#GlobalAzure
#GlobalAzureMilano

# Agenti Autonomi con Azure Open AI

Giancarlo Sudano
MICROSOFT

re
Milano

# Agenti Autonomi con Azure Open AI

Giancarlo Sudano
MICROSOFT

# Research Breakthroughs

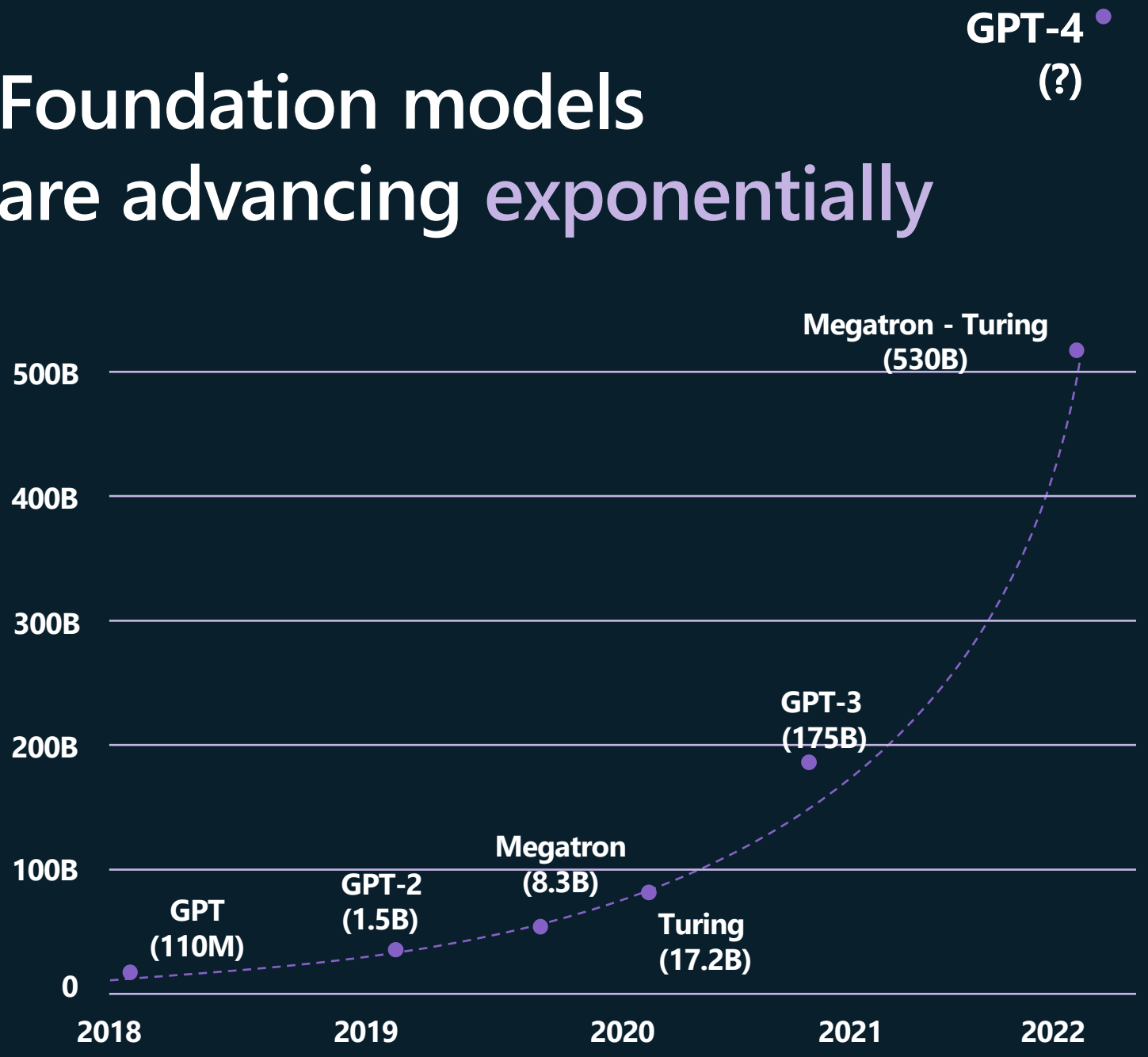
| | |
|---|---|
| 2016 | Object recognition *Human parity* |
| 2017 | Speech recognition *Human parity* |
| 2018 | Machine reading comprehension *Human parity* |
| 2018 | Machine translation *Human parity* |
| 2019 | Conversational QnA *Human parity* |
| 2020 | Image captioning *Human parity* |
| 2021 | Natural Language Understanding *Human parity* |
| 2021 | Commonsense Question Answering *Human parity* |
| 2022 | ChatGPT |
| 2023 | GPT-4 and GPT-4 Vision |

# High Level Performance

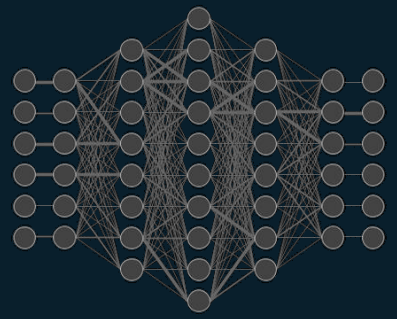

| GPT-4 test scores | Result<br>Estimated percentile |
|---|---|
| AP biology | 5<br>85th—100th |
| Uniform bar exam | 298/400<br>~90th |
| LSAT | 163<br>~88th |
| SAT reading & writing | 710/800<br>~93rd |
| SAT math | 700/800<br>~89th |

Source: OpenAI. (2023). GPT-4: Scaling up deep learning. Retrieved from https://openai.com/research/gpt-4.

# Foundation models are advancing exponentially



# Prompt Construction



Decoder

Large Language Model

NLP

NLU

- Entity Recognition
- Topic Classification
- Sentiment Analysis
- Other NLU tasks

NLG

- Summarization
- Paraphrase
- Sentence Generation
- Other NLG tasks

**Prompt Instruction**

Extract the name of this person in this text.
Text: "My name is Simon, order status?"

**Completion**

Entity (Name): Simon

**Prompt Instruction**

Decide whether a phrase's sentiment is positive, neutral, or negative.
Phrase: "How can I help you today?"

**Completion**

Sentiment: Positive

**Prompt Instruction**

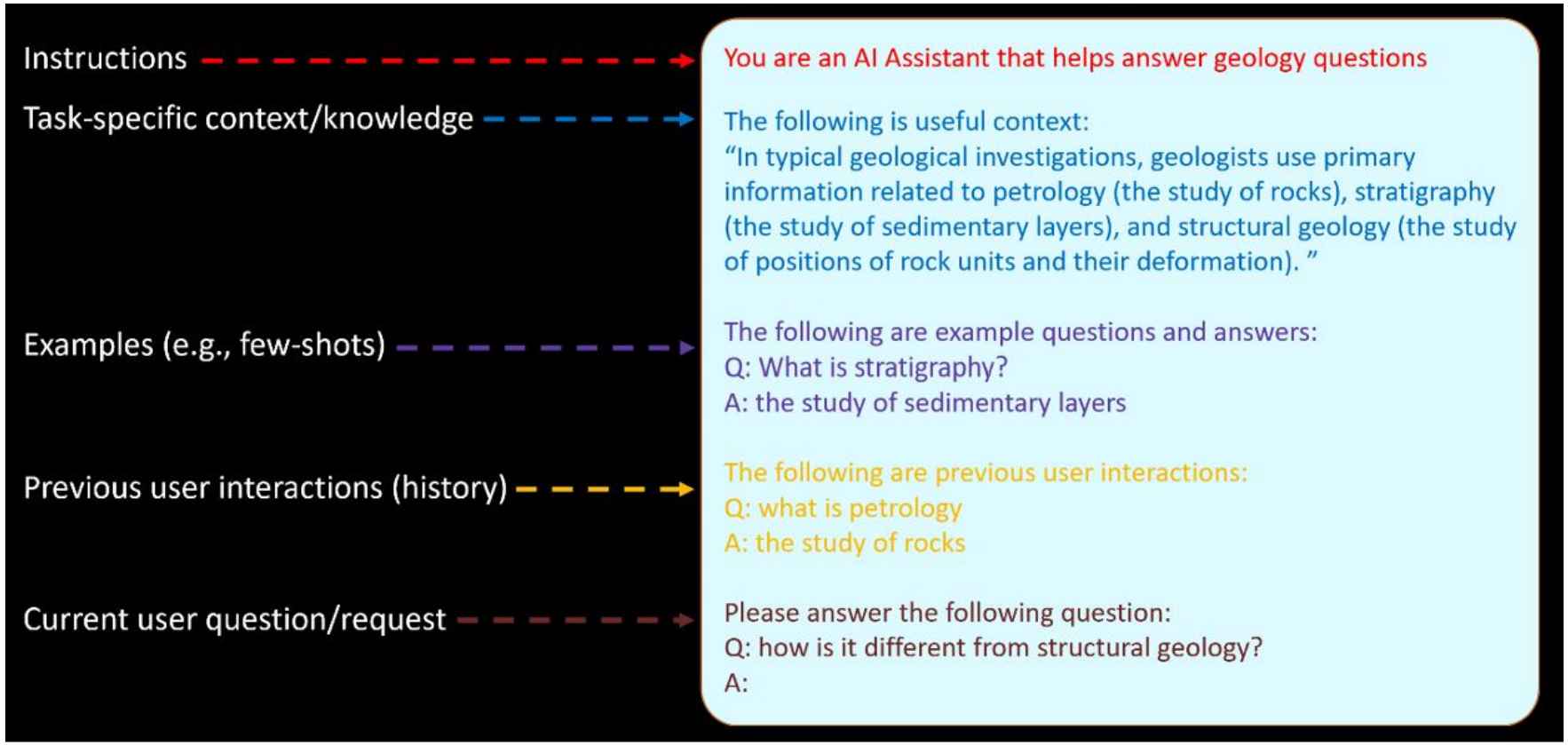
Summarize the following conversation:
Call Center: How can I help you today?
Customer: My name is Simon, order status?

**Completion**

Summary: Customer calling regarding an order.

| Zero-Shot | One-Shot | Few-Shot |
|---|---|---|
| **The model predicts the answer given only a natural language description of the task.** | **In addition to the task description, the model sees a single example of the task** | **In addition to the task description, the model sees a few examples of the task.** |

# Technical Prompt Limits




Limit input tokens (GPT4 128k)
Limit output tokens (GPT4 4k)
…and another subtle one…

# Research advancements on LLM capabilities

## SELF-REFINE: Iterative Refinement with Self-Feedback

**Aman Madaan**[1], **Niket Tandon**[2], **Prakhar Gupta**[1], **Skyler Hallinan**[3], **Luyu Gao**[1], **Sarah Wiegreffe**[2], **Uri Alon**[1], **Nouha Dziri**[2], **Shrimai Prabhumoye**[4], **Yiming Yang**[1], **Shashank Gupta**[2], **Bodhisattwa Prasad Majumder**[5], **Katherine Hermann**[6], **Sean Welleck**[2,3], **Amir Yazdanbakhsh**[6], **Peter Clark**[2]
[1]Language Technologies Institute, Carnegie Mellon University
[2]Allen Institute for Artificial Intelligence
[3]University of Washington [4]NVIDIA [5]UC San Diego [6]Google Research, Brain Team
amadaan@cs.cmu.edu, nikett@allenai.org

### Abstract

Like humans, large language models (LLMs) do not always generate the best output on their first try. Motivated by how humans refine their written text, we introduce SELF-REFINE, an approach for improving initial outputs from LLMs through iterative feedback and refinement. The main idea is to generate an initial output using an LLM; then, the same LLM provides *feedback* for its output and uses it to *refine* itself, iteratively. SELF-REFINE does not require any supervised training data, additional training, or reinforcement learning, and instead uses a single LLM as the generator, refiner and the feedback provider. We evaluate SELF-REFINE across 7 diverse tasks, ranging from dialog response generation to mathematical reasoning, using state-of-the-art (GPT-3.5 and GPT-4) LLMs. Across all evaluated tasks, outputs generated with SELF-REFINE are preferred by humans and automatic metrics over those generated with the same LLM using conventional one-step generation, improving by ~20% absolute on average in task performance. Our work demonstrates that even state-of-the-art LLMs like GPT-4 can be further improved at test-time using our simple, standalone approach.[1].

## Reflexion: Language Agents with Verbal Reinforcement Learning

**Noah Shinn**
Northeastern University
noahshinn024@gmail.com

**Federico Cassano**
Northeastern University
cassano.f@northeastern.edu

**Edward Berman**
Northeastern University
berman.ed@northeastern.edu

**Ashwin Gopinath**
Massachusetts Institute of Technology
agopi@mit.edu

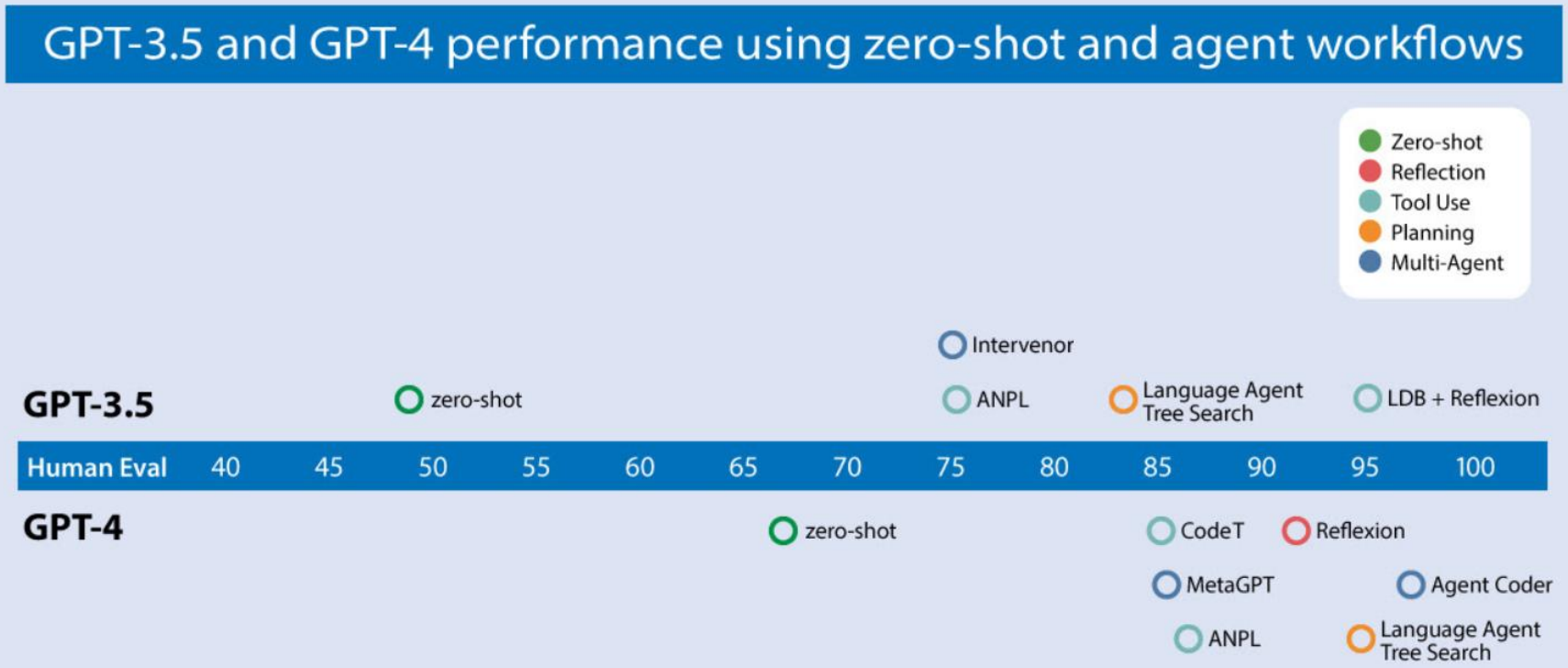
**Karthik Narasimhan**
Princeton University
karthikn@princeton.edu

**Shunyu Yao**
Princeton University
shunyuy@princeton.edu

### Abstract

Large language models (LLMs) have been increasingly used to interact with external environments (e.g., games, compilers, APIs) as goal-driven agents. However, it remains challenging for these language agents to quickly and efficiently learn from trial-and-error as traditional reinforcement learning methods require extensive training samples and expensive model fine-tuning. We propose *Reflexion*, a novel framework to reinforce language agents not by updating weights, but instead through linguistic feedback. Concretely, Reflexion agents verbally reflect on task feedback signals, then maintain their own reflective text in an episodic memory buffer to induce better decision-making in subsequent trials. Reflexion is flexible enough to incorporate various types (scalar values or free-form language) and sources (external or internally simulated) of feedback signals, and obtains significant improvements over a baseline agent across diverse tasks (sequential decision-making, coding, language reasoning). For example, Reflexion achieves a 91% pass@1 accuracy on the HumanEval coding benchmark, surpassing the previous state-of-the-art GPT-4 that achieves 80%. We also conduct ablation and analysis studies using different feedback signals, feedback incorporation methods, and agent types, and provide insights into how they affect performance. We release all code, demos, and datasets at https://github.com/noahshinn024/reflexion.

# High Level Performance




Performance of GPT-3.5 and GPT-4 (zero-shot) on HumanEval, along with algorithms that use agent workflows on top of GPT-3.5 or GPT-4. Thanks to Joaquin Dominguez and John Santerre for help with this analysis.

# Patterns for Agentic Applications

## Reflection

The LLM examines its own work to come up with ways to improve it

## Tools use

The LLM is given tools such as web search, code execution, or any other function to help it gather information, take action, or process data

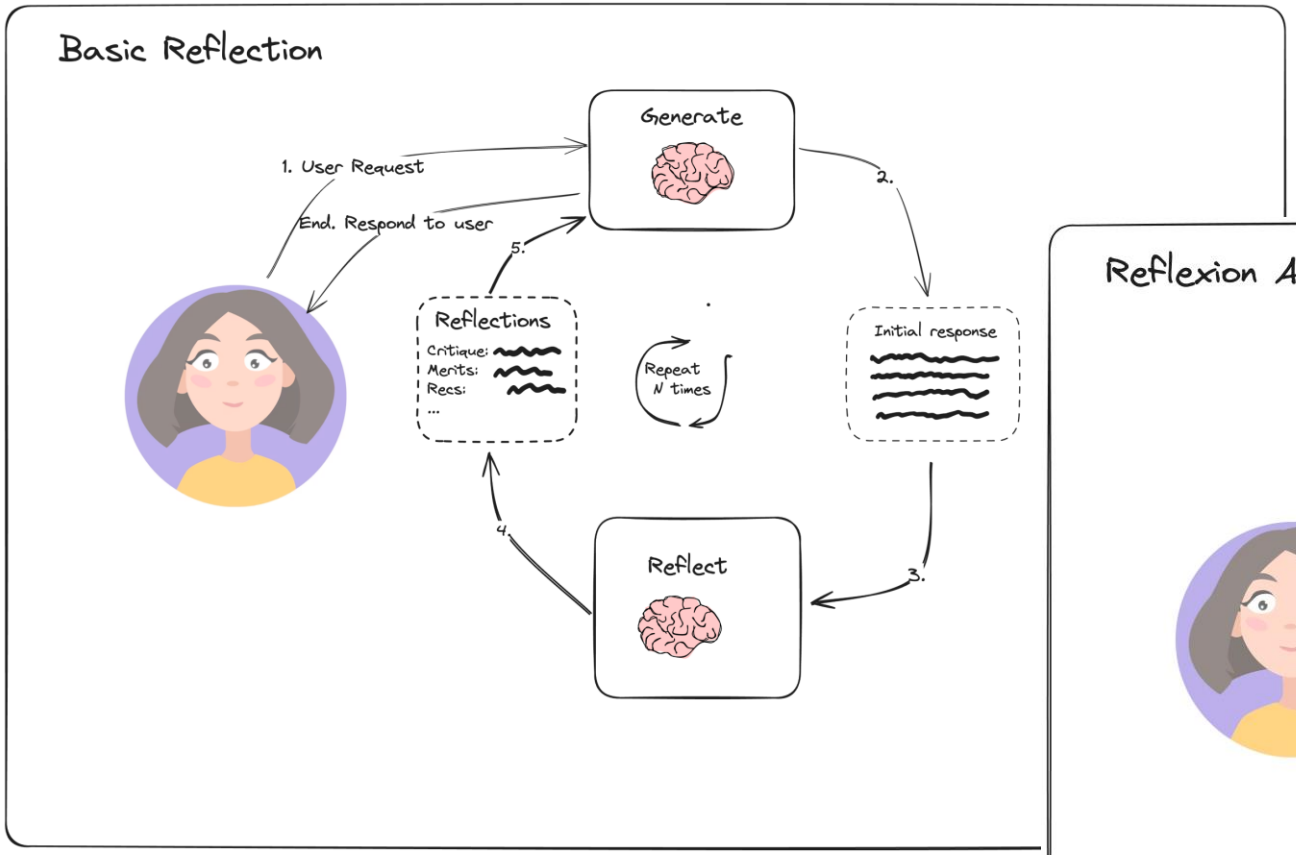
## Planning

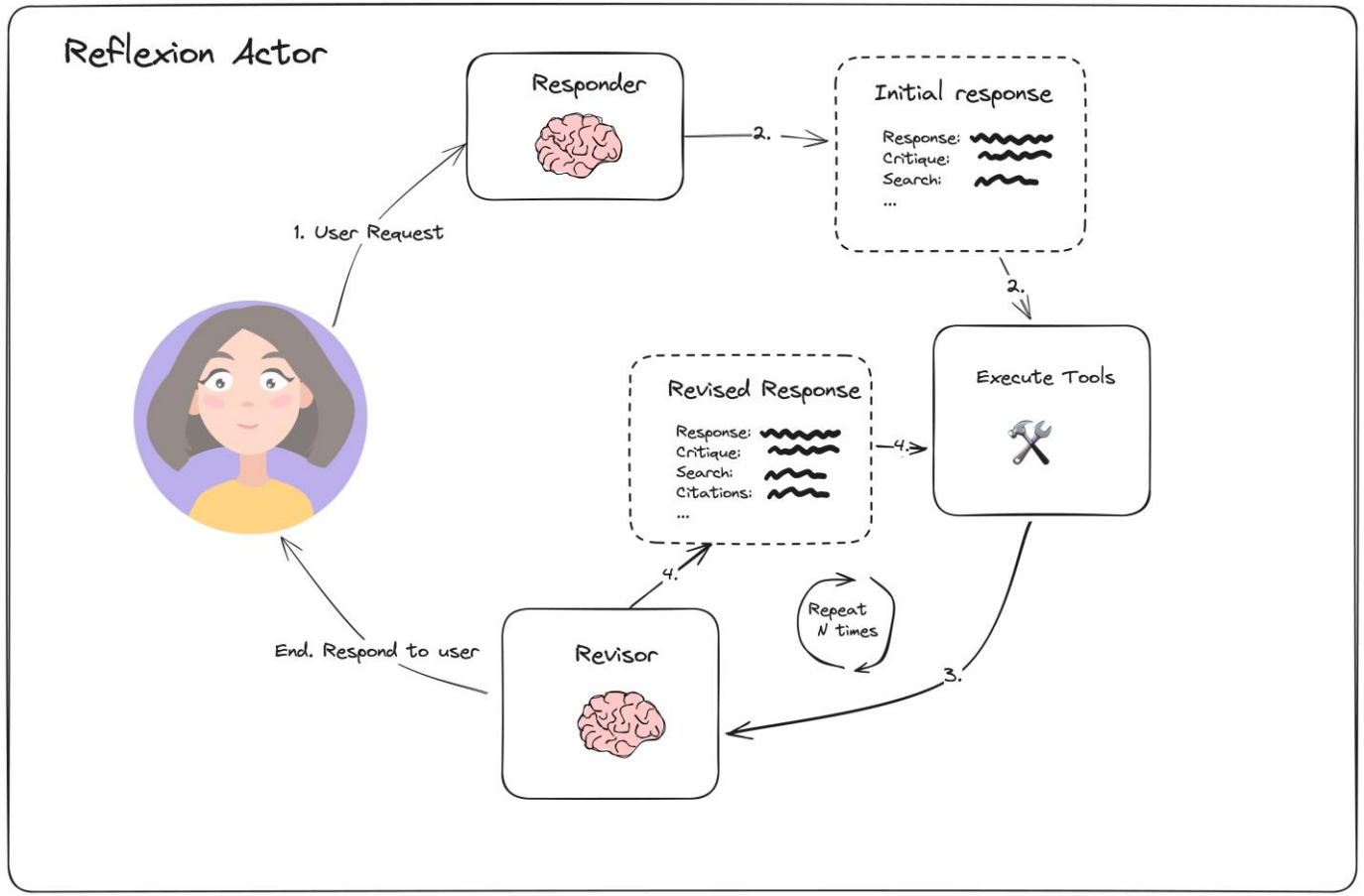
The LLM comes up with, and executes, a multistep plan to achieve a goal (for example, writing an outline for an essay, then doing online research, then writing a draft, and so on)

## Multi-agent

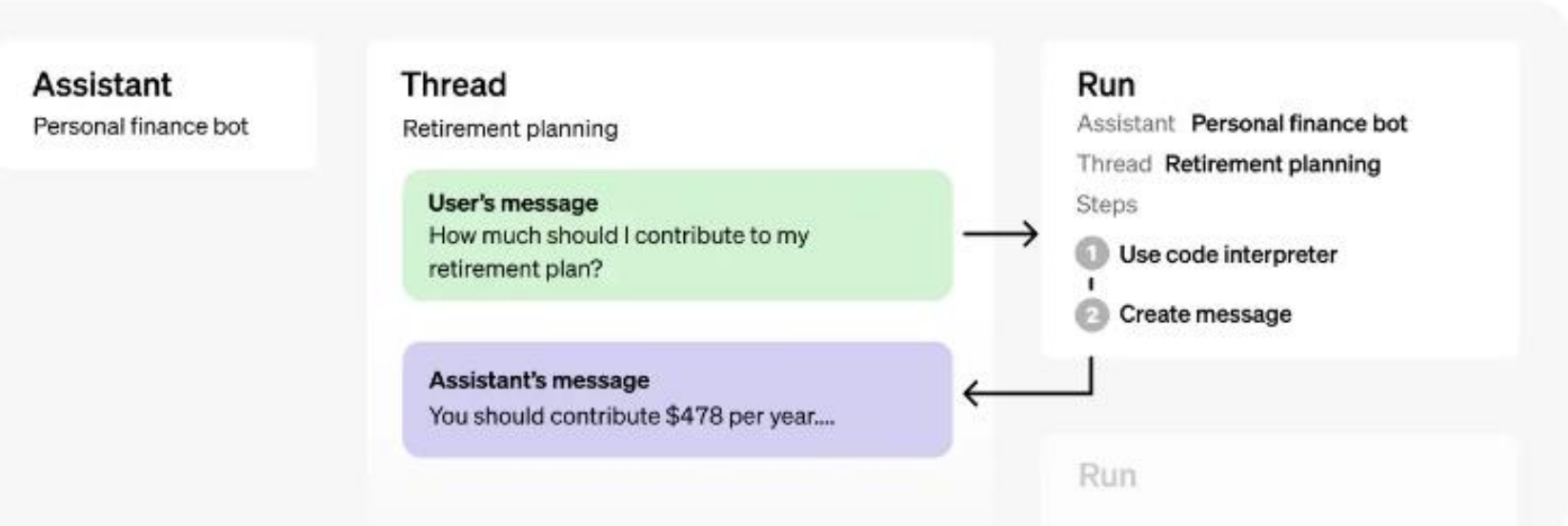
More than one AI agent work together, splitting up tasks and discussing and debating ideas, to come up with better solutions than a single agent would

# Reflection and Tool use

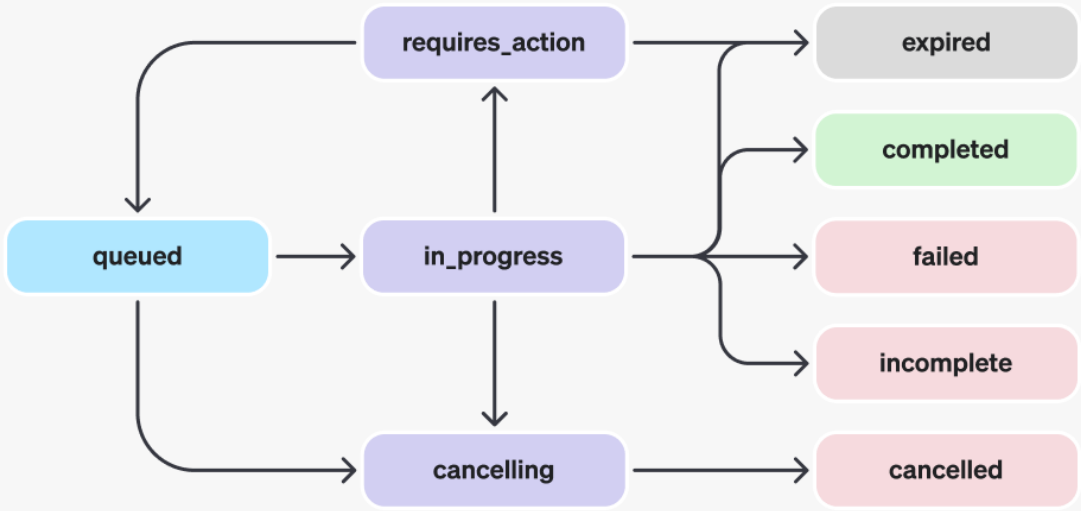


# Azure Open AI – Assistant API




| OBJECT | WHAT IT REPRESENTS |
|---|---|
| Assistant | Purpose-built AI that uses OpenAI's models and calls tools |
| Thread | A conversation session between an Assistant and a user. Threads store Messages and automatically handle truncation to fit content into a model's context. |
| Message | A message created by an Assistant or a user. Messages can include text, images, and other files. Messages stored as a list on the Thread. |
| Run | An invocation of an Assistant on a Thread. The Assistant uses its configuration and the Thread's Messages to perform tasks by calling models and tools. As part of a Run, the Assistant appends Messages to the Thread. |
| Run Step | A detailed list of steps the Assistant took as part of a Run. An Assistant can call tools or create Messages during its run. Examining Run Steps allows you to introspect how the Assistant is getting to its final results. |



# LangChain - LangGraph

downloads/month 75k | open issues 34 | LangChain Community | docs latest

⚡ Building language agents as graphs ⚡

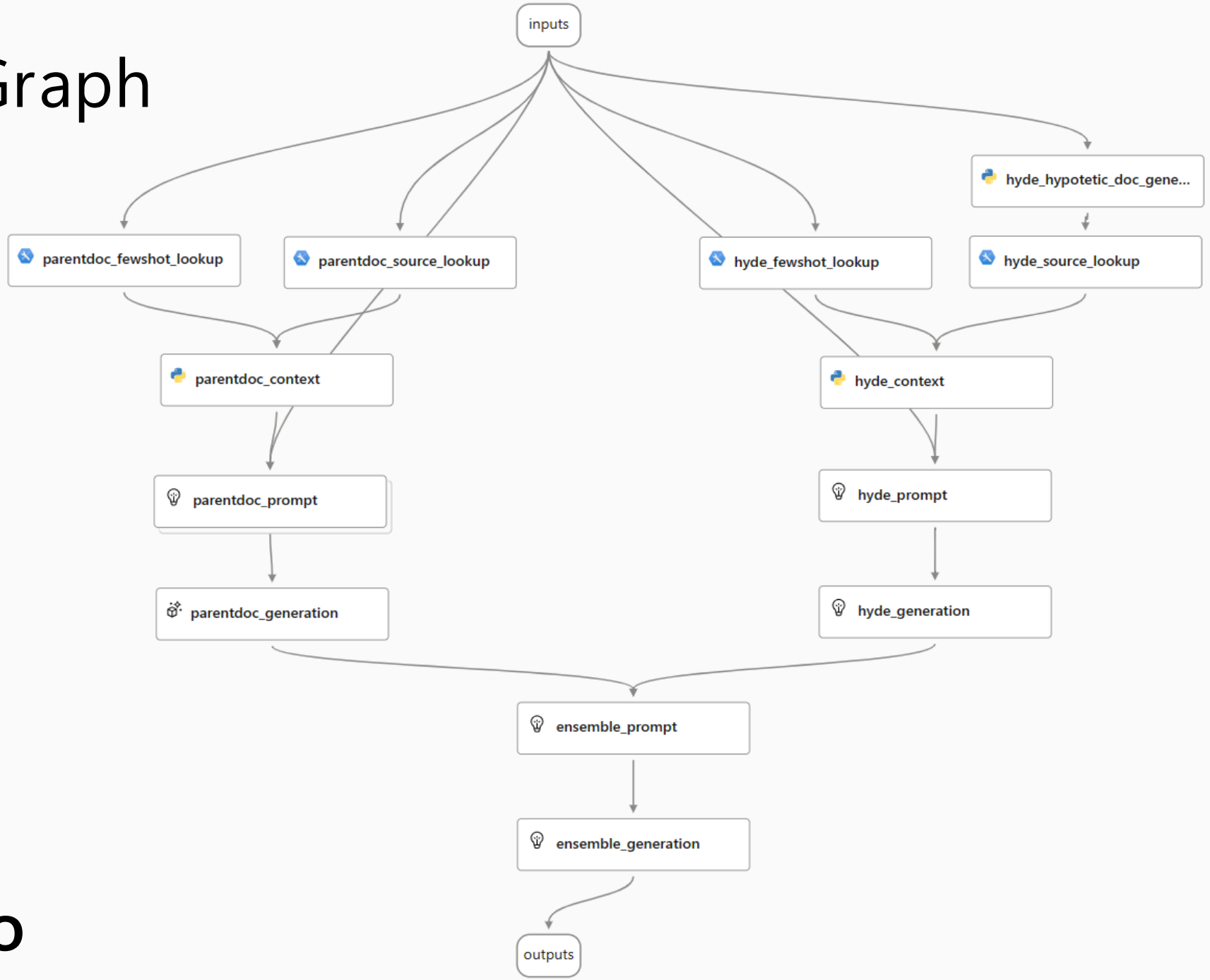
## Overview

LangGraph is a library for building stateful, multi-actor applications with LLMs, built on top of (and intended to be used with) LangChain. It extends the LangChain Expression Language with the ability to coordinate multiple chains (or actors) across multiple steps of computation in a cyclic manner. It is inspired by Pregel and Apache Beam. The current interface exposed is one inspired by NetworkX.

The main use is for adding **cycles** to your LLM application. Crucially, LangGraph is NOT optimized for only **DAG** workflows. If you want to build a DAG, you should just use LangChain Expression Language.

Cycles are important for agent-like behaviors, where you call an LLM in a loop, asking it what action to take next.

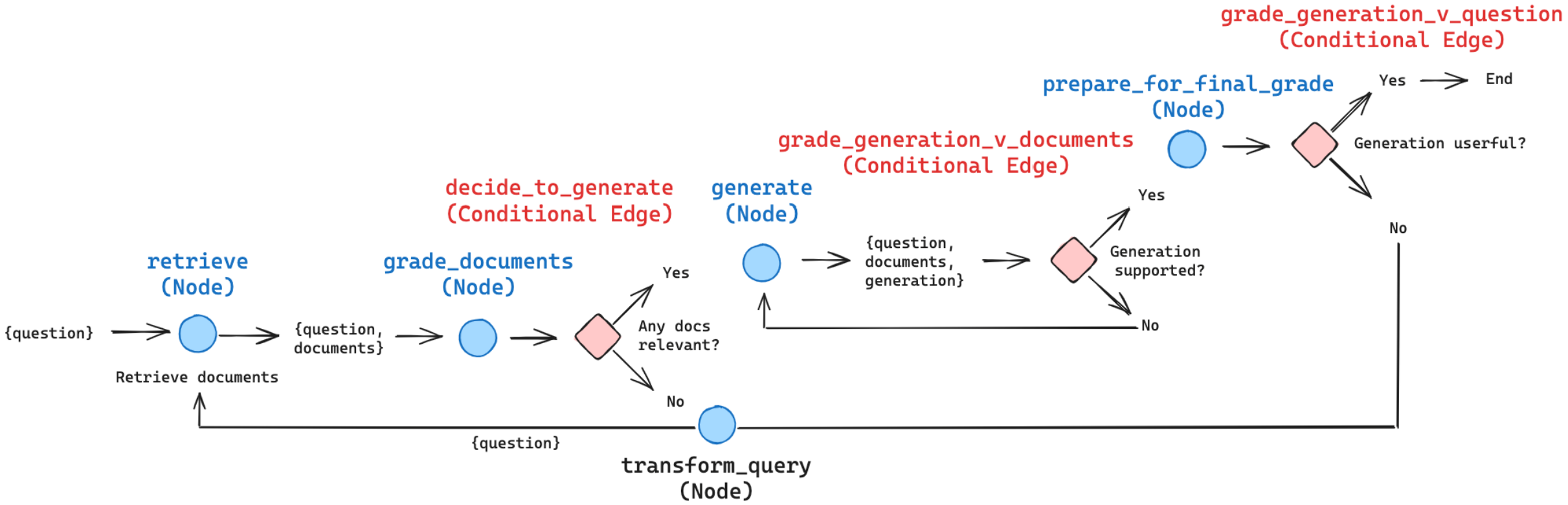
# Direct Aciclyc Graph



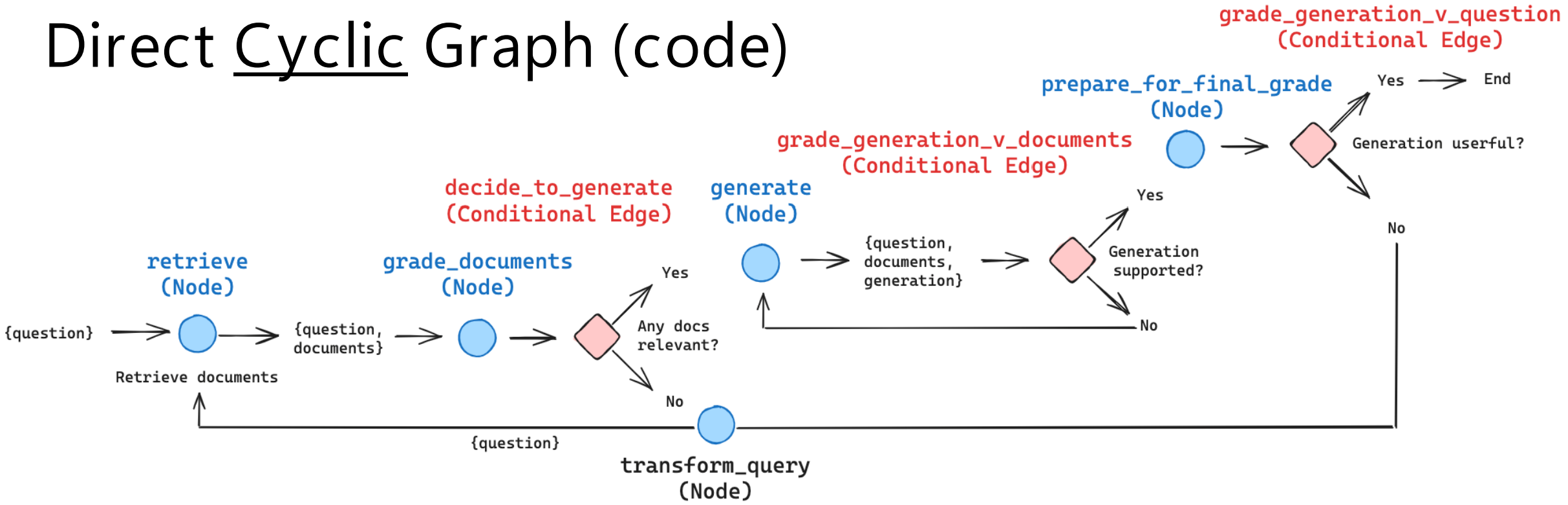

Prompt Flow

Azure AI Studio

# Direct Cyclic Graph with LangGraph



# Direct Cyclic Graph (code)




```
def retrieve(state):
    ...
    return {"documents": documents, "question": question}

def generate(state):
    ...
    return {"documents": documents, "question": question, "generation": generation}

def grade_documents(state):
    ...
    return {"documents": filtered_docs, "question": question, "web_search": web_search}

def transform_query(state):
    ...
    return {"documents": documents, "question": better_question}
```

```
class GraphState(TypedDict):
    """
    Represents the state of our graph.

    Attributes:
        question: question
        generation: LLM generation
        web_search: whether to add search
        documents: list of documents
    """
    question : str
    generation : str
    web_search : str
    documents : List[str]
```

# Patterns for Agentic Applications

## Reflection

The LLM examines its own work to come up with ways to improve it

## Tools use

The LLM is given tools such as web search, code execution, or any other function to help it gather information, take action, or process data

## Planning

The LLM comes up with, and executes, a multistep plan to achieve a goal (for example, writing an outline for an essay, then doing online research, then writing a draft, and so on)

## Multi-agent

More than one AI agent work together, splitting up tasks and discussing and debating ideas, to come up with better solutions than a single agent would

# Low code multi agents

Crew AI

Autogen

#GlobalAzure
#GlobalAzureMilano

# THANK YOU!!!

Slides will be available on Global Azure 2024 page
on Azure Meetup Milano website