

Abstract

We introduce our first-generation reasoning models, DeepSeek-R1-Zero and DeepSeek-R1. DeepSeek-R1-Zero, a model trained via large-scale reinforcement learning (RL) without supervised fine-tuning (SFT) as a preliminary step, demonstrates remarkable reasoning capabilities. Through RL, DeepSeek-R1-Zero naturally emerges with numerous powerful and intriguing reasoning behaviors. However, it encounters challenges such as poor readability, and language mixing. To address these issues and further enhance reasoning performance, we introduce DeepSeek-R1, which incorporates multi-stage training and cold-start data before RL. DeepSeek-R1 achieves performance comparable to OpenAI-o1-1217 on reasoning tasks. To support the research community, we open-source DeepSeek-R1-Zero, DeepSeek-R1, and six dense models (1.5B, 7B, 8B, 14B, 32B, 70B) distilled from DeepSeek-R1 based on Qwen and Llama.

jacopo.tagliabue@bauplanlabs.com

Want to know more? **Reach out**!

bauplan