CS590-06 Building Intelligent Agents with Frontier Models

Lecture 1: Introduction

Shuyan Zhou

2025-08-26

Lecture overview

- A brief introduction to large language models
- Limitations of LLMs on task automation
- The key ingredients of AI agents
- AI agent applications
- AI agent frameworks
- Class logistics
- In-class mixer

Large language models predict the next tokens

Mathematically, an LLM defines a probability distribution:

$$P(x_t | x_1, x_2, \ldots, x_{t-1})$$

LLMs act as chatbots

Responding to user queries in a conversational manner.

$$P(\mathbf{X}_i = x_{i,1}, x_{i,2}, \ldots, x_{i,n} | \mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_{i-1}) = \prod_{j=t}^{n} P(x_{i,j} | \mathbf{X}_1, \ldots, \mathbf{X}_{i-1}, x_{i,1}, \ldots, x_{i,j-1})$$

where $x_{i,1}, x_{i,2}, \ldots, x_{i,n}$ are the tokens generated at turn i.

This formulation is deceptively powerful but also limiting.

Let’s book a flight ticket with an LLM

👤: I need to book a flight to visit my parents this weekend.

🤖: Where are you flying from?

👤: Boston.

🤖: Where are your parents?

👤: Chicago.

🤖: What is the airline you use the most?

👤: United.

🤖: Great, I will book you a ticket.

🤖: Wait. I cannot **actually access booking websites**. I can only generate text. **The best I can do is to give you a step-by-step guideline**. First, you will login into United airline with you account name and password. Second, you will enter .. …

👤: Hold on hold on, never mind, I think I will just book the ticket myself

Three weeks later ..

👤: I need to book a flight ticket to visit my friend Alexis in SF this Saturday.

🤖: Where are you flying from?

👤: Boston.

🤖: What is the airline you use the most?

👤: Wait, I think I have told you all these information before .. Never mind, I will book the ticket myself.

The LLM again asks all the **same** clarifying questions (origin, airline etc), with **no** memory of the previous exchange.

LLMs are reactive next token generators, not actors

They cannot:

- Access external environment (e.g., booking websites) and using tools (e.g., a weather query API)
- Take actions on behalf of the user
- Store and retrieve long-term memory (e.g., remember your preferences)

Hence, pure LLMs are not **agents**.

Key ingredients: external environments

- The **world** where the task is defined and performed
- Dynamic, partially observable, often outside the agent's control.
  - State change
  - No global information
- **Examples:** a computer, our physical world

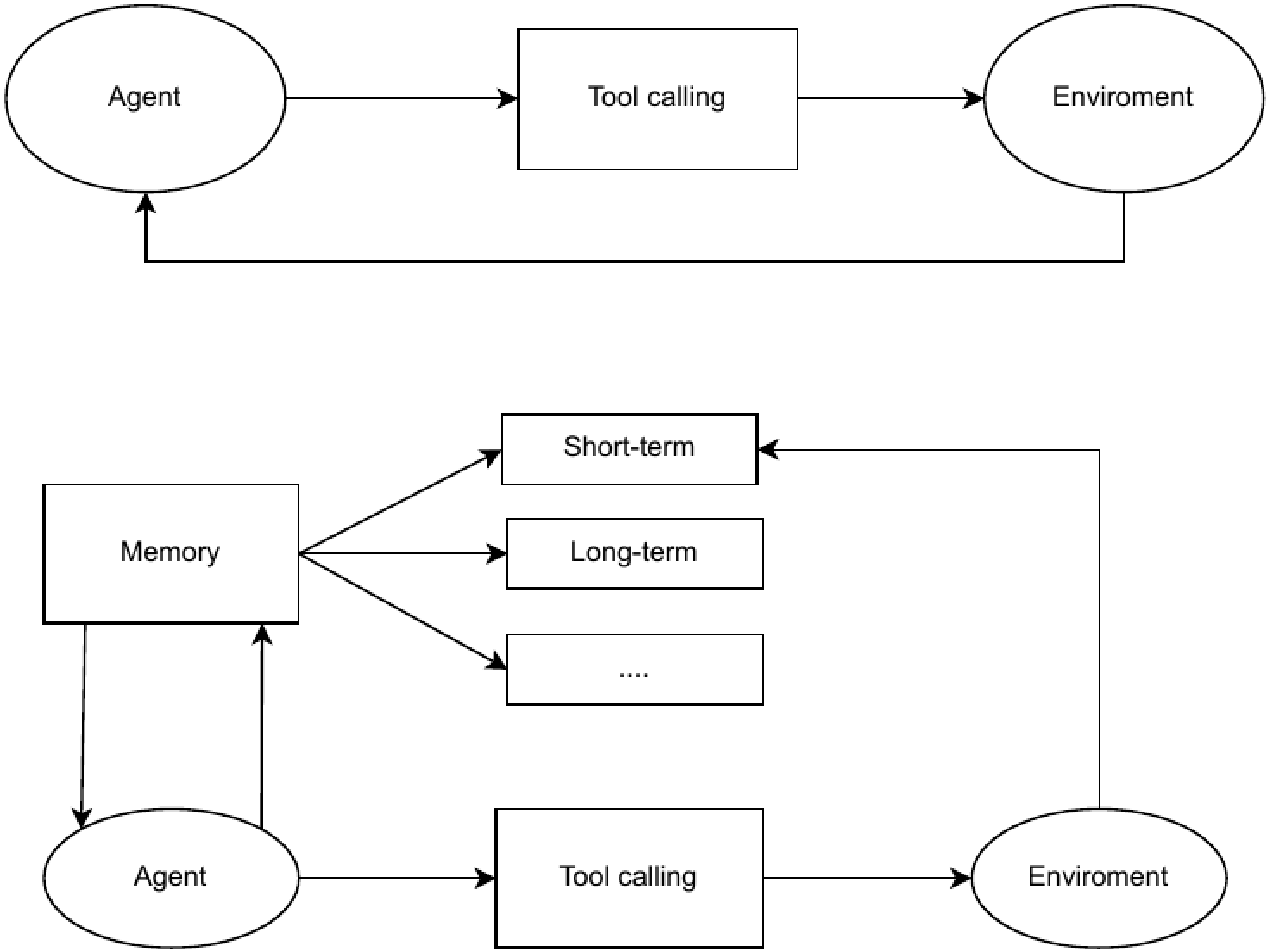
Key ingredients: tools

- The **interface** given to the agent to interact with the environment.
- Think of them as **functions/APIs** the agent can call.
- **Examples:**
  - Generic tools: An API that can translate `click (123, 456)` to the actual click event on a computer.
  - Specialized tools: a `search_flights` API, a calculator, a database query, or an `add_expense` function.

Key ingredients: memory

- **Short-term memory:** keeps track of things in the moment.
- **Long-term memory:** stores knowledge over time.
- There are also other ways to categorize memory, such as
  - **Procedural memory:** how-to knowledge.
  - **Semantic memory:** stores general knowledge and facts.

AI agents in a nutshell



Formulating as a Partially Observable Markov Decision Process (POMDP)

Assumes the agent **cannot directly observe the true state of the environment**.

A POMDP is formally defined as a tuple $(S, A, O, T, \Omega, R, \gamma)$, where:

- **States** S: the set of possible environment states (hidden from the agent).
- **Actions** A: the set of actions available to the agent.
- **Observations** O: the set of possible observations the agent can receive.
- **Transition model** $T(s_t, a_t, s_{t+1}) = P(s_{t+1} \mid s_t, a_t)$: probability that taking action a_t in state s_t results in state s_{t+1}.
  - In fully deterministic environments, this is a function $S \times A \to S$.
- **Observation model** $O(o_{t+1} \mid s_{t+1}, a_t)$: probability of observing o_{t+1} given that the agent took action a_t and arrived in state s_{t+1}.
  - In deterministic cases, each state corresponds to a unique observation.

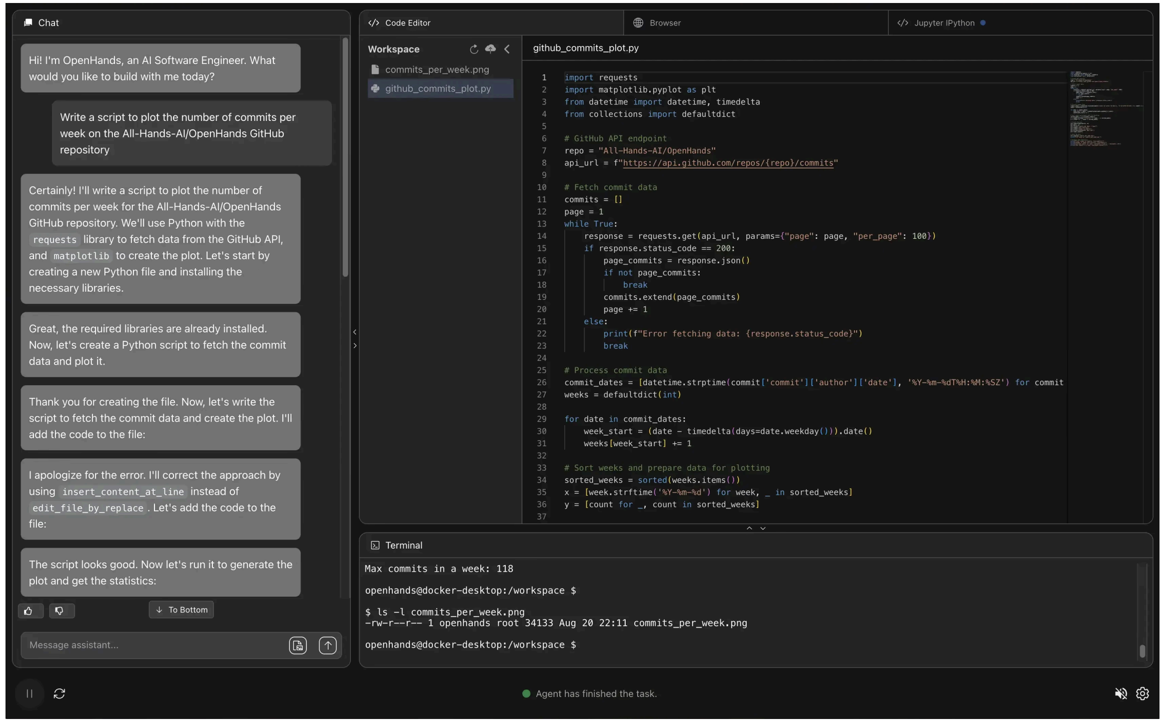
POMDP (cont'd)

- **Reward function** $R(s, a)$: expected immediate reward obtained by taking action a in state s.
  - In many cases, reward is **sparse** and only received in specific situations (e.g., reaching a goal state).
- **Discount factor** $\gamma \in [0, 1)$: determines how much future rewards are weighted relative to immediate rewards.
  - An impatient player (low γ) grabs quick points.
  - A patient player (high γ) invests in strategies that pay off later.

Application: Computer use / Digital task automation

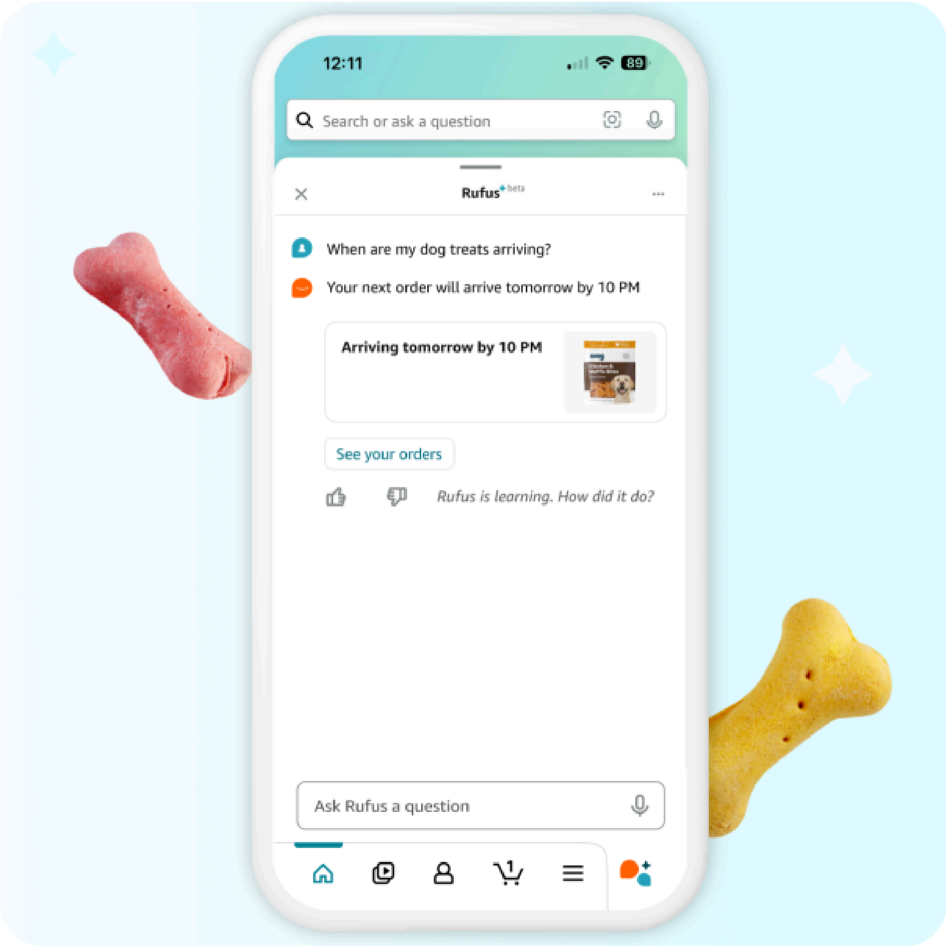
Claude | Computer use for automating operations

Application: AI engineers



The agent has access to essential development tools such as code editor, terminal and a browser.

Customer service (chat + tool use)



A basic agent: ReAct (Yao et al 2023)

- **Loop:** Goal → Observation → (Reasoning → Action) → Observation → ….
- Treat **reasoning itself** as an action that changes the **internal** state of the agent, rather than the environment.

Question: Aside from the Apple remote, what other devices can control the program Apple remote was originally designed for?

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.

Act 1: `Search[Apple Remote]`
Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple […] originally designed to control the Front Row media center program […]

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.

Act 2: `Search[Front Row]`
[…]

ReAct effectively defines a straightforward for loop

```python
def react_agent(goal, max_steps=10):
  observation = get_initial_observation()

  for step in range(max_steps):
    # Generate thought about current situation
    thought = llm.generate(f"Goal: {goal}\nObservation: {observation}\nThought:")

    # Decide on action based on thought and observation
    action = llm.generate(f"Goal: {goal}\nObservation: {observation}\nThought: {thought}\r

    # Execute action in environment
    env.execute(action)
```

Reasoning can go beyond reactive

Besides reason about the immediate observation, the agent can also

- Break high-level task into subtasks.
- Examine the history, perform backtracking or re-planning if necessary

Hence, the control loop can be more complicated.

```python
def planning_agent(goal, max_steps=10):
  # Initial planning phase
  plan = llm.generate(f"Create plan for: {goal}")
  subtasks = parse_plan_into_subtasks(plan)
```

```python
  for subtask in subtasks:
    # Execute subtask using ReAct loop
    while not is_subtask_completed(subtask):
      thought = llm.generate(f"Current subtask: {subtask}")
      action = llm.generate(f"Based on thought, what action?")
      observation = env.execute(action)
```

```python
      # Replan if stuck
      if should_replan(observation):
        subtasks = replan(goal, current_progress)
        break
```

Go beyond single agent

- **Multi-agent systems**: Multiple agents collaborate or compete
  - e.g., code review (writer/reviewer), debate systems, hierarchical teams

```python
def multi_agent_collaboration(goal, agents):
  # Distribute work among agents
  subtasks = decompose_task(goal, num_agents=len(agents))
```

```python
  for round in range(max_rounds):
    results = []
    for i, agent in enumerate(agents):
      # Each agent works on their subtask
      result = agent.execute(subtasks[i])
      results.append(result)
```

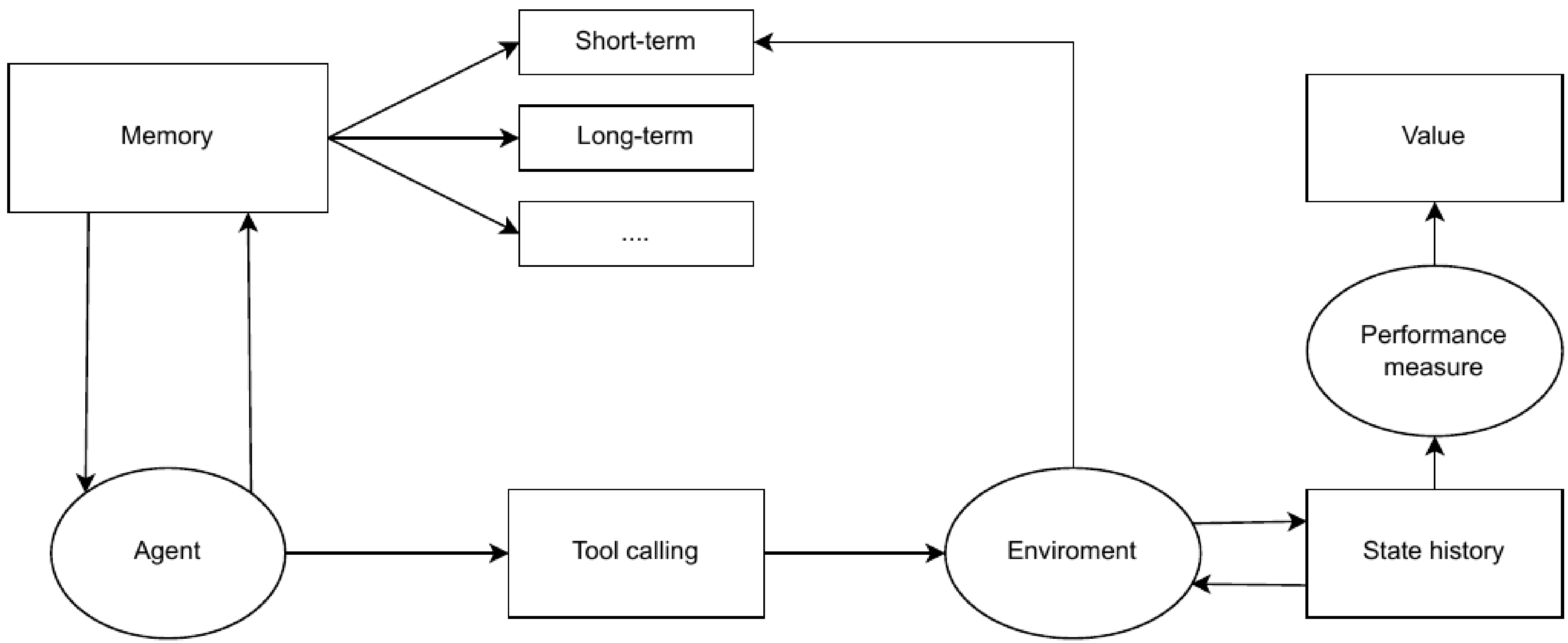
```python
    # Agents communicate and share results
    shared_knowledge = aggregate_results(results)

    # Update each agent's understanding
    for agent in agents:
      agent.update_knowledge(shared_knowledge)

    if is_goal_achieved(shared_knowledge, goal):
      break
```

Evaluating AI agents

How could we evaluate the performance of AI agents effectively?



Verifiable Tasks

Directly check the **correctness** of the task executions. For example

- **Coding agents**: run tests; correctness = binary.
- **Math problem solvers**: check against ground-truth answer.

Non-Verifiable Tasks

- **Creative or subjective tasks** (e.g., write a sci-fi novel)
- Find aspects that can be quantified, or directly ask for human preference.
- Rubric
  - e.g., Does the story has over 50k words?
  - Does this novel has a complicated plot with X twists and turns?
- Human preference (e.g., LMArena)
  - e.g., Given novel A and B, which one would you like better?
- **Open question**: to what extent can we turn open-ended tasks to verifiable ones?

Agents are systems, equal performance ≠ equal value.

An **agent** is a *system*, not just a model.

- **Agent 1**: LLM + advanced calculator
- **Agent 2**: LLM + basic calculator
- Task: solve algebra problems.
- Observation: Both agents achieve similar performance on advance math problem solving.
- Which method is “better”?

Depends on **what** are we looking for

- Agent 2 may reveal stronger reasoning in the LLM
- Agent 1 leans more on its tool, which may be insightful for understanding how agents use complex tools.

Goal of this class

Understand the basic concepts of AI agents, how they work, and how they can be applied in various scenarios through **student-led** paper discussions and **hands-on** projects.

Syllabus

- **First 4 classes**: Lectures
  - Class 1: Introduction to AI Agents
  - Class 2: Pretraining and Scaling Laws
  - Class 3: Post-training: SFT, RLHF and RLVR
  - Class 4: Evaluation and benchmarking
- **Remaining classes**: Paper discussions (students present & lead).
  - 25 minutes presentation + 10 minutes discussions
  - Please checkout the course website for the topics. Listed papers subjected to minor updates.
- Guest lectures + project presentations

Topic list

Grading

- Class participation (15%)
- In-class paper presentation (25%)
- Project (60%)
  - Lighting talk: 15%
  - Project presentation: 35%
  - Project code + report: 50%

Important Dates

- Main presenters: Submit slides in the dedicated Slack channel 24 hours before class.

Presentation

- 8/26: Form out in Slack to submit presentation preferences
- 8/28: Assignment out, open for negotiation, swap
- 9/2: Final assignment out

Project

- 8/28-9/16 Team formation and topic preference
- 10/7 in-class project lighting talk
- 11/17 (Mon) & 11/19 (Wed): Submit the project presentation slides
- 12/11: Final report

Contact

- You will be invited to a **Slack channel** for course announcements and discussions.
- **Office hour**: 2:40-3:10 PM every Thursday (after the Thursday class)

In-Class Mixer: Exploring AI Agents

Next class

- Language Models are Few-Shot Learners.
- LLama 3.1 Sections 1, 2, 3.1, 3.2, 3.4, 5.1