CS 577: NATURAL LANGUAGE PROCESSING

Guest Lecture: Yunxin Sun

Lecture 24: Language Model Agents

[some slides borrowed from https://language-agent-tutorial.github.io/ and Stanford cs224n]

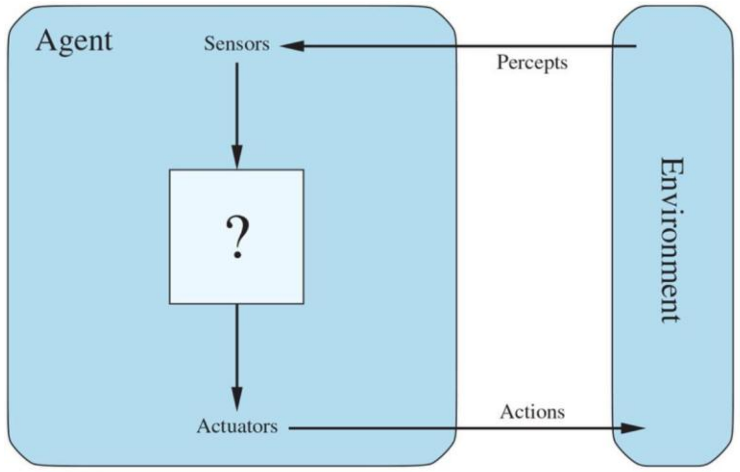
TODAY'S LECTURE

- Introduction
- Core Capabilities
    - Memory
    - Planning
    - Tool Using
    - Reasoning
    - Multi-agent
- Applications and Evaluations
- Conclusions

ARE THESE AGENTS?

WHAT ARE AGENTS?

- Agents are one of the most fundamental concepts in the history of AI.



“An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.” –– Russell & Norvig, AI: A Modern Approach (2020)

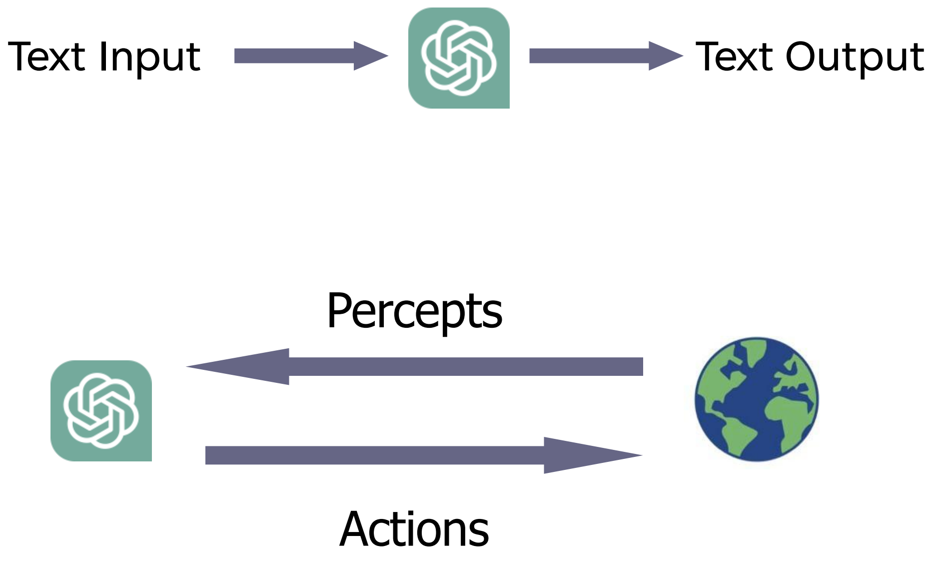
AGENTS: HISTORICAL PERSPECTIVE

- Agents that are around for a few decades (Logics-based Agent)
  - Software
  - Robotic Arms
- Characteristics
  - Do what you tell them
  - Very reliable
  - Relatively low cost

AGENTS: HISTORICAL PERSPECTIVE

- Agents that are around for a few years (NN-based Agent)
  - Game agents
  - Robots
- Characteristics
  - Do what you want
  - But only in a limited domain!
  - Could be expensive to train or deploy

MODERN LLM-BASED AGENTS



Modern LLM-based Agents
= LLM + Environments?

But still token in, token out!

WHY LLM-BASED AGENTS?

- Language is at the core of human intelligence and a universal vehicle for reasoning and communication
- LLM is the first to give us a sense of AGI (Artificial General Intelligence)
- LLM has many strong capabilities that can be leveraged
  - In-context learning
  - Instruction following
  - Reasoning and meta-reasoning
  - Decision making
- What about other modalities and perception?
  - Many frontier LLMs have native multimodality support.
  - But language still plays an important role

TWO COMPETITIVE VIEWS

- ***LLM-First View***: We make LLM an agent!
  - Implications: scaffold on top of LLMs, prompting-focused, heavy on engineering

- ***Agent-First View***: We integrate LLM into AI agents!
  - Implications: All the same challenges by previous AI agents, but we need to re-examine them through the new lens of LLMs

HYPE VS THE REALITY

Agents are bringing about the ***biggest revolution in computing*** *since we went from typing commands to tapping on icons.*

Bill Gates

I think AI agentic workflows will drive ***massive AI progress*** *this year.*

Andrew Ng

2025 is when ***agents will work****.*

Sam Altman

Current agents are just **thin wrappers around LLMs**.

LLMs **can never reason or plan**.

Auto-GPT's limitations in ... reveal that it is **far from being a practical solution**.

AGENTIC AI RESEARCH LANDSCAPE

Common Issues

Synthetic Data | Efficiency | Reliability | Safety | Evaluation

Core Capability

- Reasoning
- Self-evolving
- Tool Using
- Self-reflection
- Planning
- Multi-agent
- Memory
- ?

Applications

Science Agents

Retrieval Agents

Data Agents

Coding Agents

GUI Agents

We will discuss two flavors for each core capability

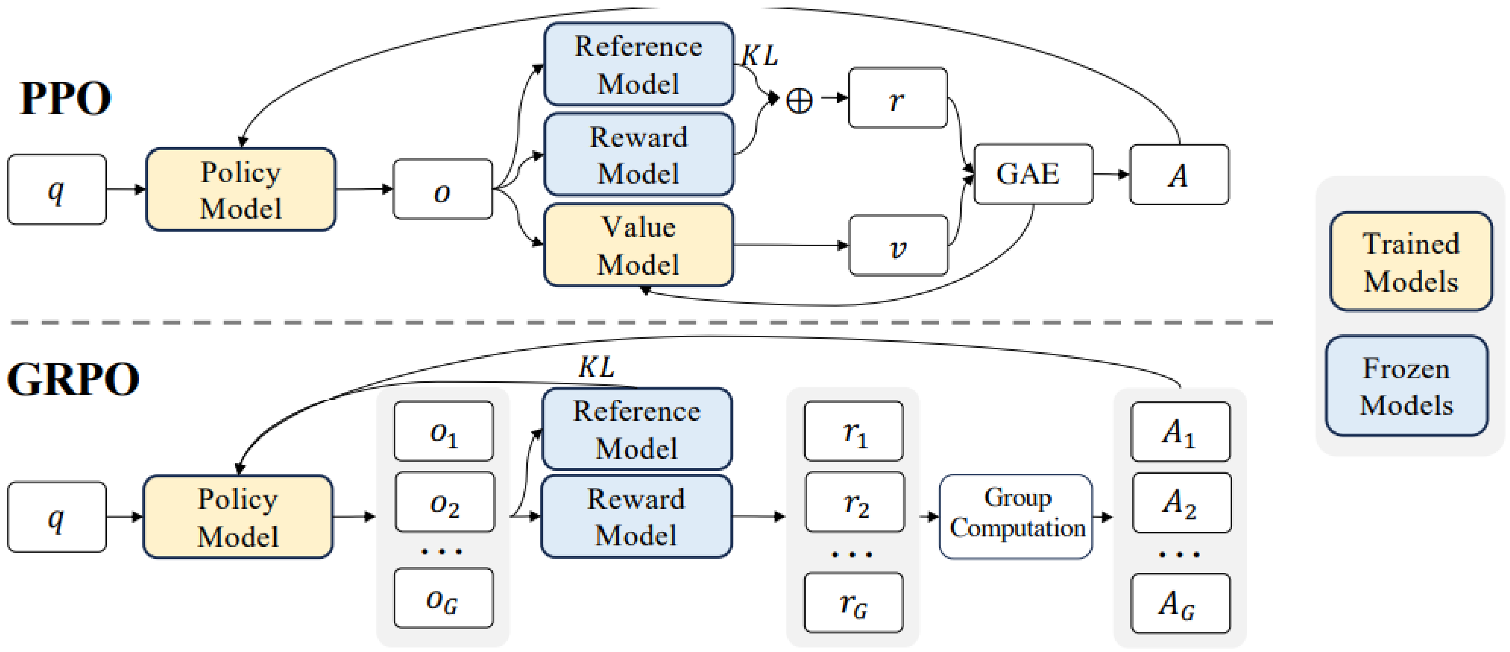
REASONING

- Reasoning is the most important capabilities for an LLM to be *agentic*
- This is why frontier AI labs devote so many resources to improve reasoning capabilities
- Informally, reasoning is the process of answering questions from what you already know.
- There are many *flavors* of reasoning as well: deductive reasoning, abductive reasoning, inductive reasoning, causal reasoning, social reasoning, common sense reasoning, counterfactual reasoning...
- We will cover two aspects
  - How to train a reasoning model via RL
  - How to prompt the model to elicit the reasoning capabilities

REASONING VIA RL

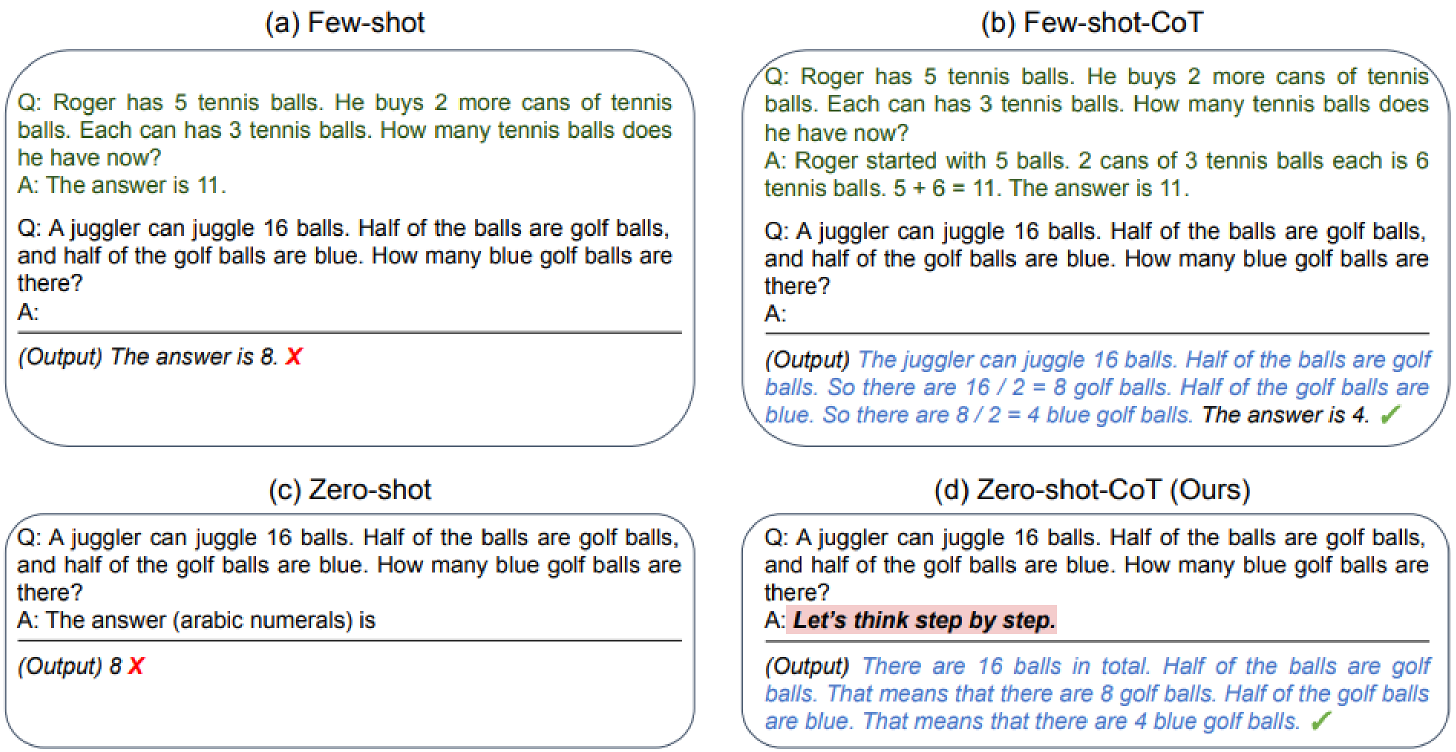
- Remember our RLHF lecture? We first train a reward model, and then we use the reward to reinforcement fine tune the language model using PPO.
- The same recipe can be used for training reasoning models as well. The only difference is that we don't need to train a reward model; the correctness itself is the reward!
- This is also called RLVR: *Reinforcement Learning with Verifiable Rewards*
- RLVR can also use GRPO: a *simplified* version of PPO from DeepSeek; instead of using the value model, we directly estimate the value by Monte Carlo sampling to estimate

REASONING VIA RL: GRPO



DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models

REASONING VIA PROMPT: COT AND COT-SC



CoT-SC: instead of greedy decoding, sample several times and do a majority vote.

REACT-STYLE AGENTIC REASONING PROMPT

```
You are an agent capable of the following actions:
1. Type X on Y
2. Move mouse to
3. Click on X
4. Type Char x on Y

Your objective is to follow user instructions, by mapping them into a sequence of
actions.
Instruction: {g}

So far, you have taken the following actions and observed the following
environment states:

Previous Actions and Observations:
o1:
a1:

o2:
a2:
…

After executing these actions, you observe the following HTML state: <HTML state>

Now, think about your next action:
Thought: [model-pred]

Now, take an action:
```

1. Action space in text
2. Instruction in text .
3. Previous observations and actions
4. Provide current observation [as text]

Model generates next action and use that action to update the environment and repeat.

ReAct: Synergizing Reasoning and Acting in Language Models

REACT-STYLE AGENTIC REASONING PROMPT

- Core ideas
  - Implicit language action space (thoughts) and explicit action space
  - Explicit action space interacts with the environment to generate observations
  - Observations are fed into the language model to generate next implicit language actions (thoughts)
  - Repeat the process

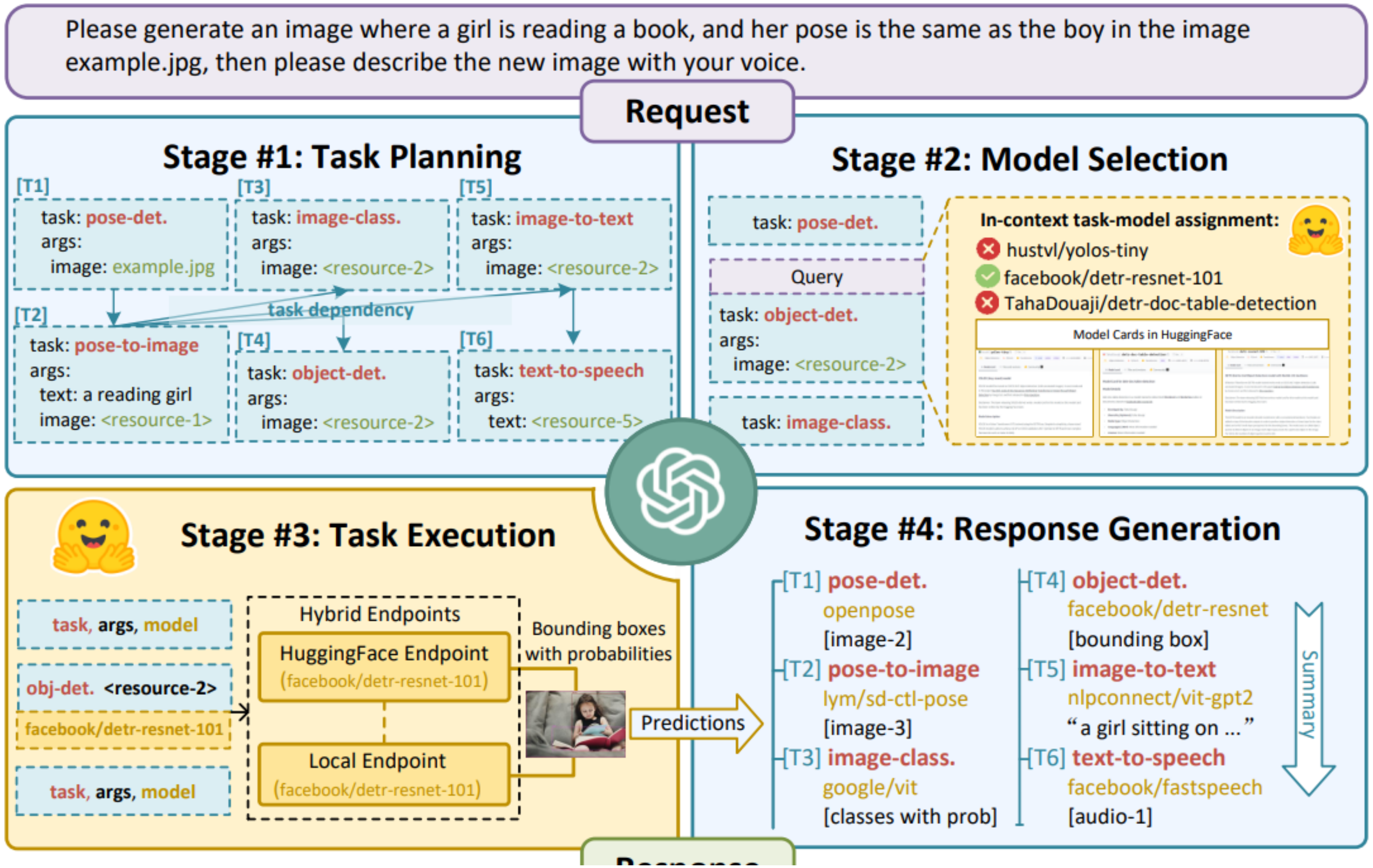
REASONING SUMMARY

- Train a reasoning model uses the same recipe as RLHF.
- Prompting based methods: CoT, CoT zero shot, CoT-SC, ReAct.
- Reasoning model is expensive but favored for agentic tasks.

PLANNING

- Planning is the process of "thinking ahead". Formally, given an environment E, a goal g, Θ as parameters, and P as prompts; ***plan*** seeks to find a sequence of actions to achieve the goal
  - $p = (a0, a1, \cdots, at) = plan(E, g; \Theta, P)$
- We will discuss two approaches
  - *Use LLM to decompose the problem into solvable pieces*
  - *Use an external planner*

THE HUGGINGGPT APPROACH



HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face

THE HUGGINGGPT APPROACH

- Given a request, it goes through these stages: **task planning**, model selection, task execution, and response generation.
- Let the LLMs decide the followings:
  - Analyze the user request and decompose it into a collection of structured tasks.
  - Determine dependencies and execution orders for these decomposed tasks.
  - Output the task in JSON formats
  - Key idea: planning is emergent from the prompt

HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face

TASK PLANNING PROMPT

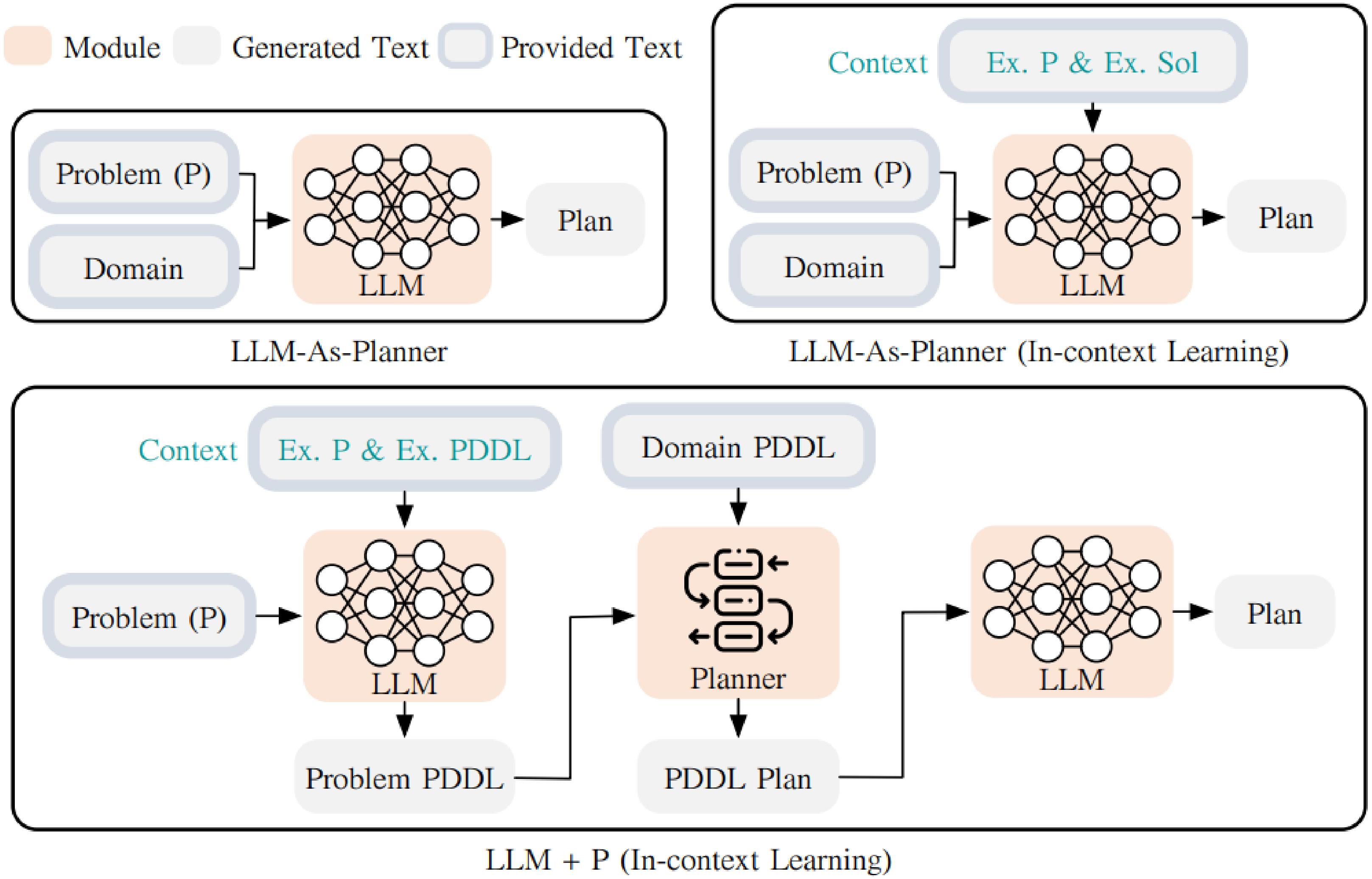
Task Planning	Prompt	
	#1 Task Planning Stage - The AI assistant performs task parsing on user input, generating a list of tasks with the following format: [{"task": task, "id", task_id, "dep": dependency_task_ids, "args": {"text": text, "image": URL, "audio": URL, "video": URL}}]. The "dep" field denotes the id of the previous task which generates a new resource upon which the current task relies. The tag "<resource>-task_id" represents the generated text, image, audio, or video from the dependency task with the corresponding task_id. The task must be selected from the following options: {{ *Available Task List* }}. Please note that there exists a logical connections and order between the tasks. In case the user input cannot be parsed, an empty JSON response should be provided. Here are several cases for your reference: {{ *Demonstrations* }}. To assist with task planning, the chat history is available as {{ *Chat Logs* }}, where you can trace the user-mentioned resources and incorporate them into the task planning stage.	
	Demonstrations	
	Can you tell me how many objects in e1.jpg?	[{"task": "object-detection", "id": 0, "dep": [-1], "args": {"image": "e1.jpg" }}]
	In e2.jpg, what's the animal and what's it doing?	[{"task": "image-to-text", "id": 0, "dep":[-1], "args": {"image": "e2.jpg" }}, {"task":"image-cls", "id": 1, "dep": [-1], "args": {"image": "e2.jpg" }}, {"task":"object-detection", "id": 2, "dep": [-1], "args": {"image": "e2.jpg" }}, {"task": "visual-quesrion-answering", "id": 3, "dep":[-1], "args": {"text": "what's the animal doing?", "image": "e2.jpg" }}]
	First generate a HED image of e3.jpg, then based on the HED image and a text "a girl reading a book", create a new image as a response.	[{"task": "pose-detection", "id": 0, "dep": [-1], "args": {"image": "e3.jpg" }}, {"task": "pose-text-to-image", "id": 1, "dep": [0], "args": {"text": "a girl reading a book", "image": "<resource>-0" }}]

THE LLM+P APPROACH

- The prompt approach has its drawbacks
- We can leverage existing solvers for planning problems and use LLMs to translate natural language to formal languages required by the solver
- This is the LLM+P paradigm

LLM+P: Empowering Large Language Models with Optimal Planning Proficiency

THE LLM+P APPROACH



PLANNING SUMMARY

- Planning is very important for LLM-based agents.
- We cover two approaches: directly prompting LLMs and use an external solver
- There are still many open research questions in LLM-based agentic planning.

TOOL USING

- One of the fundamentals between humans and animals is that human can create and use tools
- With tools, LLMs can enhance their capabilities and bypass their shortcomings (e.g., instead of computing 1+2 directly, use a calculator to compute 1+2)
- Notice: LLMs can't call the tools directly; it will instead output *how* to call the tools.
- We will discuss two flavors to train LLMs to learn to call the tools
  - SFT based approaches
  - RL based approaches

SFT APPROACH: TOOLFORMER

- Construct a dataset as follows
- And then we can fine tune the model using self-supervised loss function.

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in **[QA("Where was Joe Biden born?")]** Scranton, **[QA("In which state is Scranton?")]** Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or **[QA("What other name is Coca-Cola known by?")]** Coke, is a carbonated soft drink manufactured by **[QA("Who manufactures Coca-Cola?")]** the Coca-Cola Company.

Toolformer: Language Models Can Teach Themselves to Use Tools

DRAWBACKS OF TOOLFORMER

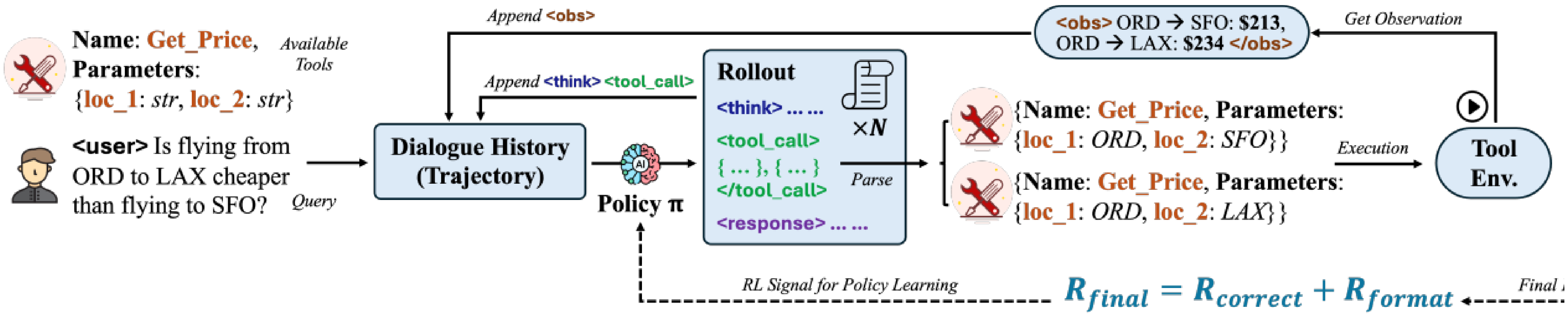
- SFT style fine tuning constrains what the model learns because it can only *mimic* things in the static dataset.
  - Static, pre-defined
  - Hard to adapt to novel scenarios
  - Just as how we post-train LM: pre-training and SFT is necessary but not enough, we further need RL!

RL APPROACH: TOOLRL

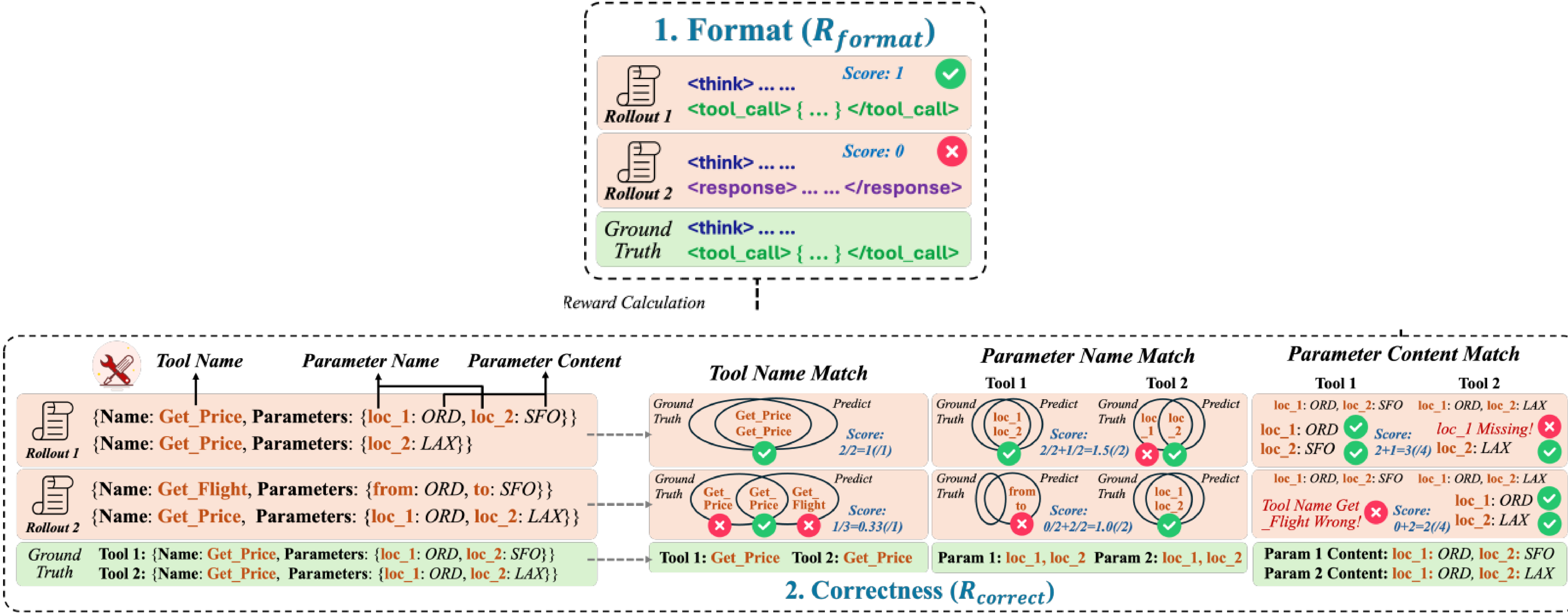
- High-level Idea:
  - Tool is increasingly used in Tool-integrated Reasoning
  - SFT based approaches show poor generalization
  - Reward contains two parts: correctness reward and format reward
  - Each trajectory contains three parts: <think>, <tool_call>, and <response>

ToolRL: Reward is All Tool Learning Needs

RL APPROACH: TOOLRL TRAINING RECIPE



TOOLRL REWARD FUNCTION



TOOL USING SUMMARY

- Before ToolRL, most works use SFT like approach to train LLMs to use tools
- ToolRL uses RL to train LLMs to use tools
- The reward includes correctness reward and format reward
- Future research directions:
  - On-the-fly tool using instead of pre-defined tool sets
  - Long horizon tool using

MEMORY

- There are two types of memory:
  - Short term memory that is associated with problem solving (~context in LLM)
  - Long term memory
    - Episodic memory: stores information about past events (?)
    - Semantic memory: stores language and knowledge (~parameters in LLM)
    - Hippocampus: turn short term memory into long term memory (?)
- Memory related operations:
  - Encoding
  - Storage/Reconsolidation
  - Retrieval

MEMORY RESEARCH QUESTIONS IN LLM

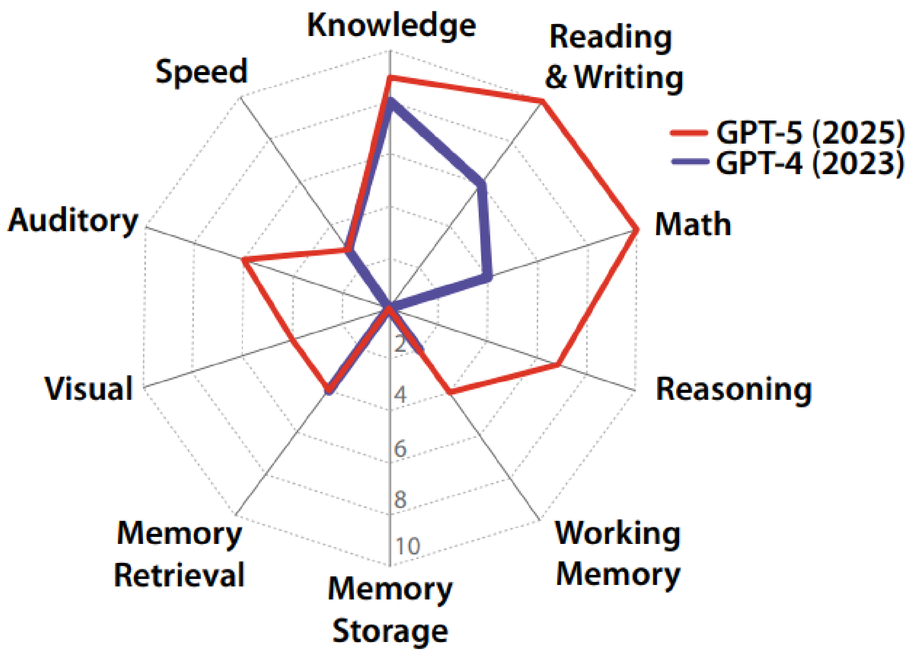
- How to manage short-term memory (a.k.a context) in LLM?
  - Context is very important in LLM. We ask questions about *what, where, when, and how* to put into context.
  - All prompting based works we've seen so far can be seen as context management.
  - There are two *major* differences between prompting in chatbox vs context in agentic tasks
    - Agentic tasks usually have much more #turns and each turn are highly coupled with each other (think of the ReAct loop).
    - Context in agentic tasks contains much more structural information (think of the action, obs pairs in ReAct style prompting)

MEMORY RESEARCH QUESTIONS IN LLM

- How to manage long-term memory?
  - LLMs have semantic memory that are acquired during pre-training. But what about episodic memory?
  - Humans can learn from past experiences (i.e. *continual learning* capabilities), how do LLMs achieve this?
  - Long-term memory could enable *personalized AI*: each person has its own GPT without fine tuning
- Memory can be stored in different forms:
  - Plain Text
  - Latent Embeddings
  - Structured Graph

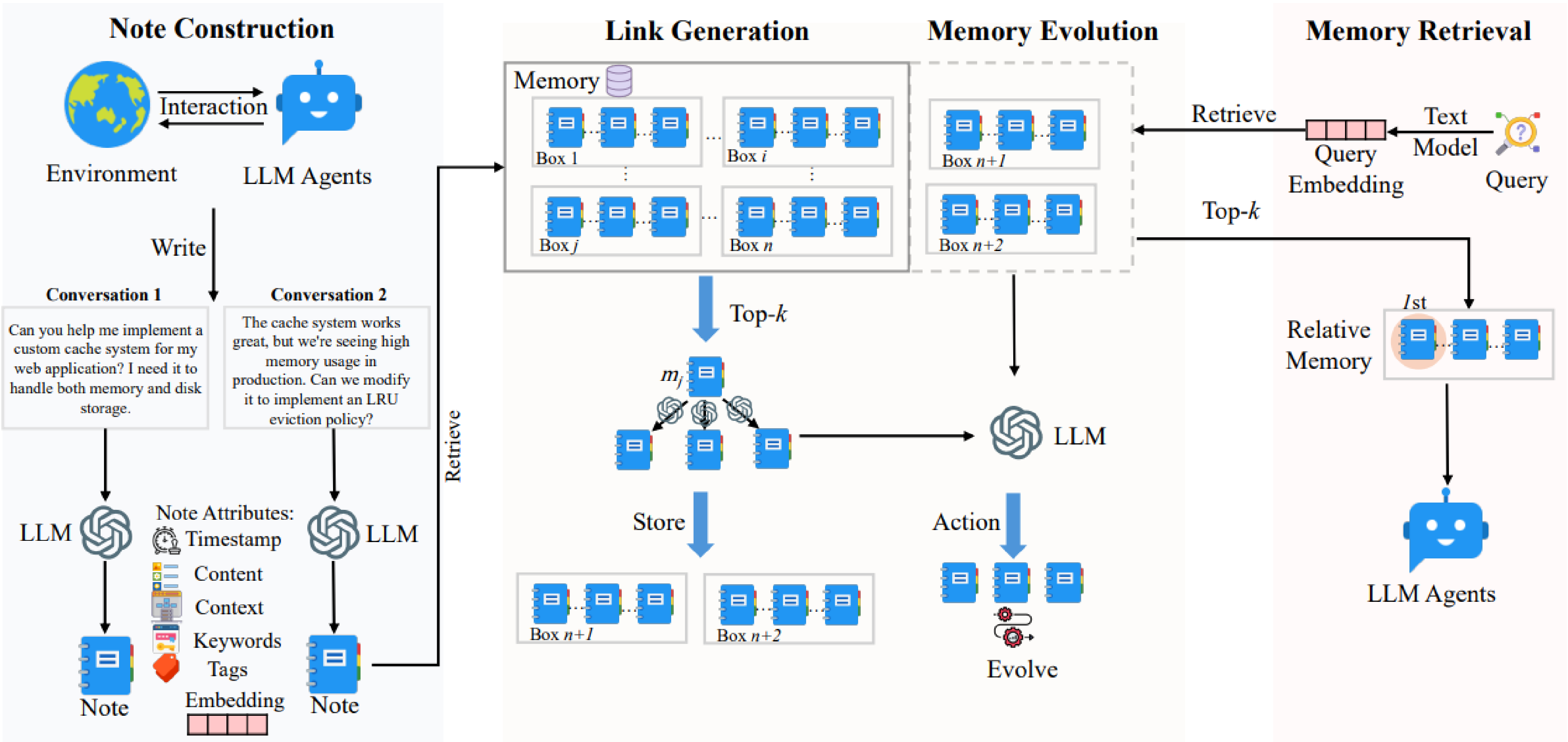
WHY IS MEMORY IMPORTANT?

- It could be the solution for self-evolving agents.
- Current AI systems are bad at this.
- Context is the key to many problems.



A Definition of AGI

SHORT TERM MEMORY: A-MEM

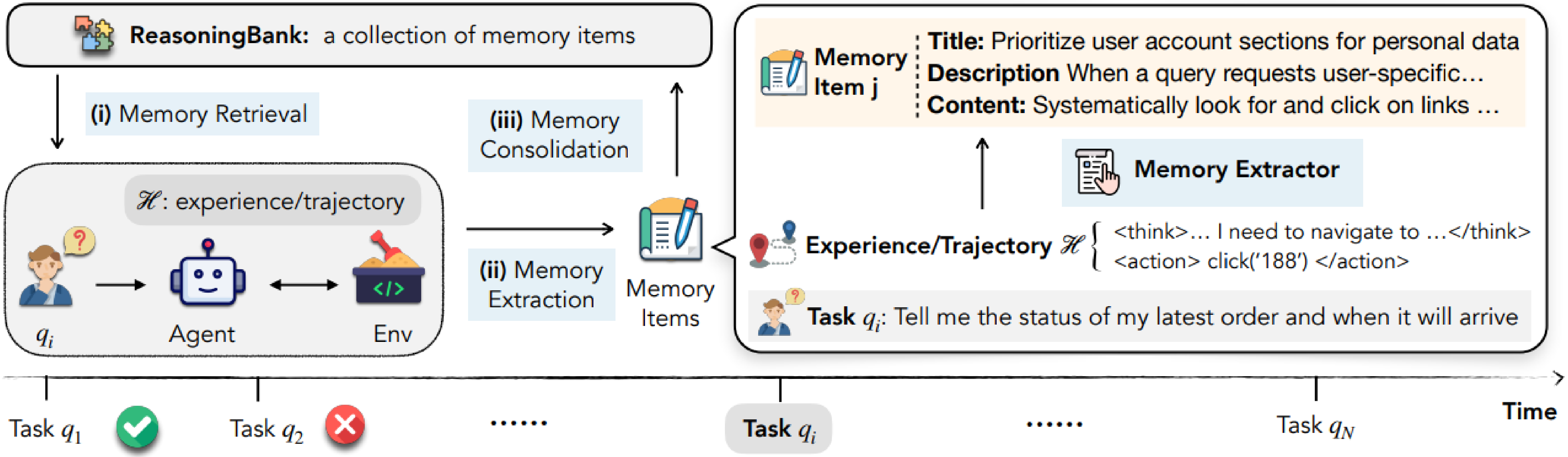


A-MEM: Agentic Memory for LLM Agents

SHORT TERM MEMORY: A-MEM

- For each turn, it goes through three steps:
  - **Note construction** *(summarize context, keywords, tags)*
  - **Link construction** *(link new note with old note)*
  - **Memory evolution** *(update old notes)*

LONG TERM MEMORY: REASONING BANK



LONG TERM MEMORY: REASONING BANK

- When we execute a task, we first retrieve relevant memory from the bank using embedding similarity
- We include the memory in the prompt
- After the task finishes, we construct the memory based on whether the task success or fails
- We then consolidate the memory into the bank.

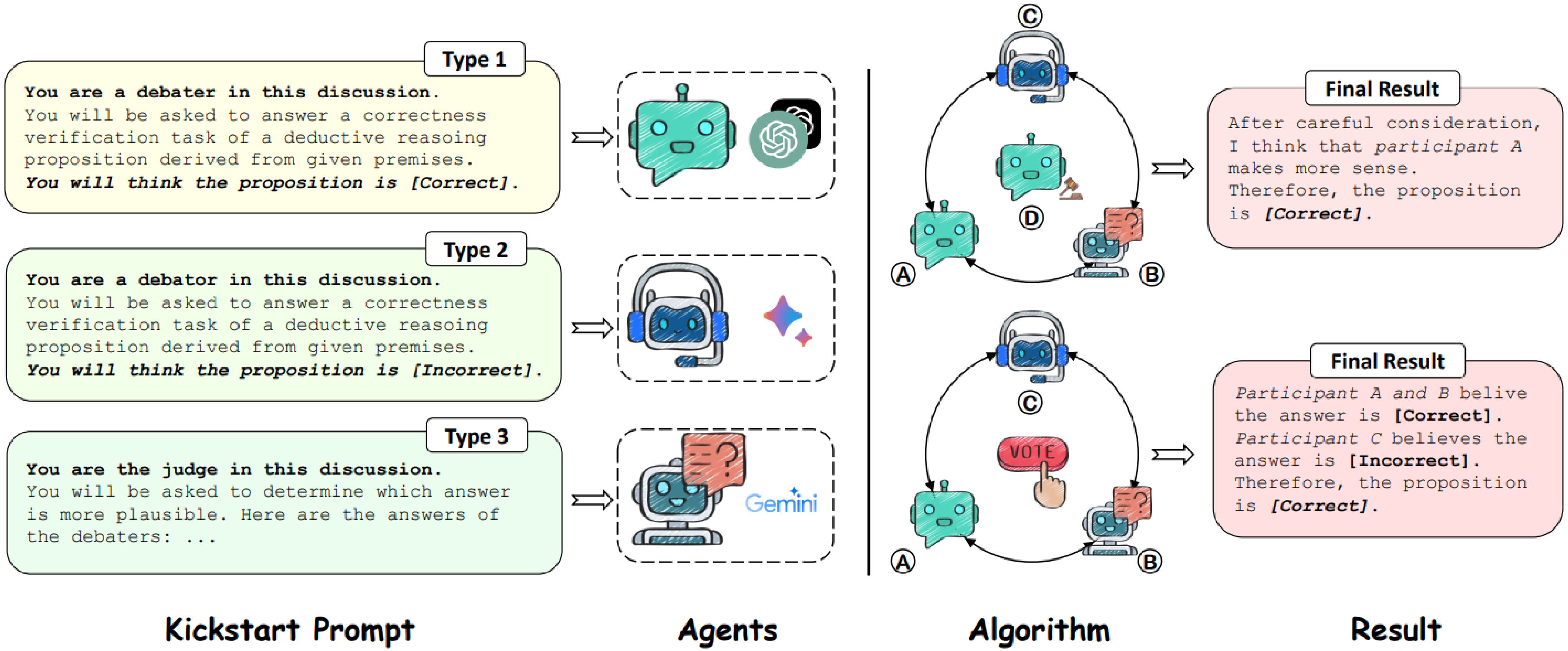
MEMORY SUMMARY

- Memory has two types: short-term memory and long-term memory
- Short-term memory is roughly equal to context management in LLM
- Long-term memory (semantic memory) is acquired during LLM pretraining.
- Long-term memory (episodic memory) is beneficial for continual learning agents.
- Long-term memory could enable personalized AI.
- Memory has different forms: plain text, embedding, graph.

MULTI-AGENT

- Sometimes we require multiple agents (where each agent is backed by an LLM) to work together to solve a task.
- Pros
  - Each agent only needs to solve a simple, specialized task
  - Each agent needs to consider a smaller context
  - May increase the efficiency because each agent can in principle run in parallel.
- Cons
  - Exponentially more design choices
    - Agent architecture
    - Communication protocol
    - Context management
  - Cost

CONQUER-AND-MERGE DISCUSSION



Rethinking the Bounds of LLM Reasoning: Are Multi-Agent Discussions the Key?

CONQUER-AND-MERGE DISCUSSION

- There are three stages:
  - During the ***discussion stage***, each agent can talk with other agents in the same group.
  - Given a reasoning question, in each round, each agent will generate an answer and an explanation.
  - Agents can access explanation within the same group from previous round, with answers only from different groups.
  - During the ***voting stage***, agents try to reach a consensus.
  - During the ***decision stage***, another LLM will serve as a judge to decide the result.

CHAIN-OF-AGENTS

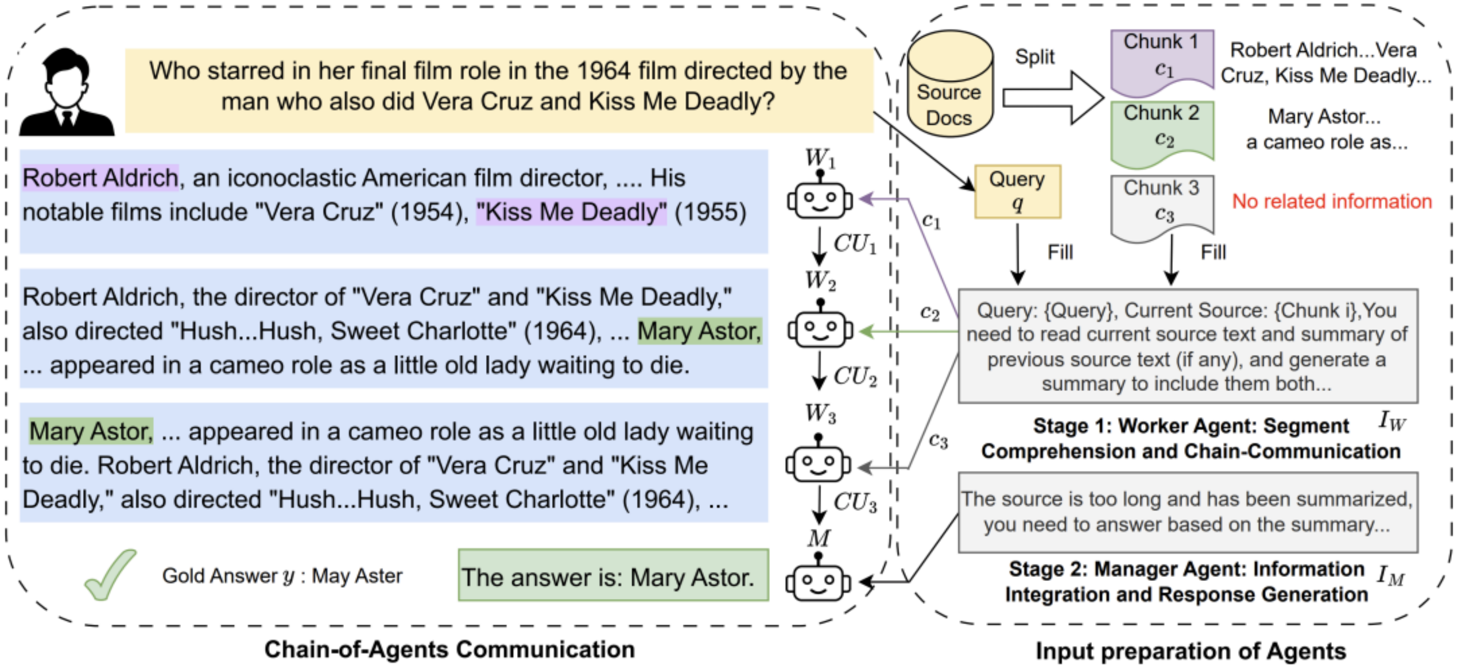


Figure 1: Overview of Chain-of-Agents, a training free, task agnostic, and highly-interpretable framework that harnesses multi-agent collaboration for long-context tasks. It consists of multiple worker agents who sequentially communicate to handle different segmented portions of the text, followed by a manager agent who synthesizes these contributions into a coherent final output.

Chain of Agents: Large Language Models Collaborating on Long-Context Tasks

CHAIN-OF-AGENTS

- Answer a query from a large pool of documents
- Split documents into chunks
- Each agent will read the chunk, previous summarizations, and the query, summarize all contents, and pass it into next agents.
- A manager agent will generate the final response

MULTI-AGENT SUMMARY

- Multi-agent may solve more complex problems at the cost of using more LLMs
- The design space is huge
- We discuss two potential architectures:
  - CMD
  - Chain-of-Agents

APPLICATIONS AND EVALUATION

- Many applications: science agents, data science agents, GUI/web agents, search agents, recommendation agents...
- Evaluation is a key research question.

CONCLUSIONS

- LLM-based agents are getting very popular.
- Core capabilities: reasoning, planning, memory, self-evolving, self-reflection, tool using, multi-agent.
- Some important research directions
  - How to better evaluate agentic tasks?
  - How to generate better synthetic data to train better agentic models?
  - How do we make agents safe and reliable?
  - How can we scientifically evaluate the progress of agentic tasks?

QUESTIONS?