Machine Learning Model Management and Inference (MLMMI)
05 AI Agents and Compound AI Systems

Dr. Arnab Phani
BIFOLD, TU Berlin
DEEM Lab

27.05.2026

Announcements

- **No Lecture on June 03**
  - I will be in India to attend SIGMOD
- **Invited Talk 2: June 10**
  - Matthias Seeger (https://mseeger.github.io/)
  - Please try to attend
- **Open Student Assistant Position (w/ Olga)**
  - New benchmark for evaluating MLE agents
  - Non-contaminated tasks (tabular, multi-modal)
  - https://deem.berlin/#jobs-204356

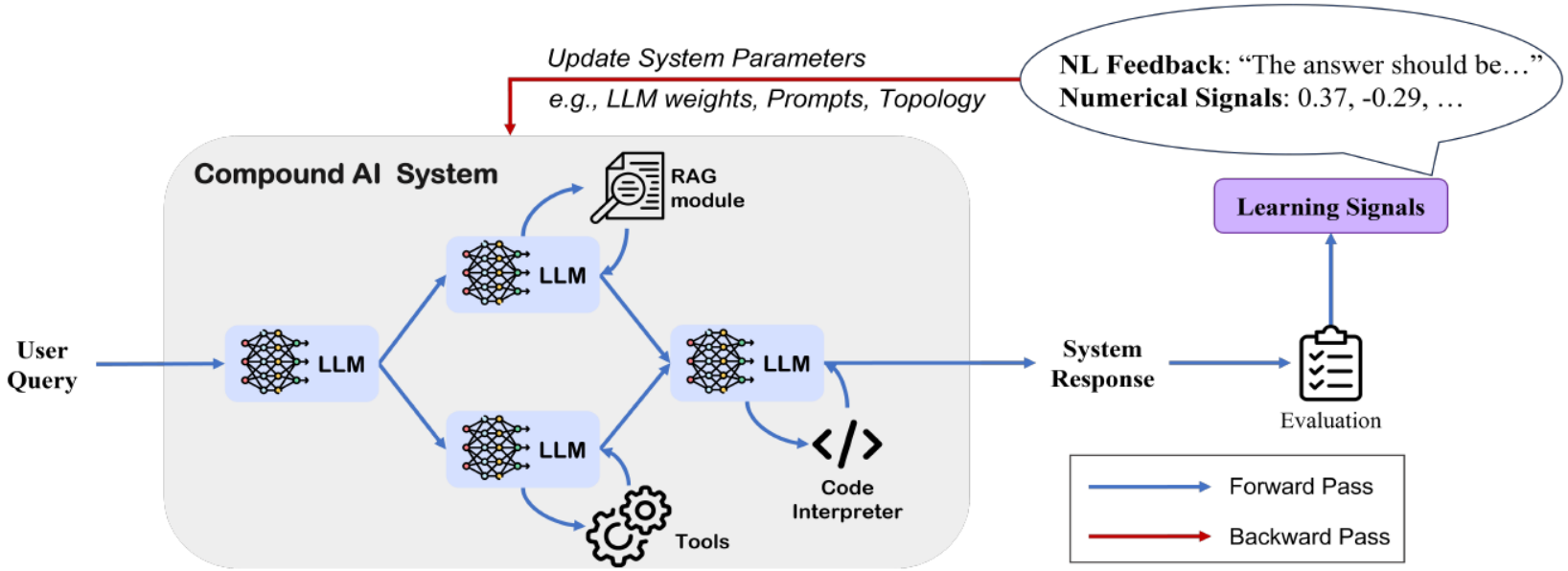
Matthias W. Seeger received a Ph.D. from the School of Informatics, Edinburgh university, UK, in 2003 (advisor Christopher Williams). He was a research fellow with Michael Jordan and Peter Bartlett, University of California at Berkeley, from 2003, and with Bernhard Schoelkopf, Max Planck Institute for Intelligent Systems, Tuebingen, Germany, from 2005. He led a research group at the University of Saarbruecken, Germany, from 2008, and was assistant professor at the Ecole Polytechnique Federale de Lausanne from fall 2010. He joined Amazon as machine learning scientist in 2014. He received the ICML Test of Time Award in 2020.

Agenda

- **Compound AI Systems and Use Cases**
- **AI Agents**
- **Agent Context and Memory Management**
- **Agent Tooling**
- **RAG**
- **Agents Use Cases**

Compound AI Systems and Use Cases

Compound AI Systems



- **Overview**
  - Systems that tackle certain tasks using multiple interacting components
  - Scaling training data vs. building a compound system
  - Systems can be dynamic. Models are static with fixed knowledge
  - Improving control, performance goal, and trust is easier with systems
  - Large design space: models, components, prompts, …
  - Composition frameworks: LangChain, LlamaIndex, …

Text2SQL systems
QA / Enterprise search
LLM-assisted data analysis

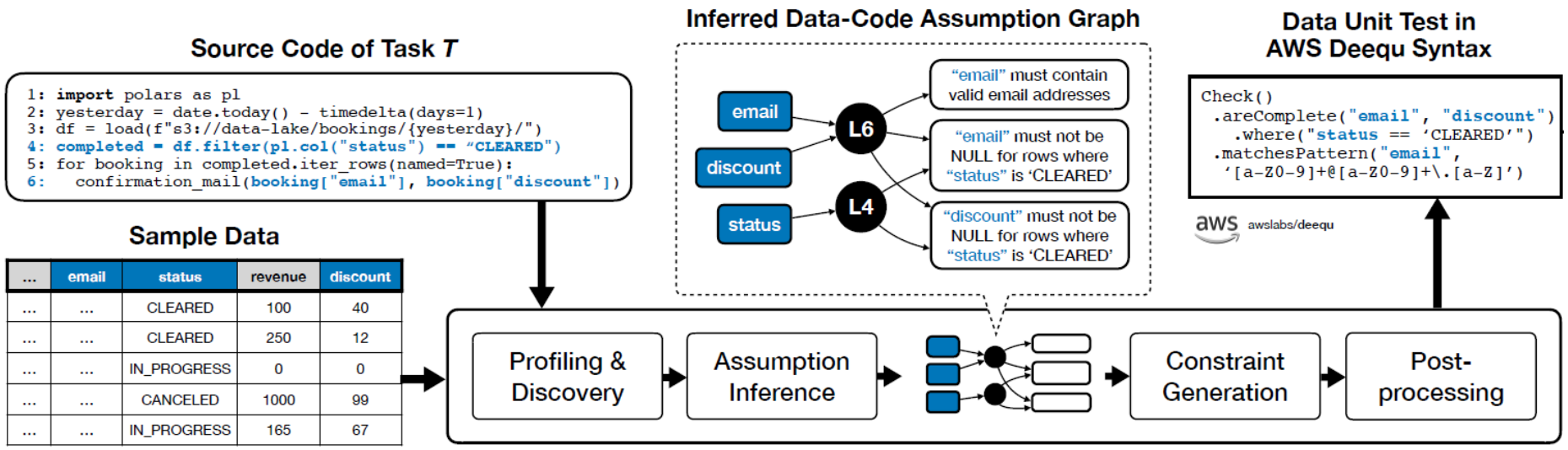
* https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/
* Compound AI Systems Optimization: A Survey of Methods, Challenges, and Future Directions. ACL. 2025.

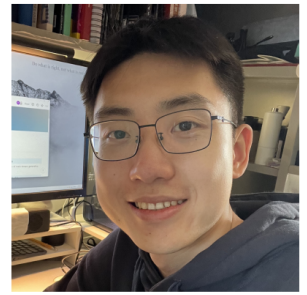
Compound AI System Optimization

Methods	Structural Flexibility	Learning Signals	Component Options	System Representations
TextGrad (2025)	Fixed	NL Feedback	LLM	Graph†
Trace (2024)	Fixed	NL Feedback	LLM	Graph†
AIME (2024)	Fixed	NL Feedback	LLM	Graph†
Revolve (2024b)	Fixed	NL Feedback	LLM	Graph†
GASO (2024a)	Fixed	NL Feedback	LLM	Graph†
LLM-AutoDiff (2025)	Fixed	NL Feedback	LLM; RAG	Graph
DSPy (2023)	Fixed	Numerical Signals^{a}	LLM	Graph†
MIPRO (2024)	Fixed	Numerical Signals^{a}	LLM; RAG	Graph
BetterTogether∗ (2024)	Fixed	Numerical Signals^{a,b1}	LLM; RAG	Graph†
SiriuS∗ (2025)	Fixed	Numerical Signals^{b1}	LLM	Graph
MAPoRL∗ (2025)	Fixed	Numerical Signals^{b2}	LLM	Graph
SysDPO∗ (2025a)	Fixed	Numerical Signals^{b3}	LLM; IGM	Graph†
Multiagent Finetuning∗ (2025)	Fixed	Numerical Signals^{b1}	LLM	Graph
Agent Symbolic Learning (2024)	Flexible	NL Feedback	LLM; RAG	Graph†
ADAS (2024)	Flexible	NL Feedback	LLM; CI	Python Code
AFlow (2024a)	Flexible	NL Feedback	LLM; CI	Python Code
MASS (2025)	Flexible	NL Feedback	LLM	Graph
DebFlow (2025)	Flexible	NL Feedback	LLM	Graph
DyLAN (2024)	Flexible	Numerical Signals^{a}	LLM	Graph
GPTSwarm (2024)	Flexible	Numerical Signals^{b2}	LLM; Tools	Graph†
AutoFlow∗ (2024)	Flexible	Numerical Signals^{b2}	LLM	NL Programs
MaAS (2025)	Flexible	Numerical Signals^{b2}	LLM; Tools; CI	Graph†
ScoreFlow∗ (2025b)	Flexible	Numerical Signals^{b3}	LLM; CI	Python Code
MAS-GPT∗ (2025)	Flexible	Numerical Signals^{b1}	LLM; CI	Python Code
W4S∗ (2025)	Flexible	Numerical Signals^{b2}	LLM; CI	Python Code
FlowReasoner∗ (2025)	Flexible	Numerical Signals^{b1,b2}	LLM; CI	Python Code

- **Overview**
  - Large space: prompts, components, tools usage, cost; e.g., DSPy, GEPA
  - Active research area

PrismaDV: Task-aware Data Unit Test Generation



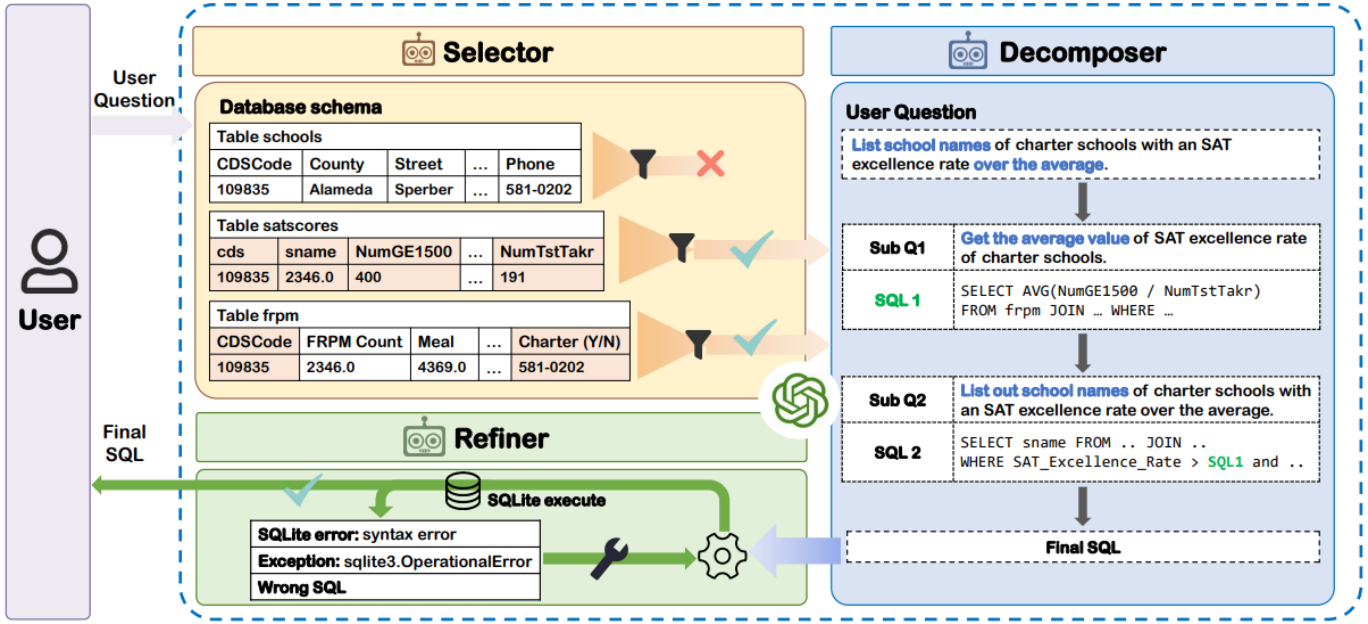


Compound AI System

- **PrismaDV**
  - Jointly analyzes downstream task code and data
  - Builds a "**data-code assumption graph**" linking data columns to code-level assumptions
  - Synthesizes executable data unit tests for deployment

https://arxiv.org/pdf/2604.21765

MAC-SQL: A Multi-Agent Framework for Text-to-SQL



- **Overview**
  - Text2SQL is difficult for large databases and complex user questions
  - MAC-SQL is an LLM-based multi-agent collaborative framework
  - Selector: decomposes a large database into a smaller sub-database; reduce irrelevant schema items; reduce text length (context length, API cost)
  - Decomposer: generate a series of intermediate steps (i.e. sub-questions and SQLs); enhance reasoning
  - Refiner: detect and automatically correct SQL errors (without executing)

* MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. 2025. COOLING.

AI Agents and Use Cases

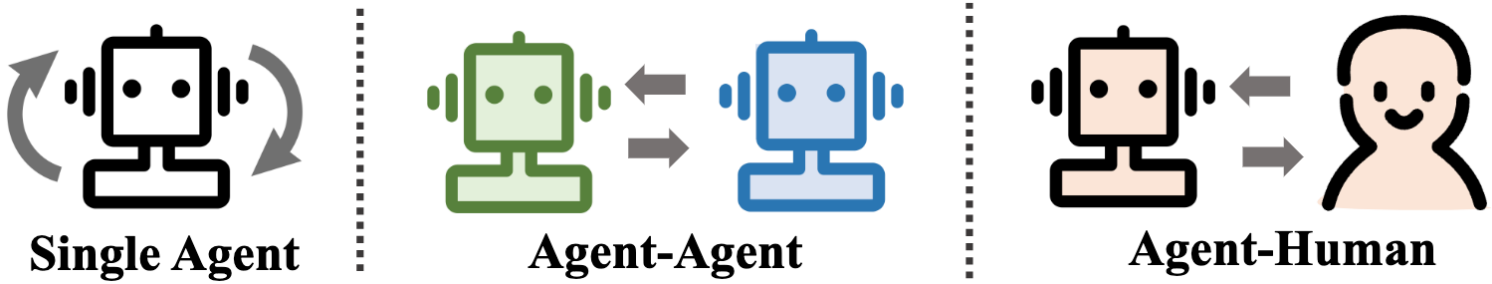
AI Agents

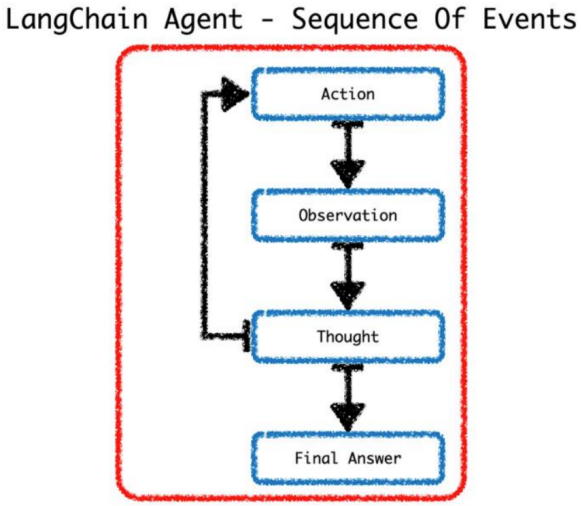
- **What is an AI Agent?**
  - An autonomous entity that can operate on its own
  - What's inside: a program involving model calls and tool calls
  - Difference from an LLM: make decisions, take actions
  - Compound AI Systems: static workflow, fixed DAG, orchestration-centric, mostly deterministic.
  - Agent: dynamic workflow, adaptive execution, decision-centric, autonomous.

- **Why LLM Agents Stand Out?**
  - Language mastery and seamless user interaction
  - Decision-making and reasoning abilities
  - Flexibility; they can be molded for diverse applications
  - They can collaborate with humans and other agents

* https://cseweb.ucsd.edu/~yiying/cse291a-fall25/reading/ai-agents.pdf

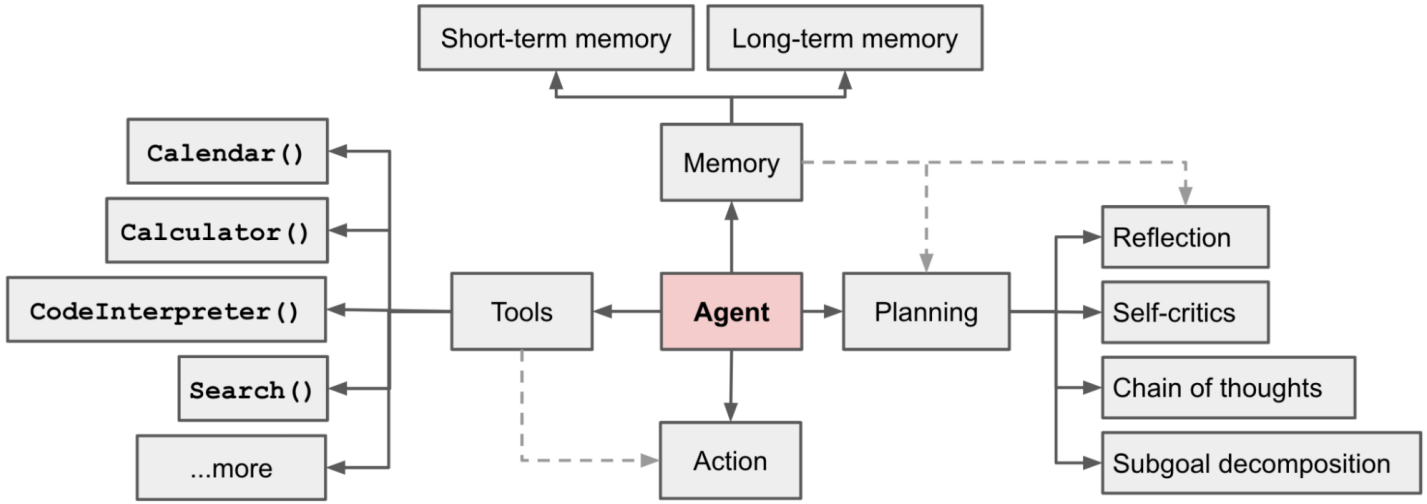
Categories of Agents





- **Categories**
  - Single-agent applications: work as personal assistants in day-to-day activity
  - Multi-agent systems: agents interact with each other to solve a complex task
  - Human-agent cooperation: agents interact with humans; e.g., ChatGPT

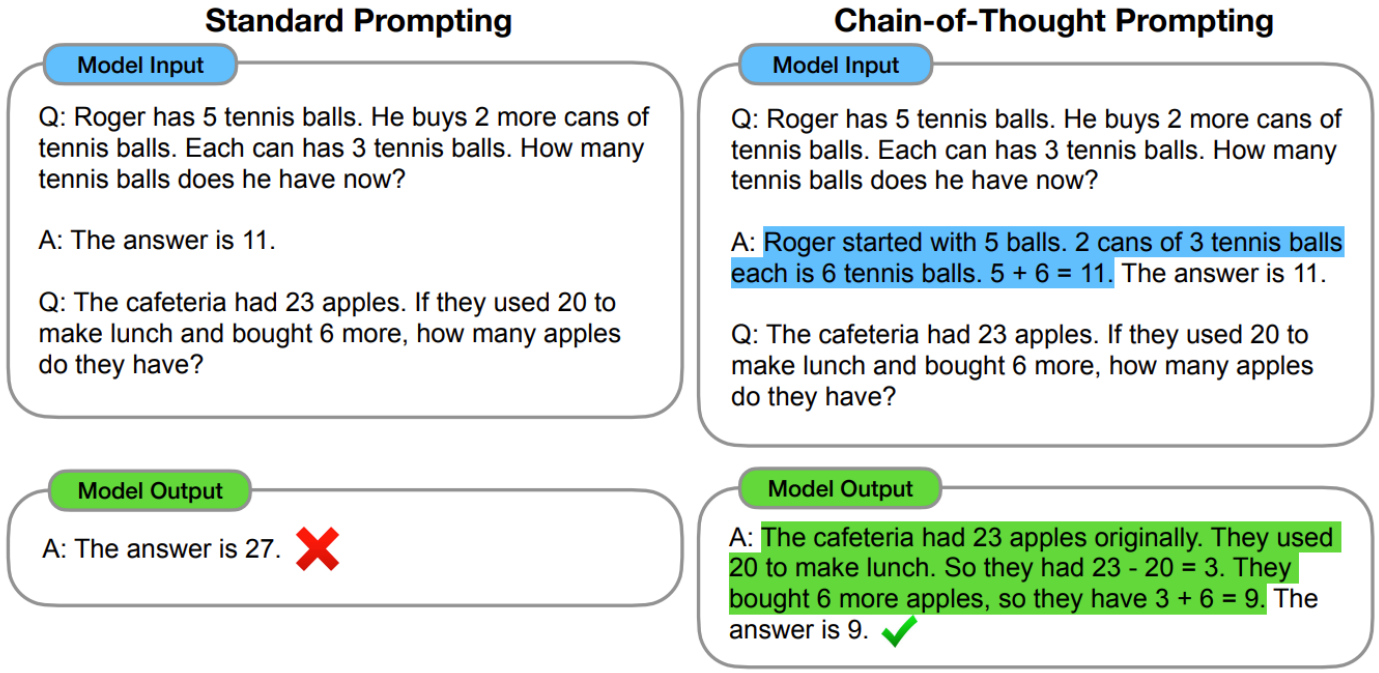
Planning



- **Planning**
  - Subgoal and decomposition: The agent breaks down large tasks into smaller, manageable subgoals, enabling efficient handling of complex tasks.
  - Reflection and refinement: The agent can do self-criticism and self-reflection over past actions, learn from mistakes and refine them for future steps, thereby improving the quality of final results.

* https://lilianweng.github.io/posts/2023-06-23-agent/

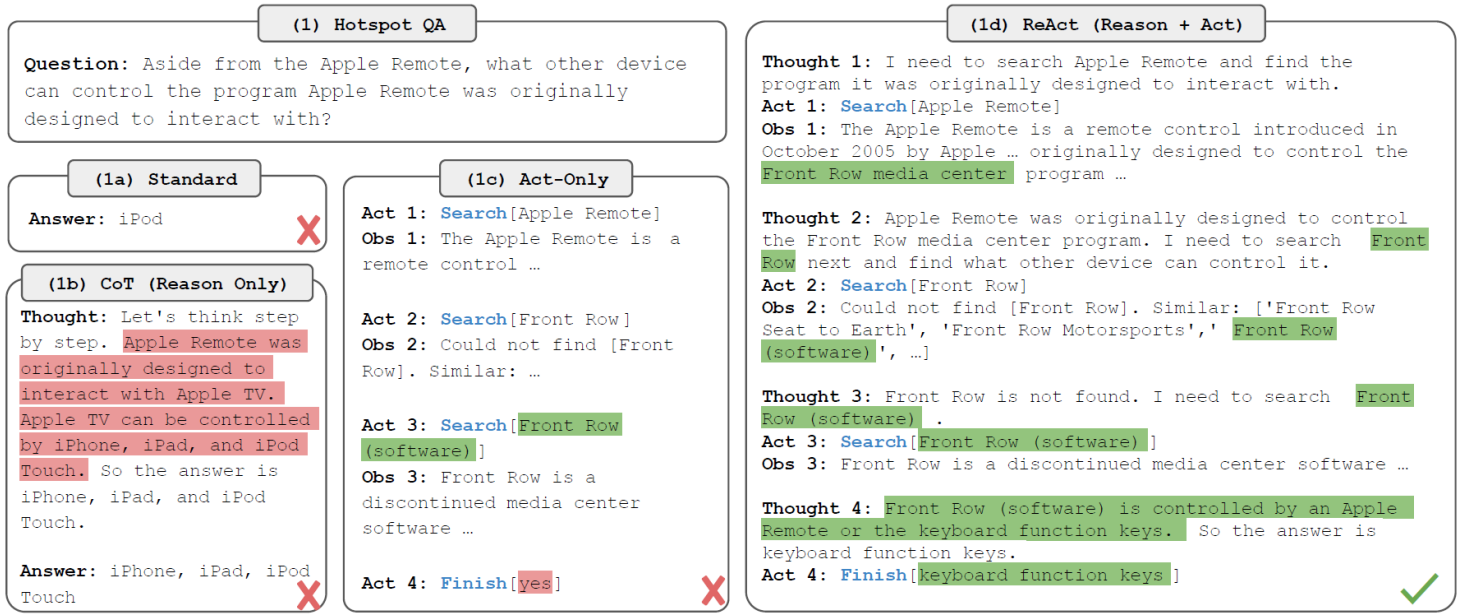
Task Decomposition: Chain of Thought



Chain of Thought (CoT) has become a standard prompting technique for enhancing model performance on complex tasks. The model is instructed to *think step by step* to utilize more test-time computation to **decompose hard tasks into smaller and simpler steps**. CoT transforms big tasks into multiple manageable tasks and shed lights into an interpretation of the models thinking process.

* Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. NeurIPS 2022.

ReAct




```
> Entering new AgentExecutor chain...
 I need to find out who Olivia Wilde's
Action: Search
Action Input: "Olivia Wilde boyfriend"
Observation: Olivia Wilde started datin
Sudeikis – see their relationship timel
Thought: I need to find out Harry Style
Action: Search
Action Input: "Harry Styles age"
Observation: 29 years
Thought: I need to calculate 29 raised
Action: Calculator
Action Input: 29^0.23
Observation: Answer: 2.169459462491557
```

- **Overview**
  - Reasoning-only: Thought -> Thought -> Answer. Hallucinate as cannot fetch new info
  - Acting-only: Action -> Action -> Answer. Doesn't think, cannot handle failed actions
  - ReAct: Reasoning to act; combine reasoning and acting
  - Few-shot prompting; Thought -> Action -> Observation
  - Fine-tune small models from ReAct logs generated by large models

```
llm = OpenAI(model_name="text-davinci-003" ,temperature=0)
tools = load_tools(["google-serper", "llm-math"], llm=llm)
agent = initialize_agent(tools, llm, agent="zero-shot-react-description"
```

LangChain ReAct Usage

* ReAct: Synergizing Reasoning and Acting in Language Models. ICLR 2023.

Agent Context and Memory

- **Why Do Agents Need Memory?**
  - A standard LLM's memory is limited to its context window
  - Once the conversation exceeds this limit, earlier information is lost
  - Agents need memory for complex tasks
    - Personalization: Remember preferences ("I'm allergic to peanuts")
    - Long-term Tasks: Managing a project over weeks
    - Learning: Not making the same mistake twice
    - Stateful Interaction: Picking up conversation where it was left off

Context-unaware (stateless)

User: "I'm allergic to peanuts."
Agent: "Noted. How can I help?"
(5 minutes later)
User: "Find me a good Thai restaurant."
Agent: "Here is a list of Thai restaurants..."
(This list includes places that use peanut oil, as the allergy info is "out of context")

- **Two Types of Memory**
  - In-context Memory (short-term): Information fed directly into the prompt
  - External Memory (long-term): Information stored outside the model (e.g., database)

Memory Management

- **Spectrum of Memory Management**
  - Finite Context (Stateless): No long-term memory. (e.g., Basic API call)
  - Simple Context (Non-RAG): Simple, programmatic rules (Summaries)
  - Passive Retrieval (RAG): Saves and retrieves context in a vector db;
    external tool "passively" adds info to the context.
  - Active Management (Agentic): The LLM itself controls its memory via function calls.

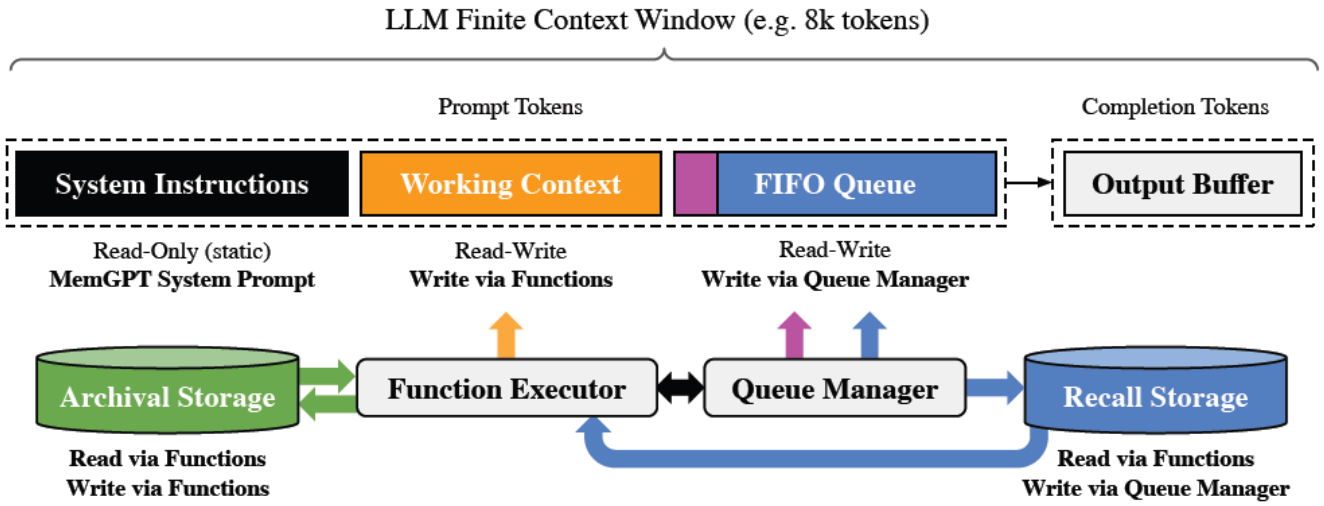
- **Simple Context Management**
  - Sliding Window (FIFO): Keep only the last k messages/tokens
    - Pro: Simple fast, no extra cost
    - Con: Cannot recall longer history, fails for long-running tasks
  - Summarization: Periodically summarize the conversation so far using an LLM
    - Pro: Compress history; keeps gist of long conversations
    - Con: Information loss, latency; increased cost due to LLM cals

Memory Management, cont.

- **Passive Retrieval**
  - Systems passively retrieve information for the LLM
  - Offline: Docs -> Chunks -> Embeddings -> Vector DB
  - Online: Query -> Embed -> Vector Search -> Top-k Chunks -> Add to Prompt
  - Limitation: RAG is stateless; doesn’t know what’s in the context

- **Active (Agentic) Memory**
  - Let agents decide what to remember, what to forget, and what to retrieve
  - Agents call functions to manage its own memory
  - Sometimes also called Agentic RAG
  - Many recent works in this direction

MemGPT: Towards LLMs as Operating Systems



- **MemGPT**
  - Virtual memory paging from OS; illusion of infinite memory
  - Main Context: The prompt
    - System prompt (read-only), working context (read/write scratchpad for context relevant to current task), FIFO queue (recent conversation history)
  - External Context
    - Recall storage (store all interactions, searchable via vector search), archival storage (stores longer-term context like documents)

* MemGPT: Towards LLMs as Operating Systems. 2023. arXiv.

MemGPT: Towards LLMs as Operating Systems, cont.

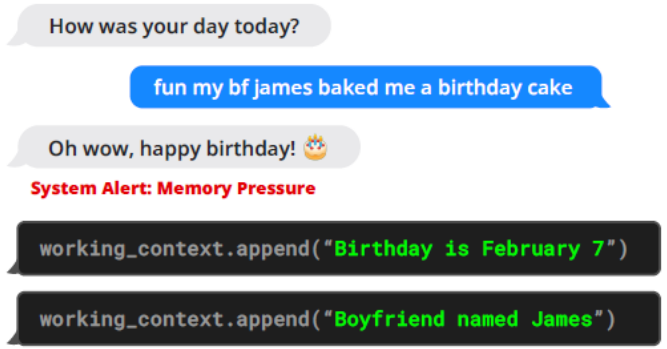


Figure 1. MemGPT (left) writes data to persistent memory after it receives a system alert about limited context space.

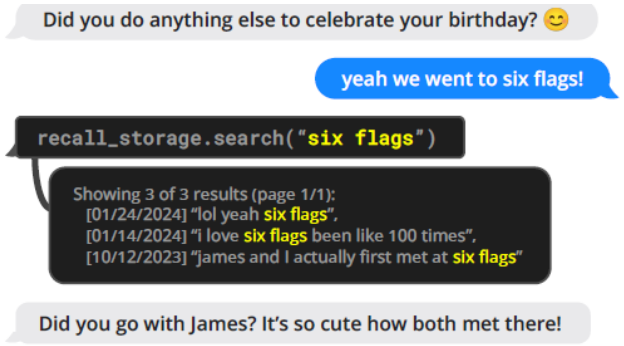


Figure 2. MemGPT (left) can search out-of-context data to bring relevant information into the current context window.

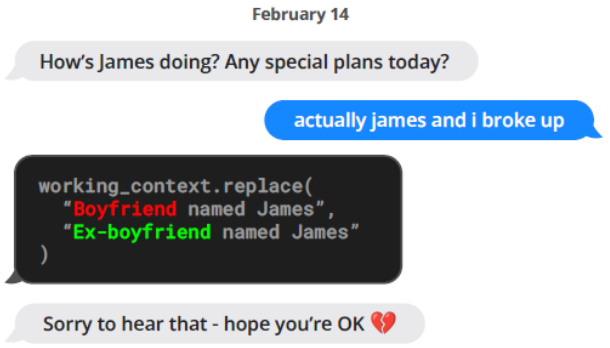


Figure 4. An example conversation snippet where MemGPT (left) updates stored information. Here the information is stored in working context memory (located within the prompt tokens).

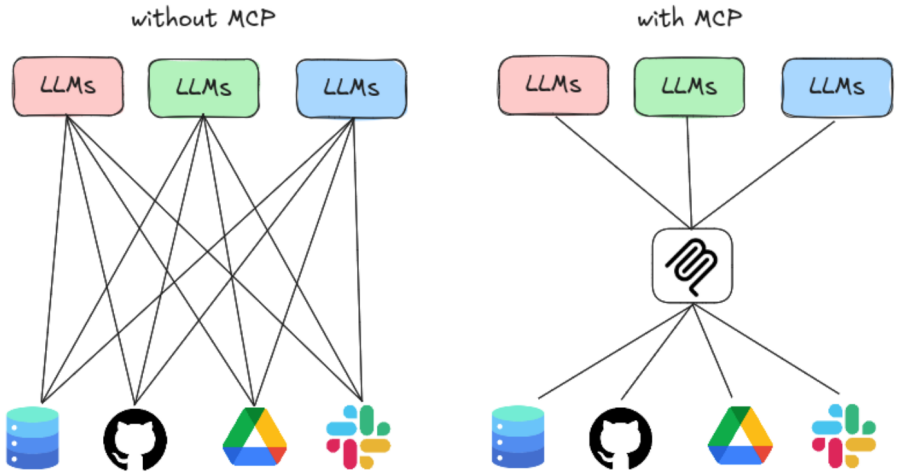
- **Basic Flow**
  - The LLM processes the event and its current Main Context (RAM).
  - It might think: "The user is asking about 'Project X', which is not in my working context. I need to 'page in' that data from my recall storage."
  - The LLM emits a function call to its "OS" (the MemGPT system).
  - The system executes it by retrieving the info, and pages it (i.e., insert) into the Main Context.
  - The LLM runs again, now with the new info, and generates a response
  - Append the generated answer to FIFO queue

AI Agent Tooling

- **“Brain in a Jar” Problem**
  - LLM’s information frozen at the time of training
  - No real-time access; e.g., cannot check stock prices, weather, news
  - Cannot perform actions; e.g., book a flight, send an email, run code
  - When pushed beyond their knowledge, LLMs hallucinate

- **How Do Models Call Tools?**
  - Instruction Tuning: Fine-tuned on a dataset of instructions, tool call, tool output, final answer; fast but cannot easily add new tools without re-tuning
  - System Prompt: A system prompt instructs LLMs to call tools with a ReAct loop; flexible as new tools can be added just by describing, but less reliable than fine-tuning
  - In-context Learning: Provide a few examples; flexible but less reliable and need good examples

Model Context Protocol

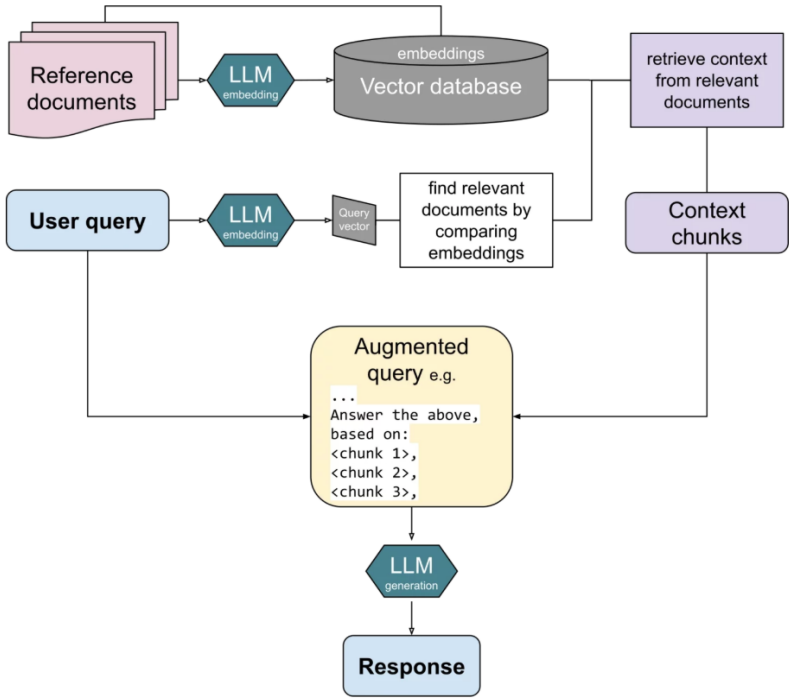



```
// 1. Define the tool the AI will see
server.setRequestHandler(ListToolsRequestSchema, async () => ({
  tools: [{
    name: "get_weather",
    description: "Get the current weather for a location",
    inputSchema: {
      type: "object",
      properties: { location: { type: "string" } },
      required: ["location"]
    }
  }]
}));
```

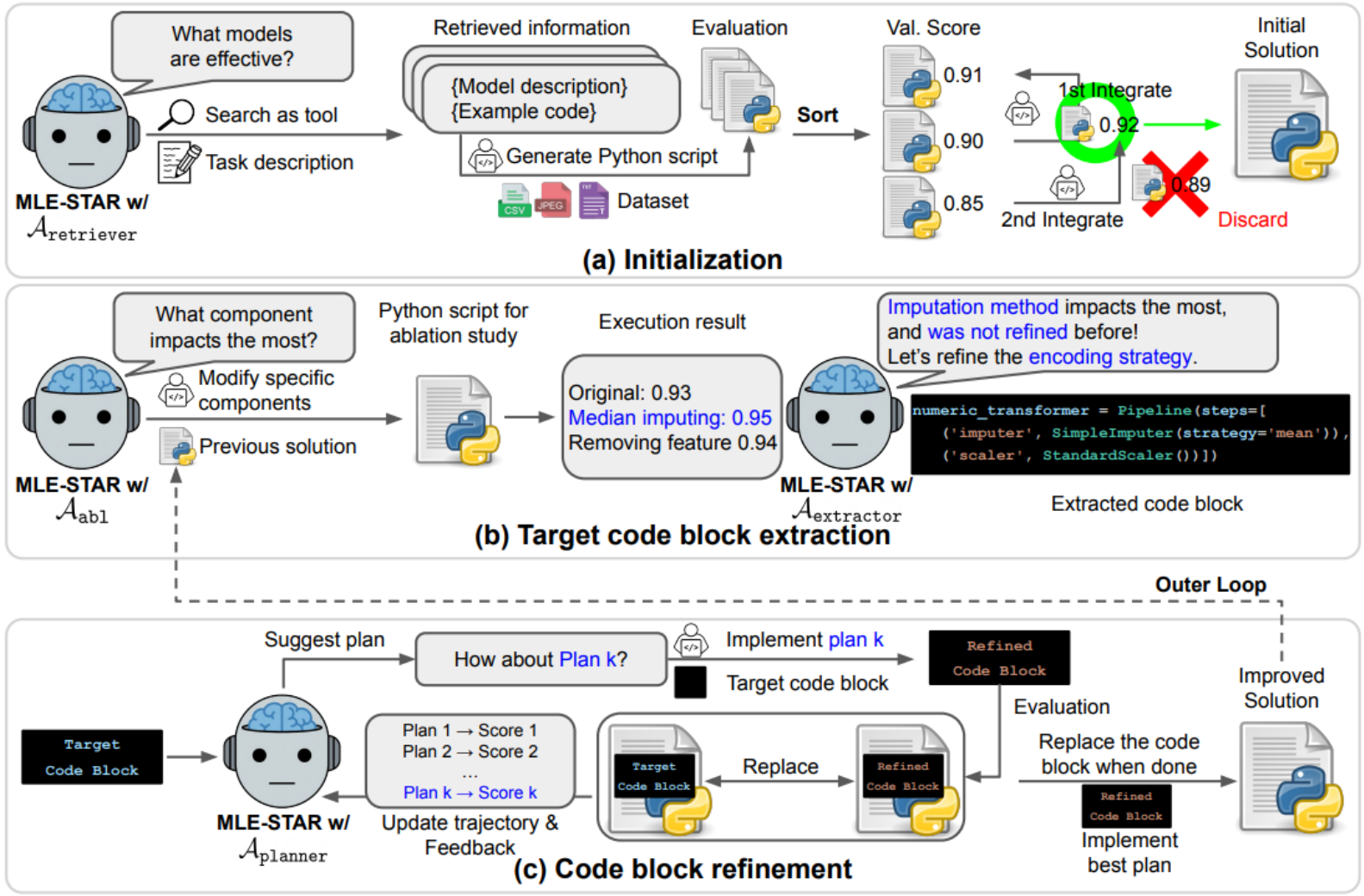
- **Overview**
  - N Models: (GPT-4, Claude 3, Gemini, Llama 3...), M Tools: (Google, Stripe, Slack, JIRA, internal APIs...) all have different authentication, schemas, and endpoints; custom code for every model-tool combination
  - MCP standardizes the LLM-tool communication; client-server model
  - MCP Client: the agent that needs to call tool/data
  - MCP Server: a service to expose external tools and data sources

Retrieval-Augmented Generation (RAG)

- **Why RAG?**
  - LLM's knowledge is frozen at the time of training
  - No proprietary info; LLMs are trained with only public data; companies' growing private documents
  - Easier to hallucinate when asked about knowledge outside training set
- **RAG System**
  - Preprocess data by chunking and embedding, stored in vector DB
  - Retrieve: convert query into embedding
  - Get the chunks from DB with highest similarity with the embedding
  - Summarize and combine retrieved chunks with existing context
  - Feed to downstream LLMs
- **RAG vs. Fine-Tuning**
  - RAG adds external knowledge at answer time, fine-tuning teaches models a task, style, or response pattern
  - RAG has higher data freshness
  - Orthogonal to each other; can employ both



MLE-STAR: MLE Agent via Search and Targeted Refinement



* MLE-STAR: Machine Learning Engineering Agent via Search and Targeted Refinement. 2025. NeurIPS.

Summary

- LLMs are static with fixed knowledge
- Compound AI systems consist interacting LLM-based components
- Compound AI systems vs. AI agents vs. multi-agent systems
- Memory management techniques to increase context memory
- Tool usage and MCP server to standardize LLM-tool communication
- RAG vs. fine-tuning

Please read the recommended papers

Projects

- **Project Selection**
  - Please complete if not done
- **Project Presentation**
  - July 15
  - 10min presentation + 10min QA for each group
  - Everybody should be present to ask questions
- **Project Report and Code**
  - Deadline July 29

Responses	1	2	3	4	5	Total
Project 1	2 (11%)	5 (28%)	**6** (33%)	3 (17%)	2 (11%)	**(18)**
Project 2	5 (28%)	3 (17%)	**6** (33%)	4 (22%)	0	**(18)**
Project 3	3 (17%)	2 (11%)	1 (6%)	**7** (39%)	5 (28%)	**(18)**
Project 4	5 (28%)	5 (28%)	2 (11%)	0	**6** (33%)	**(18)**
Project 5	3 (17%)	3 (17%)	3 (17%)	4 (22%)	**5** (28%)	**(18)**

17 unique submissions