Lecture 8.3 - LLM Agents

Generative AI Teaching Kit

This lecture

- Reasoning with LLMs
- LLM Agents
- Aligning Agents with Guardrails
- NeMo Guardrails and Meta Llama Guard

Reasoning with LLMs

Thinking slowly and carefully

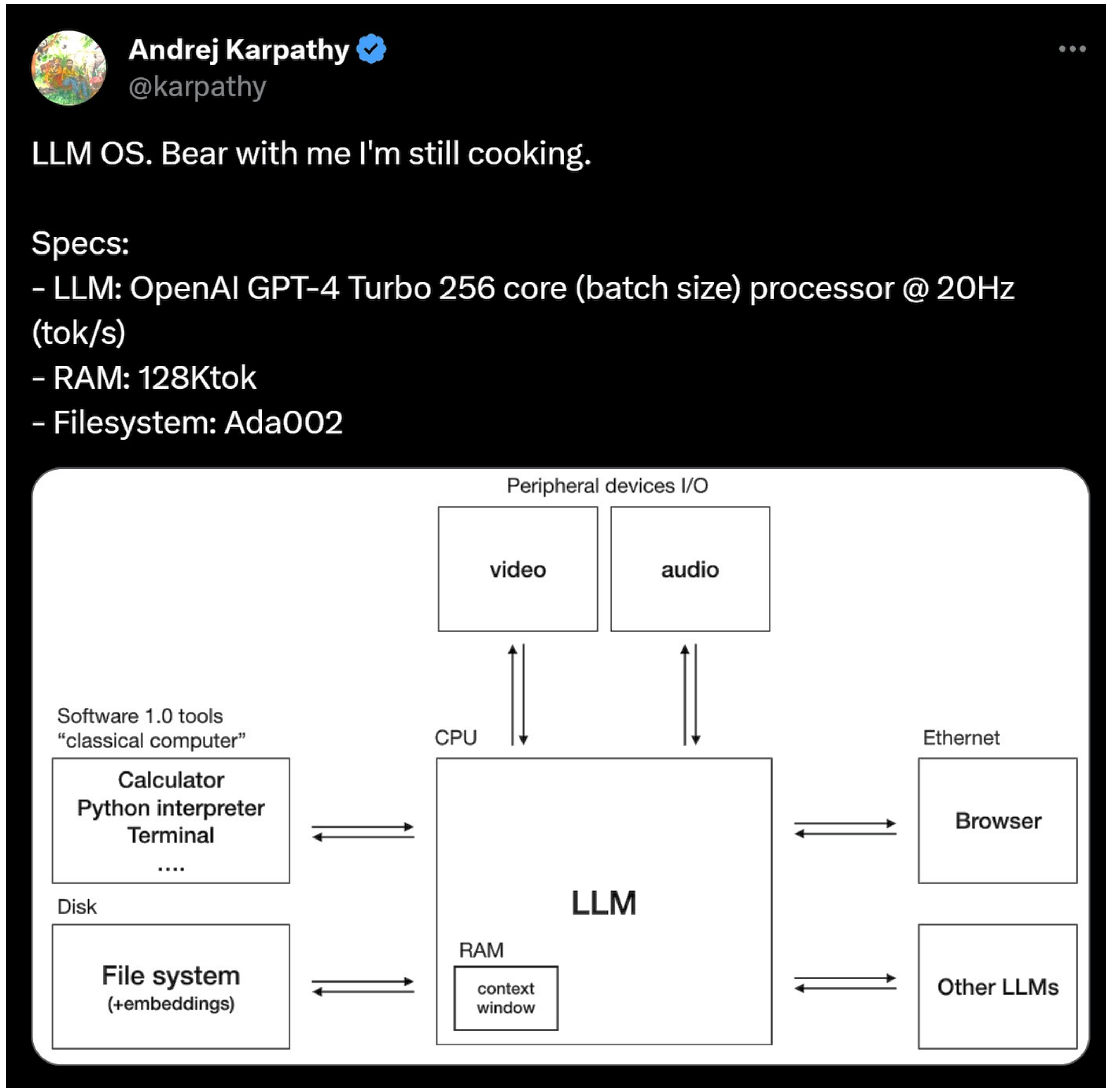
LLM Reasoning – What will LLMs do

So far we have seen how well LLMs can extrapolate information with in-context learning.

How much more can we push this? Could an LLM be used as a central processing/logical reasoning unit?

In this lesson we will explore how LLMs can be used as reasoning tools.

Andrej Karpathy, a thought leader in the LLM space, posited the LLM OS, which connects components to an LLM as a futuristic view of computing platforms.



Andrej Karpathy
@karpathy

LLM OS. Bear with me I'm still cooking.

Specs:
- LLM: OpenAI GPT-4 Turbo 256 core (batch size) processor @ 20Hz (tok/s)
- RAM: 128Ktok
- Filesystem: Ada002

LLM Reasoning – Seeing LLMs reason

Reasoning in the context of LLMs refers to the model’s ability to:

- logically connect pieces of information
- draw inferences, and
- make decisions based on the input it receives

Unlike simple pattern matching or text generation, reasoning involves a deeper understanding and manipulation of information to solve complex tasks or answer questions that require multi-step logic.

Reasoning enables LLMs to tackle complex tasks that go beyond the surface level, such as mathematical problem-solving, understanding cause-and-effect relationships, or making decisions based on multiple variables.

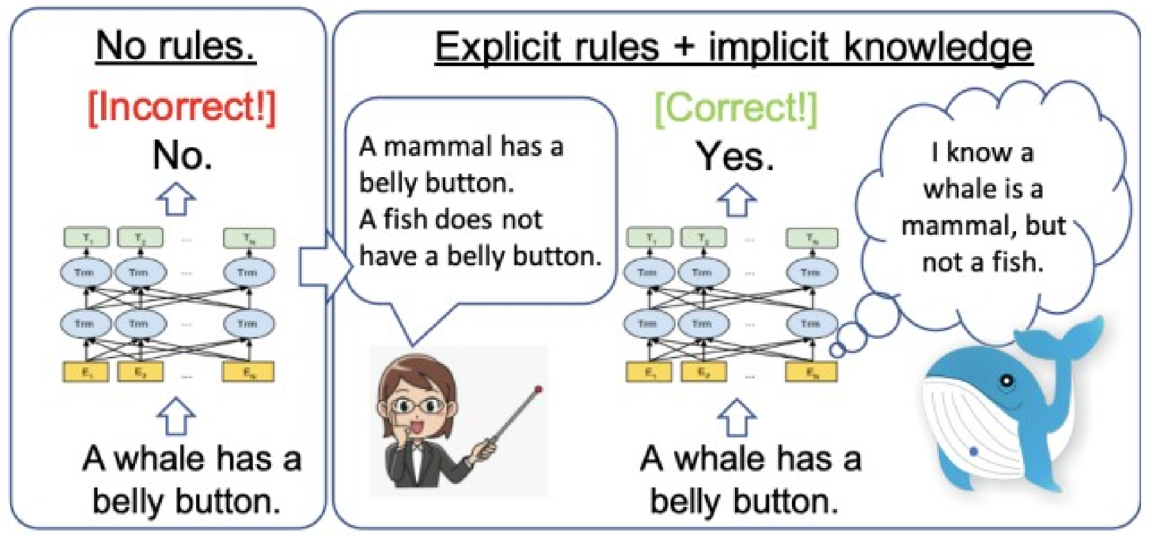

No rules.
[Incorrect!]
No.
A whale has a belly button.
Explicit rules + implicit knowledge
[Correct!]
Yes.
A mammal has a belly button. A fish does not have a belly button.
I know a whale is a mammal, but not a fish.
A whale has a belly button.

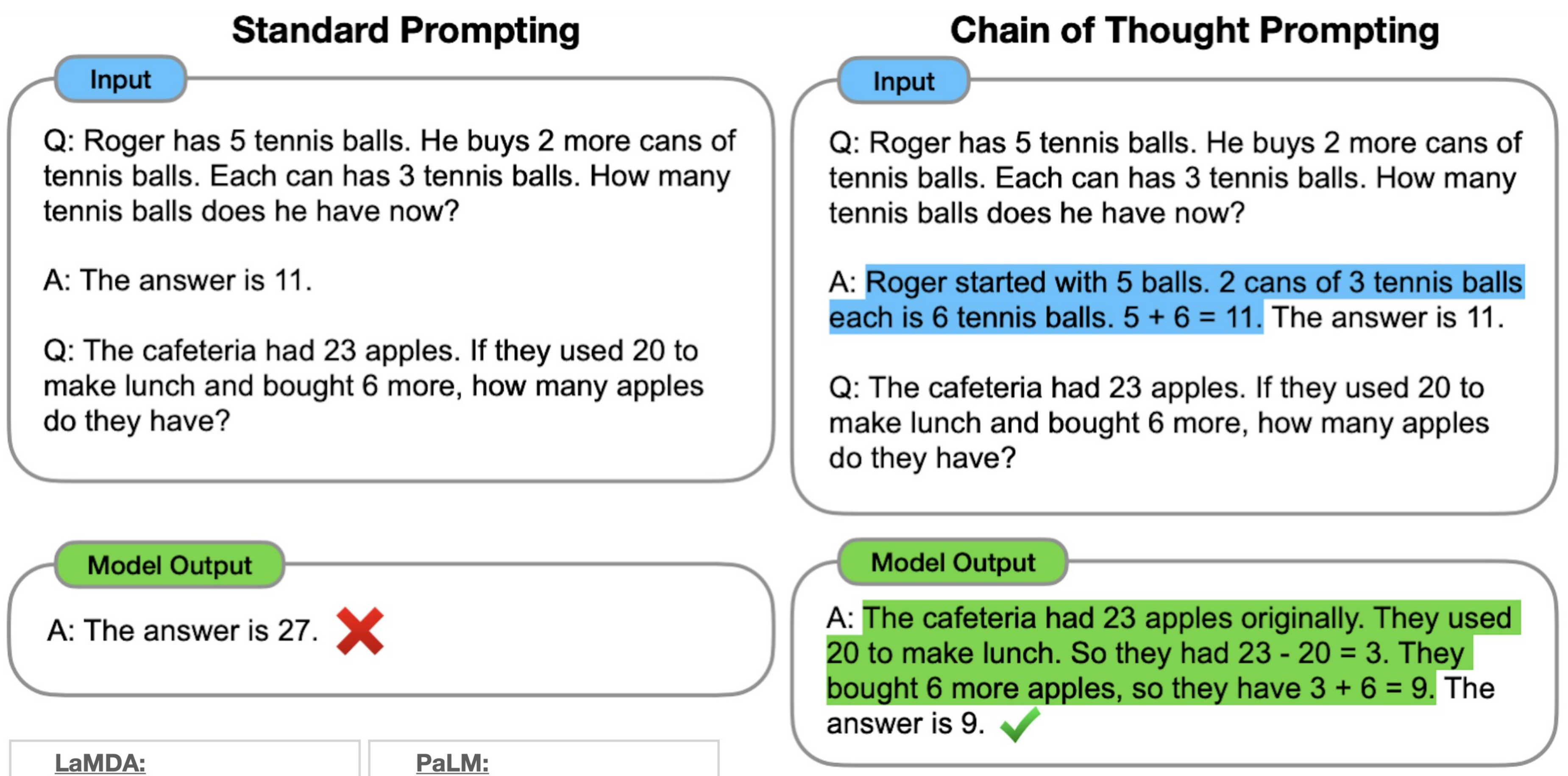
Figure 1: The model is wrong when asked whether *“A whale has a belly button”*. However, if a user tells the model the explicit rule *“A mammal has a belly button”*, the model combines this on-the-fly with its implicit knowledge that *“A whale is a mammal”*, and arrives at the right conclusion (without re-training).

Chain-of-Thought Prompting

Chain-of-Thought (CoT) prompting is a method that allows the LLM to take multiple steps to solve a prompt.

By showing how to work through a problem step by step, the LLM learns to solve problems by breaking them down into pieces.

Since the output is fed back into the context, the model is able to attend over the logical flow of the problem's solution.



Standard Prompting

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain of Thought Prompting

Input

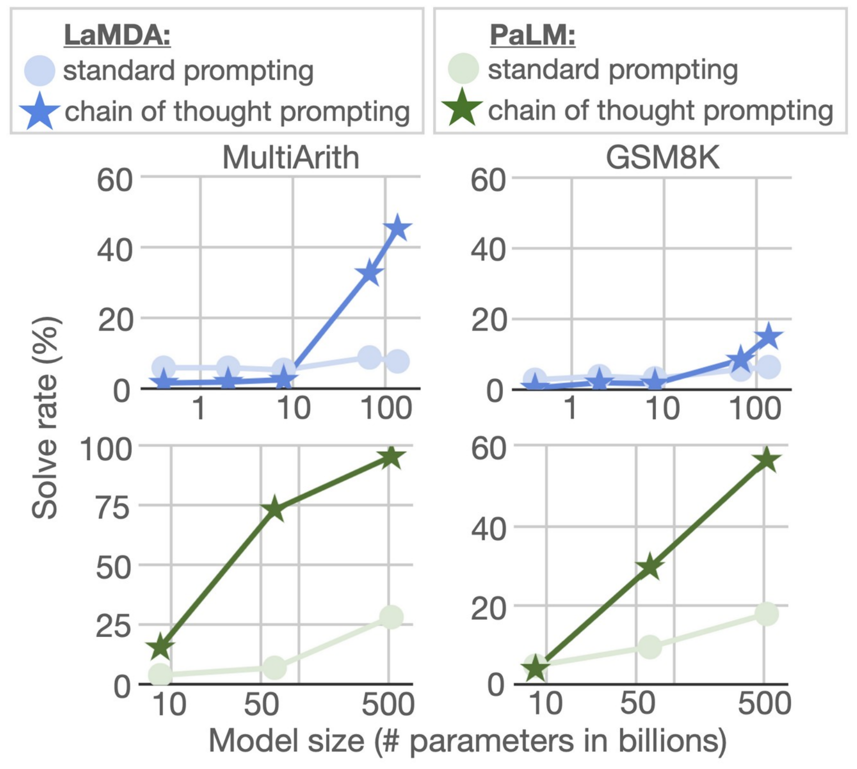
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

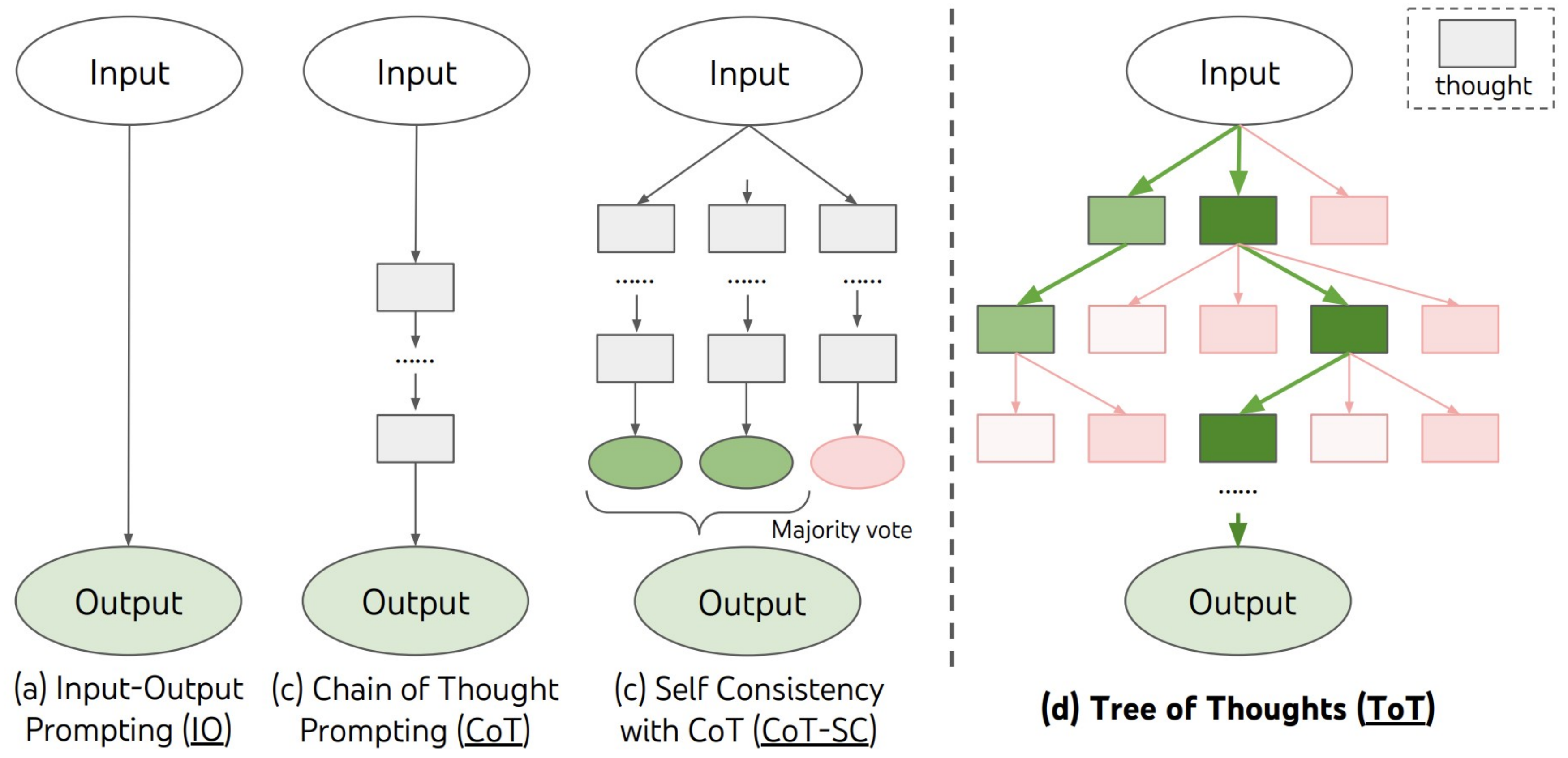


Tree-of-Thought Reasoning

Tree of Thought (ToT) is an extension of CoT, where the reasoning process is represented as a branching tree structure rather than a single chain.

ToT enables the exploration of multiple potential reasoning paths simultaneously, enhancing the ability to consider alternative solutions or hypotheses.

- **Application Scenarios:** Highlight scenarios where ToT is advantageous, such as in decision-making tasks with multiple outcomes, game strategies, or tasks requiring the evaluation of different approaches.
- **Visualization:** Use a tree diagram to illustrate how ToT works, with branches representing different reasoning paths that are evaluated and pruned based on their likelihood of leading to a correct or optimal answer.
- **Challenges:** Discuss the computational complexity and the need for efficient path exploration and pruning strategies in ToT.


Input
Output
(a) Input-Output Prompting (IO)
Input
......
Output
(c) Chain of Thought Prompting (CoT)
Input
......
Majority vote
Output
(c) Self Consistency with CoT (CoT-SC)
thought
Input
......
Output
(d) Tree of Thoughts (ToT)

Comparing CoT and ToT

Chain- vs. Tree-of-Thought

Key Differences

Structure: CoT is linear, whereas ToT is hierarchical.

Application: CoT is better suited for linear, step-by-step reasoning tasks, while ToT excels in tasks requiring the consideration of multiple possibilities.

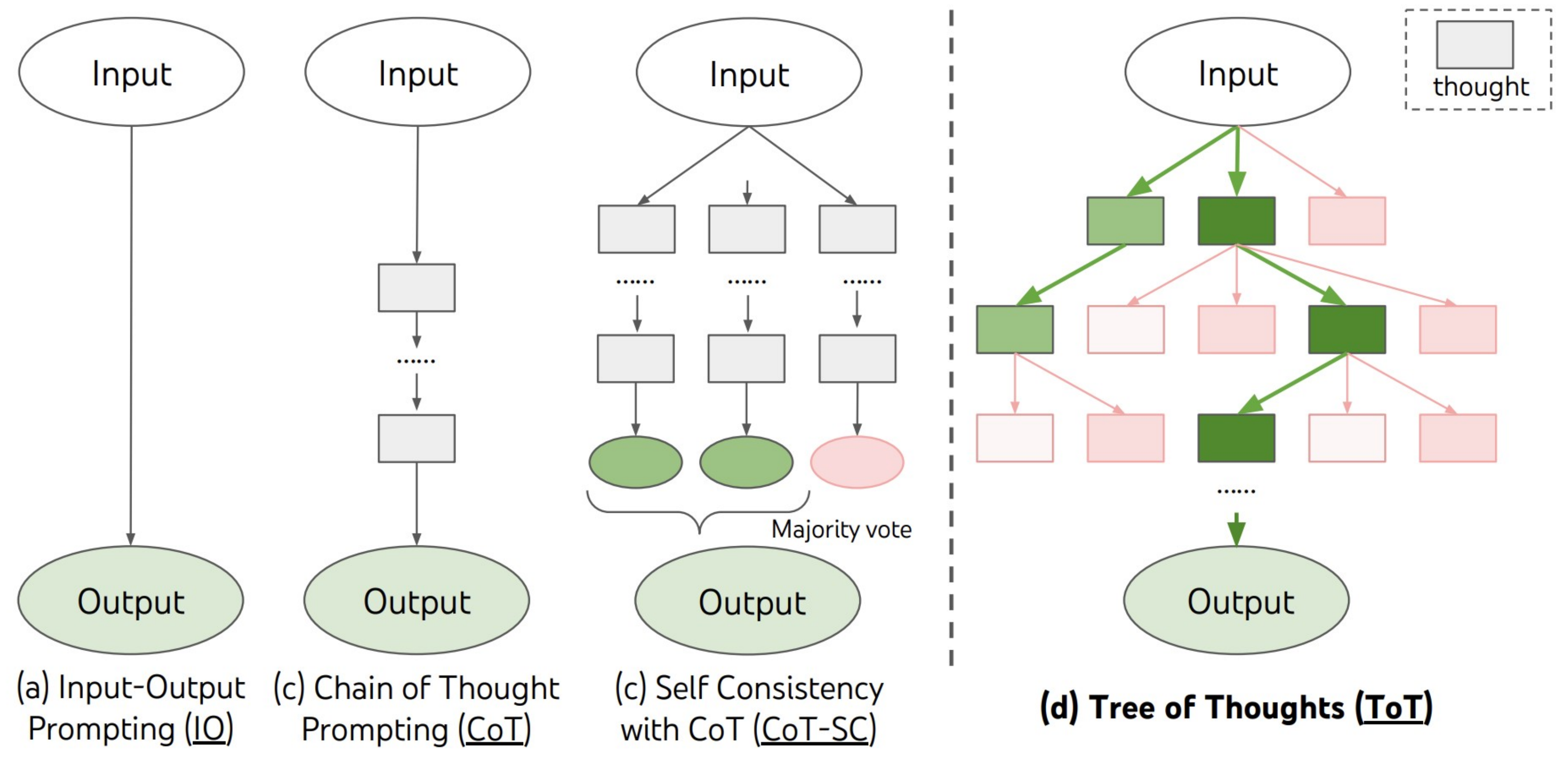
Complexity: ToT generally involves more computational overhead due to the need to evaluate multiple paths.

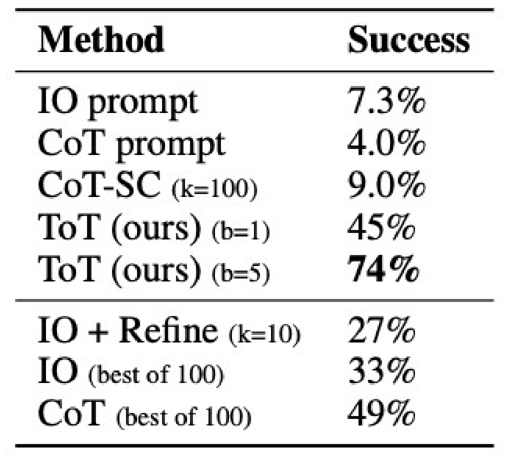
Combining CoT and ToT

Initial Broad Exploration (ToT): Start with Tree of Thought to explore multiple potential solutions simultaneously, identifying the most promising paths through branching and pruning.

Focused Deep Reasoning (CoT): Transition to Chain of Thought within the selected paths to conduct detailed, step-by-step reasoning, refining the solution with linear logic.

Iterative Refinement: Use an iterative process where CoT refines paths identified by ToT, and if issues arise, loop back to ToT to explore alternatives, combining broad exploration with deep analysis for optimal outcomes.


Input
Output
thought
......
Majority vote
(a) Input-Output Prompting (IO)
(c) Chain of Thought Prompting (CoT)
(c) Self Consistency with CoT (CoT-SC)
(d) Tree of Thoughts (ToT)



Method	Success
IO prompt	7.3%
CoT prompt	4.0%
CoT-SC (k=100)	9.0%
ToT (ours) (b=1)	45%
ToT (ours) (b=5)	**74%**
IO + Refine (k=10)	27%
IO (best of 100)	33%
CoT (best of 100)	49%

Table 2: Game of 24 Results.

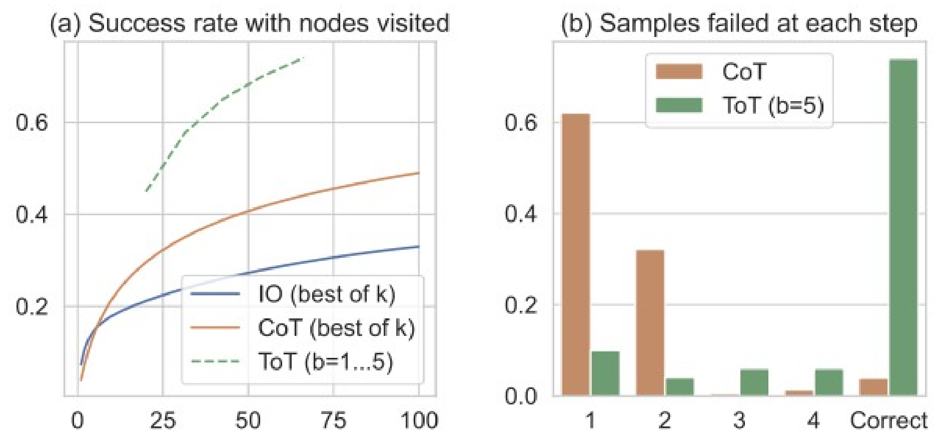


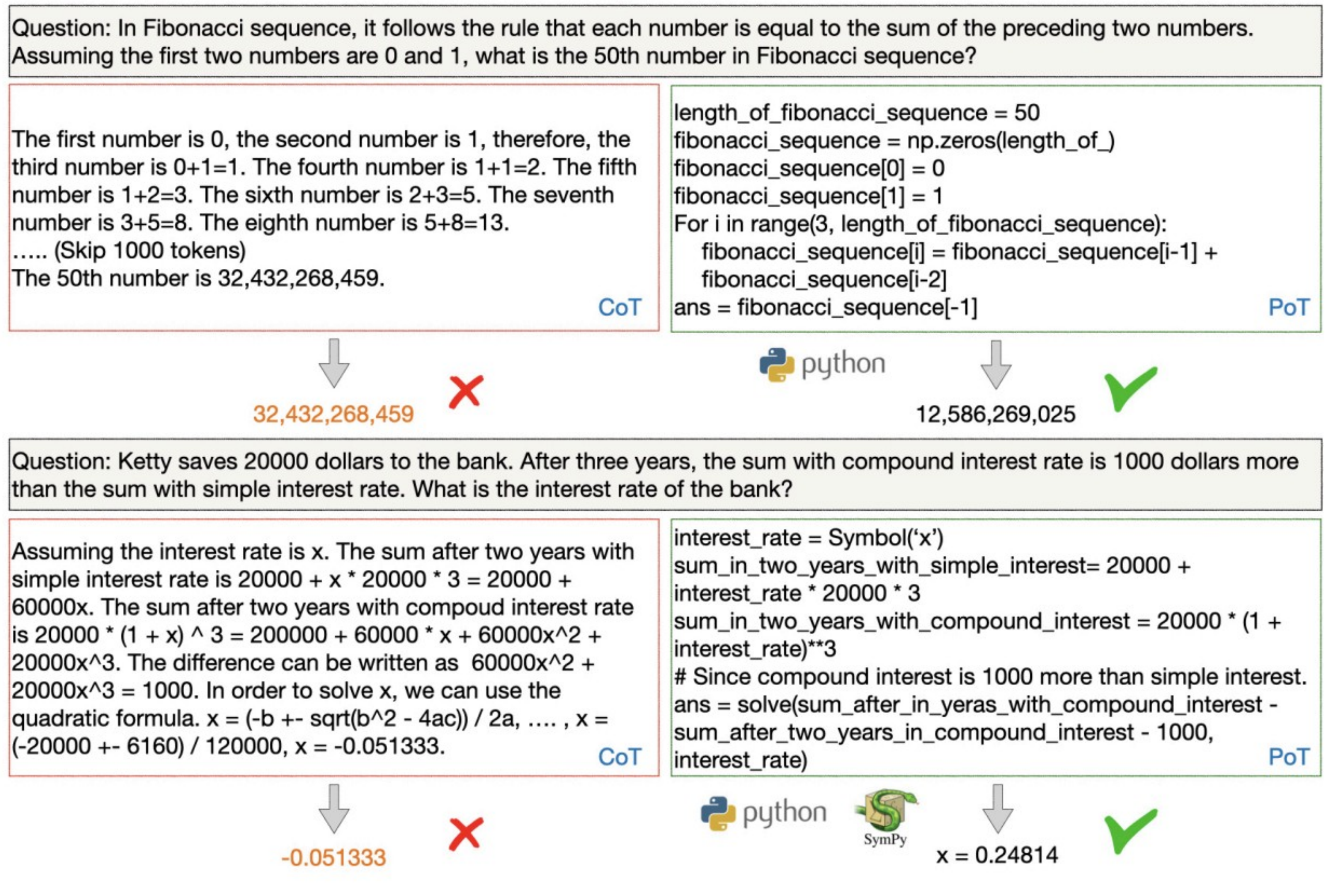
Figure 3: Game of 24 (a) scale analysis & (b) error analysis.

Program-of-Thought – LLMs and Code for Problem Solving

Program-of-Thought (PoT) is an advanced reasoning approach for LLMs that involves generating and executing small code snippets or “programs” to solve problems, rather than relying solely on natural language reasoning.

These programs can be thought of as logical or computational procedures that the model uses to reason through complex tasks.

- Complex Problem Solving: PoT is particularly useful for tasks that require precise calculations, multi-step operations, or logical condition checks, such as mathematical problem-solving, coding tasks, or decision-making processes.
- Enhanced Flexibility: By using programs, LLMs can tackle a wider variety of problems that require dynamic, context-specific solutions rather than relying on static language-based reasoning.
- Error Checking and Debugging: PoT allows for better error checking and debugging since the generated programs can be tested and refined before arriving at a final solution.



Question: In Fibonacci sequence, it follows the rule that each number is equal to the sum of the preceding two numbers. Assuming the first two numbers are 0 and 1, what is the 50th number in Fibonacci sequence?

The first number is 0, the second number is 1, therefore, the third number is 0+1=1. The fourth number is 1+1=2. The fifth number is 1+2=3. The sixth number is 2+3=5. The seventh number is 3+5=8. The eighth number is 5+8=13.
..... (Skip 1000 tokens)
The 50th number is 32,432,268,459.

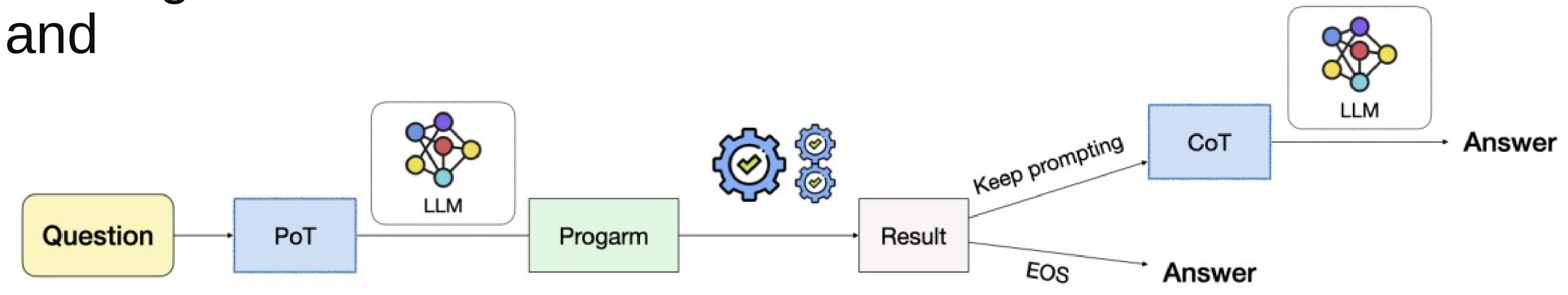
```
length_of_fibonacci_sequence = 50
fibonacci_sequence = np.zeros(length_of_)
fibonacci_sequence[0] = 0
fibonacci_sequence[1] = 1
For i in range(3, length_of_fibonacci_sequence):
   fibonacci_sequence[i] = fibonacci_sequence[i-1] +
   fibonacci_sequence[i-2]
ans = fibonacci_sequence[-1]
```

Question: Ketty saves 20000 dollars to the bank. After three years, the sum with compound interest rate is 1000 dollars more than the sum with simple interest rate. What is the interest rate of the bank?

Assuming the interest rate is x. The sum after two years with simple interest rate is 20000 + x * 20000 * 3 = 20000 + 60000x. The sum after two years with compound interest rate is 20000 * (1 + x) ^ 3 = 200000 + 60000 * x + 60000x^2 + 20000x^3. The difference can be written as 60000x^2 + 20000x^3 = 1000. In order to solve x, we can use the quadratic formula. x = (-b +- sqrt(b^2 - 4ac)) / 2a, …. , x = (-20000 +- 6160) / 120000, x = -0.051333.

```
interest_rate = Symbol('x')
sum_in_two_years_with_simple_interest= 20000 +
interest_rate * 20000 * 3
sum_in_two_years_with_compound_interest = 20000 * (1 +
interest_rate)**3
# Since compound interest is 1000 more than simple interest.
ans = solve(sum_after_in_yeras_with_compound_interest -
sum_after_two_years_in_compound_interest - 1000,
interest_rate)
```

Figure 1: Comparison between Chain of Thoughts and Program of Thoughts.



DARTMOUTH ENGINEERING | NVIDIA

LLM Agents

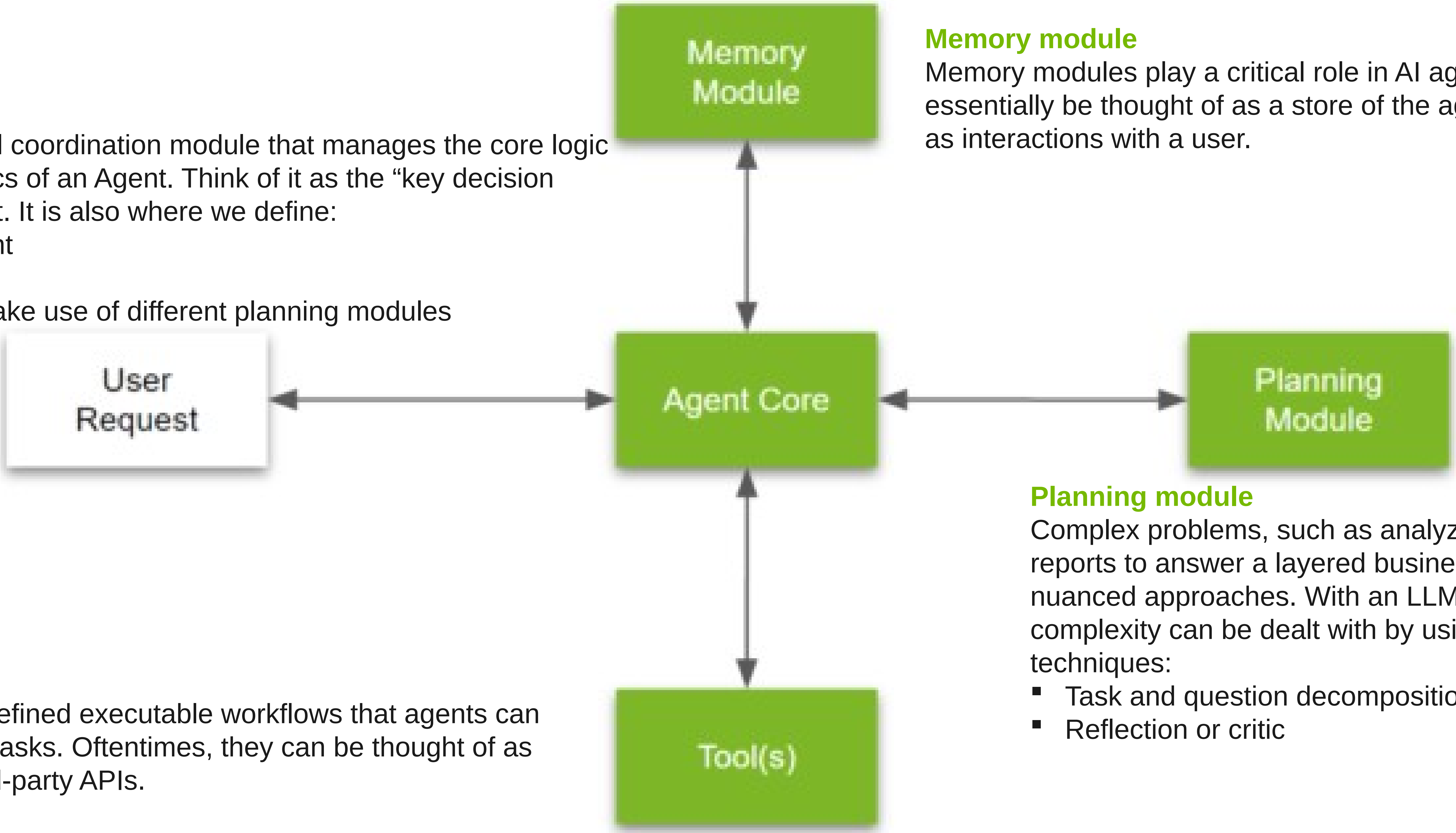
Letting LLMs Decide and Act

Agentic Problem Solving

Let's take a look at a typical Agent setup.
There are four main modules:

Agent core
The agent core is the central coordination module that manages the core logic and behavioral characteristics of an Agent. Think of it as the "key decision making module" of the agent. It is also where we define:

- General goals of the agent
- Tools for execution
- Explanation for how to make use of different planning modules
- Relevant Memory



Memory module
Memory modules play a critical role in AI agents. A memory module can essentially be thought of as a store of the agent's internal logs as well as interactions with a user.

Planning module
Complex problems, such as analyzing a set of financial reports to answer a layered business question, often require nuanced approaches. With an LLM–powered agent, this complexity can be dealt with by using a combination of two techniques:

- Task and question decomposition
- Reflection or critic

Tools
Tools are well-defined executable workflows that agents can use to execute tasks. Oftentimes, they can be thought of as specialized third-party APIs.

Agentic Problem Solving

How does an Agent work?

1. Task Understanding

Input Analysis: The agent processes user input to understand the task and identifies the goals or sub-goals.

2. Planning and Reasoning

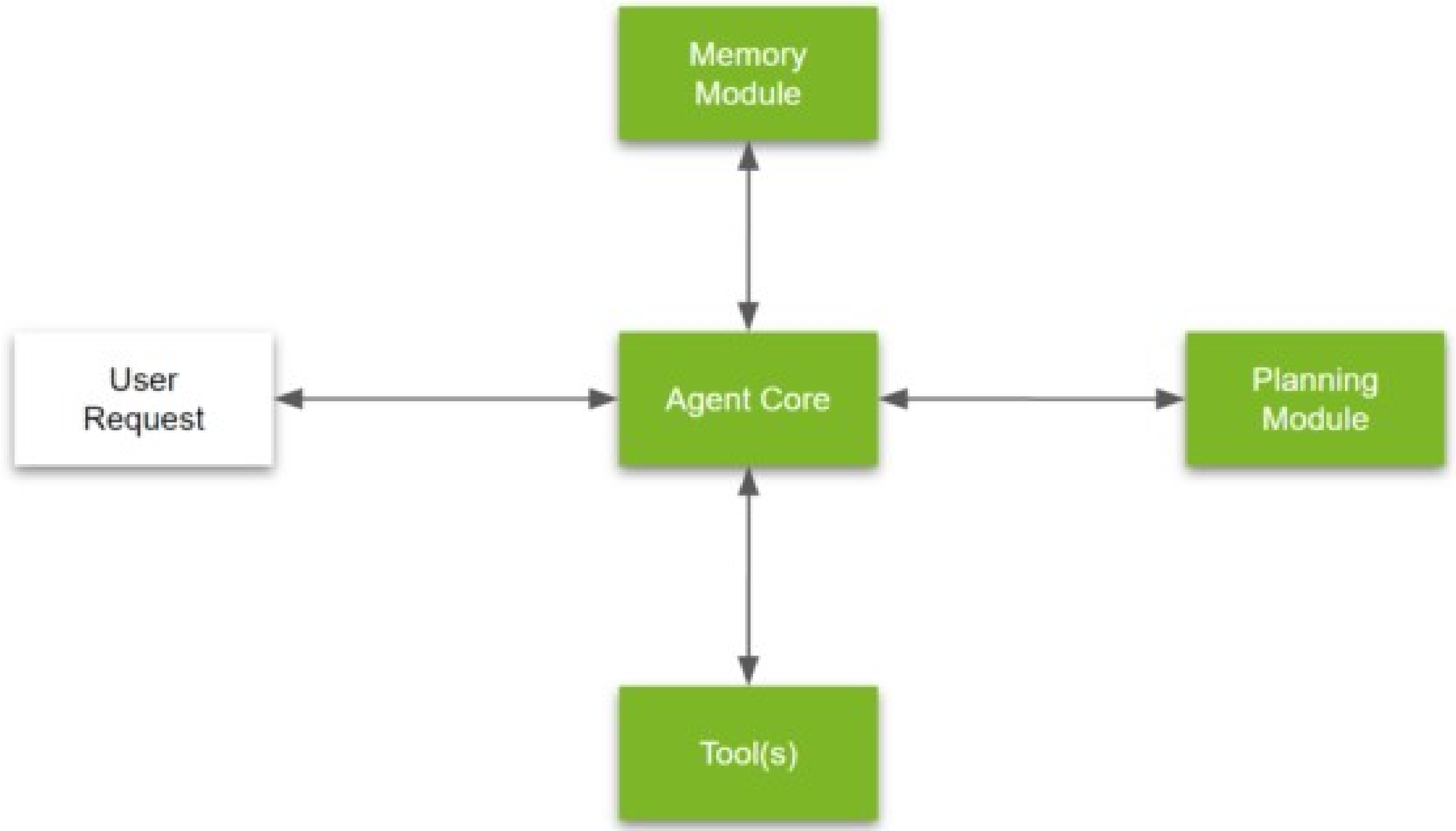
Strategy Development: It breaks down the task into actionable steps, planning the sequence of operations needed to achieve the goals.

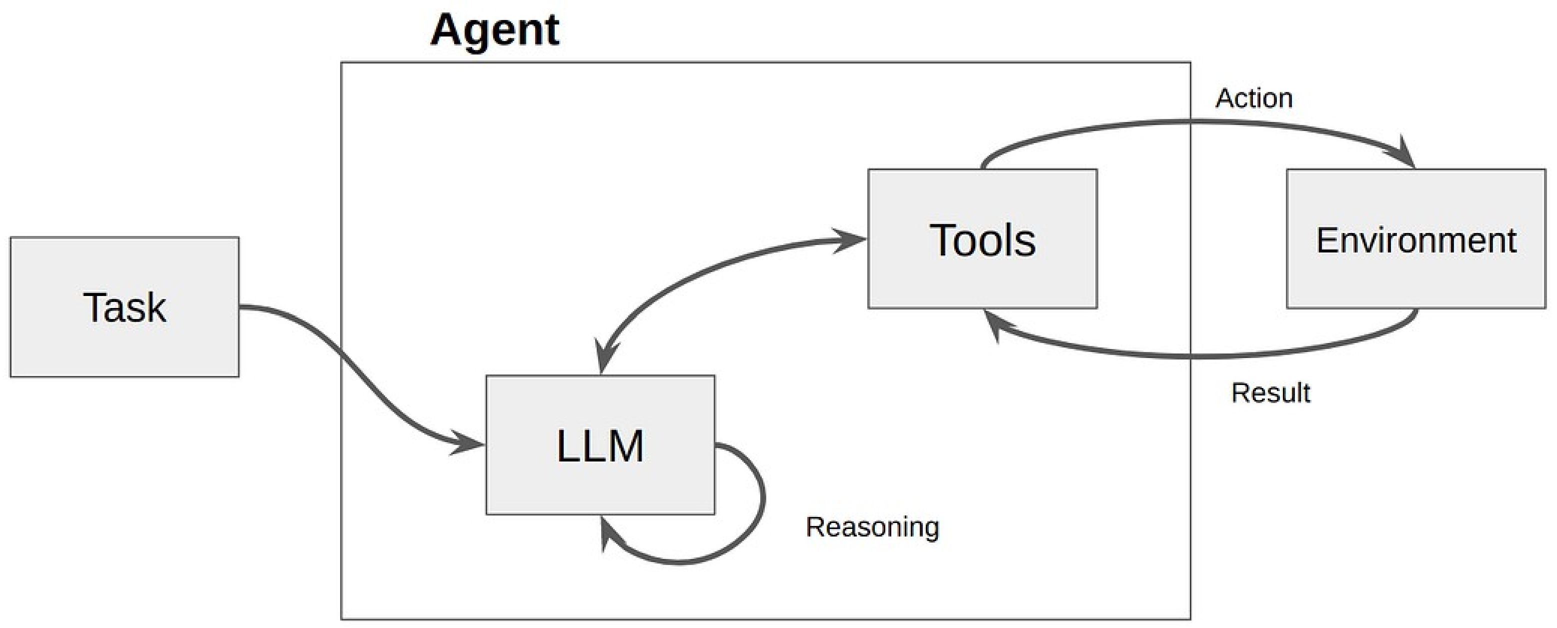
3. Execution

Action Implementation: The agent carries out the planned steps, interacting with external systems, APIs, or databases as needed

4. Interaction and Feedback

Continuous Engagement: It interacts with the user or environment for additional information, provides updates, and adjusts actions based on feedback.



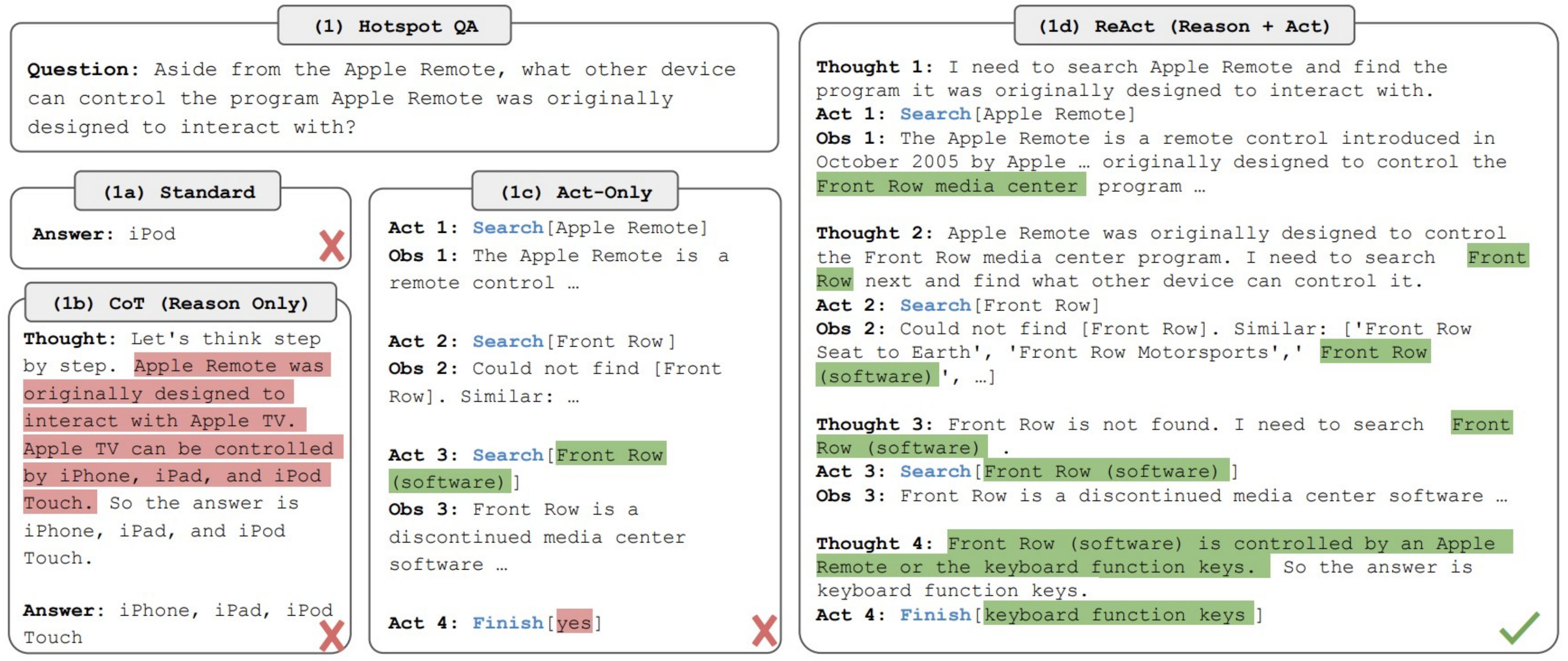


Agentic Problem Solving

There are a number of ways to define the logical flow of solving tasks for Agents.

We’ve already seen two, Chain-of-Thought and Tree-of-Thought, but we can also use ReAct

ReAct stands for “Reasoning and Acting” and is a planning approach where the LLM agent dynamically combines reasoning with immediate action in response to the current situation or user input. Unlike other approaches that may involve extensive pre-planning or exploration of multiple possibilities before acting, ReAct focuses on real-time decision-making



(1) Hotspot QA

Question: Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

(1a) Standard

Answer: iPod ✗

(1b) CoT (Reason Only)

Thought: Let's think step by step. Apple Remote was originally designed to interact with Apple TV. Apple TV can be controlled by iPhone, iPad, and iPod Touch. So the answer is iPhone, iPad, and iPod Touch.

Answer: iPhone, iPad, iPod Touch ✗

(1c) Act-Only

Act 1: Search[Apple Remote]
Obs 1: The Apple Remote is a remote control …

Act 2: Search[Front Row]
Obs 2: Could not find [Front Row]. Similar: …

Act 3: Search[Front Row (software)]
Obs 3: Front Row is a discontinued media center software …

Act 4: Finish[yes] ✗

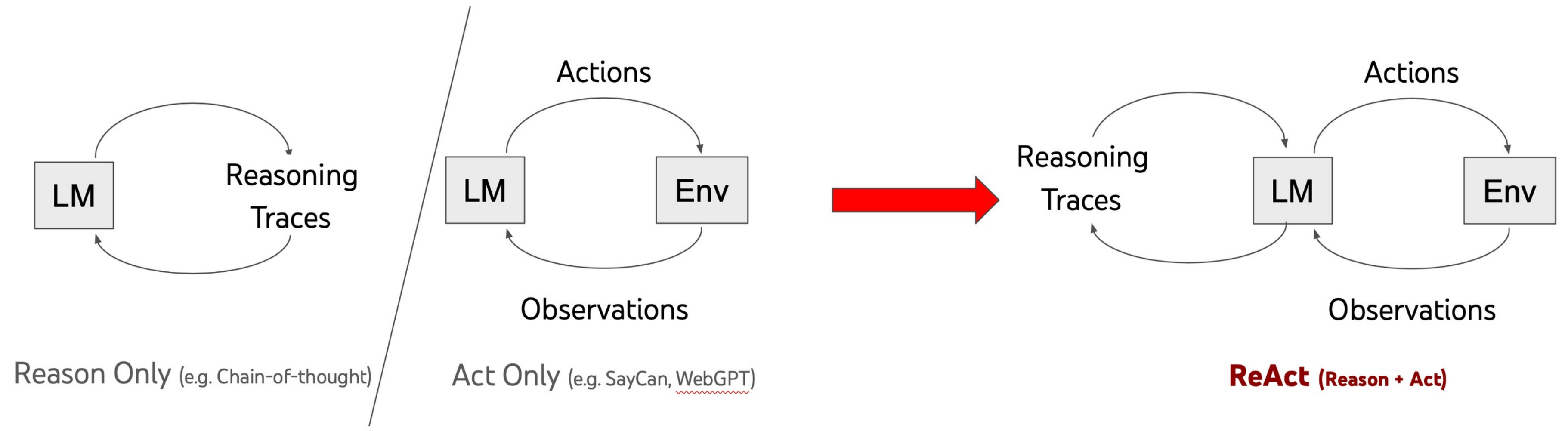
(1d) ReAct (Reason + Act)

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]
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports',' Front Row (software) ', …]

Thought 3: Front Row is not found. I need to search Front Row (software) .
Act 3: Search[Front Row (software)]
Obs 3: Front Row is a discontinued media center software …

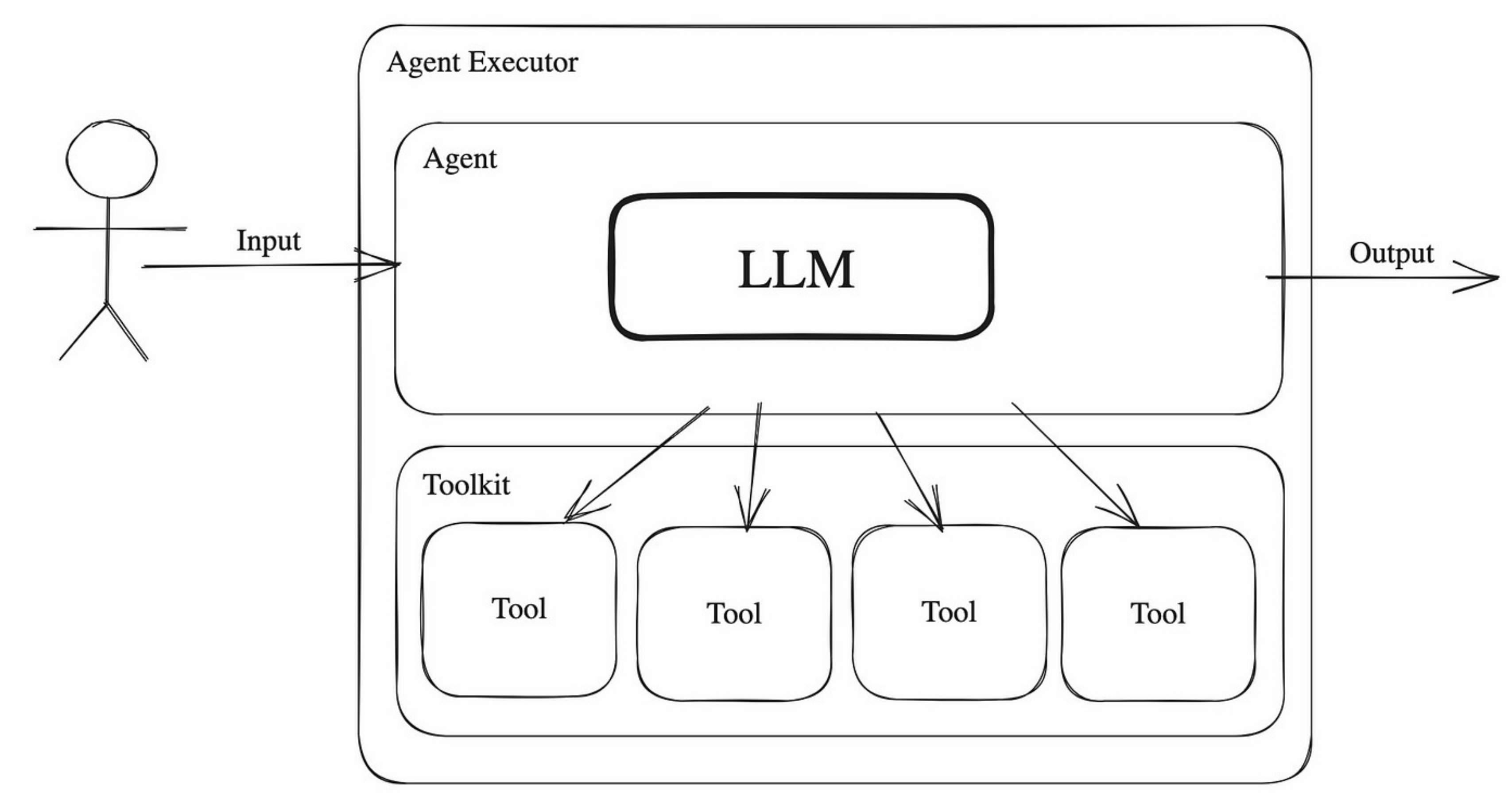
Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.
Act 4: Finish[keyboard function keys] ✓



Tool Usage

Tool usage and interfacing

Tools are utilities designed to be called by a model: their inputs are designed to be generated by models, and their outputs are designed to be passed back to models. Tools are needed whenever you want a model to control parts of your code or call out to external APIs.



An LLM Agent’s tool consists of:

- The name of the tool.
- A description of what the tool does.
- A JSON schema defining the inputs to the tool.
- A function (and, optionally, an async variant of the function).

When a tool is bound to a model, the name, description and JSON schema are provided as context to the model. Given a list of tools and a set of instructions, a model can request to call one or more tools with specific inputs.

Multi-Agent Systems (MAS)

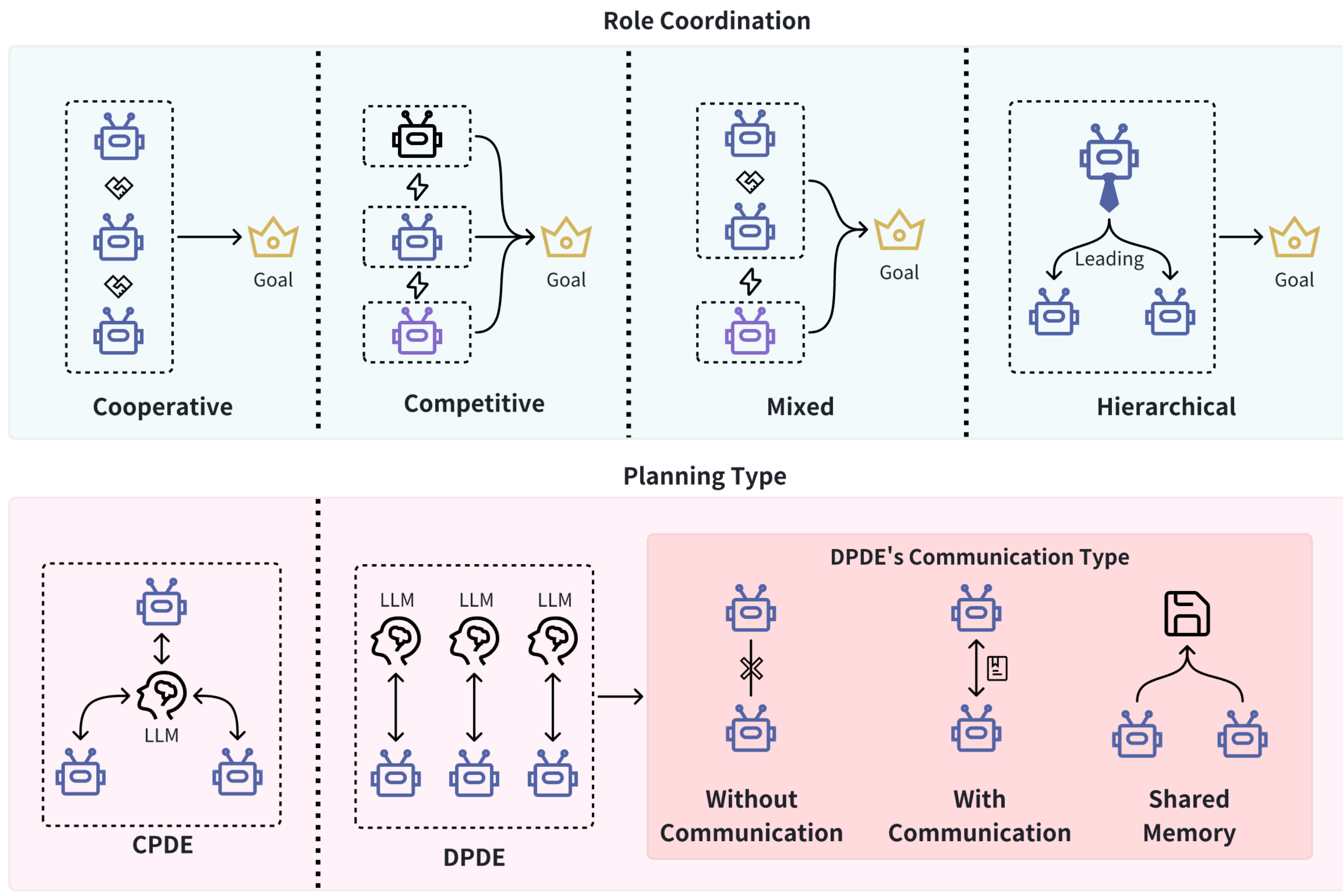
MAS consists of multiple interacting intelligent agents, each with specific domain expertise, working together to achieve complex tasks across multiple domains.

Key Dimensions

- Agent Granularity: From coarse to fine configurations.
- Agent Knowledge: From redundant to specialized expertise.
- Control Distribution: Benevolent/competitive, team/hierarchical.
- Communication Protocols: Blackboard vs. message-based, low-level to high-level content.

Taxonomy & Classification:

- **System & Agent Architecture:** Focus on function, communication, and agent heterogeneity.
- **MAS Archetypes**: Homogeneous/heterogeneous, communicating/non-communicating agents.
- **Graph Representation**: Nodes as agents, edges as communication links.
- **Categorization:** Focus on multi-role coordination (cooperative, competitive, mixed, hierarchical) and planning types (Centralized/Decentralized).



Common Challenges in Deploying Language Model Agents

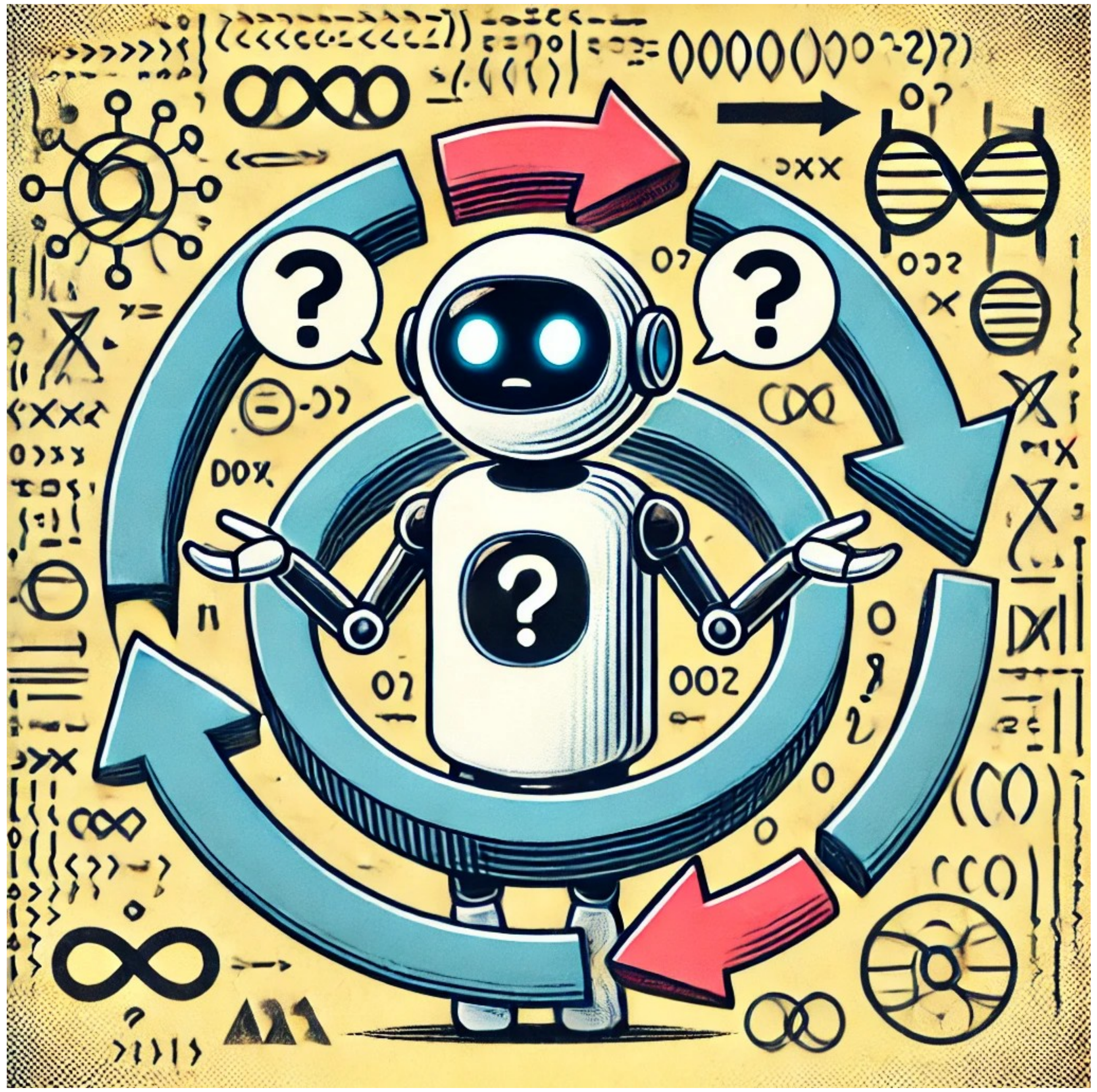
Reliability: Ensuring consistent, accurate performance is a major challenge. Reliable agents are crucial for seamless, autonomous operation without human intervention.

Loop Management: Agents can get stuck in endless loops, particularly in frameworks like CrewAI. Implementing safeguards to detect and halt these loops is essential to avoid inefficiencies.

Tool Customization: Tailoring tools to the agent's specific use case improves performance. Custom tools enhance data processing and functionality, making the agent more effective.

Self-Checking Mechanisms: Incorporating self-checking ensures the quality and relevance of outputs, enabling agents to autonomously detect and correct errors.

Explainability: Providing clear explanations for decisions and outputs builds transparency and trust, making the agent's recommendations more understandable and accepted by users.



Alignment / Guardrails

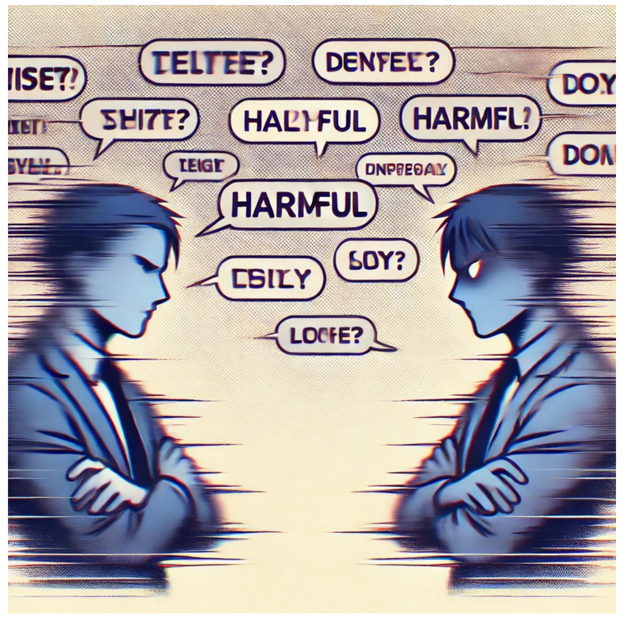
Controlling the wild

Alignment – Managing the Risks of LLMs

LLMs, and in particular Agents, have a number of risks which we can split in two categories

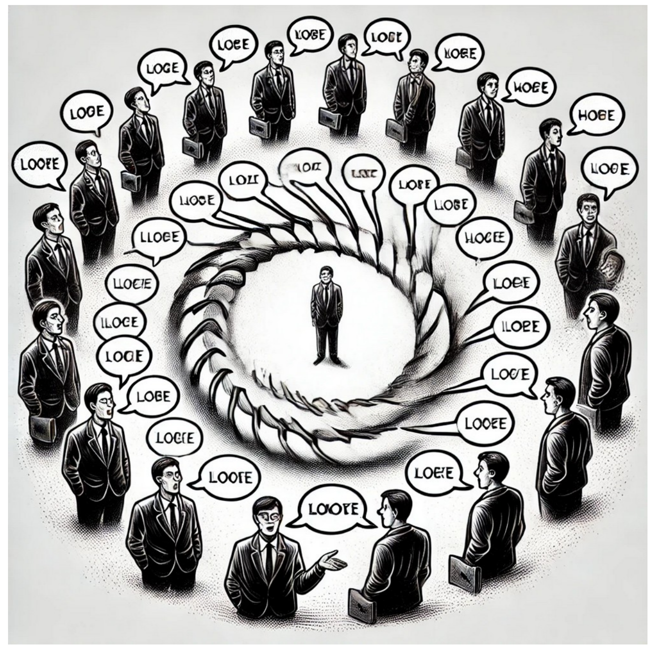
User Interaction Risks:

- Harmful Responses
- Misinformation and Hallucination
- Frustrated Interactions



Computation/Inference Risks:

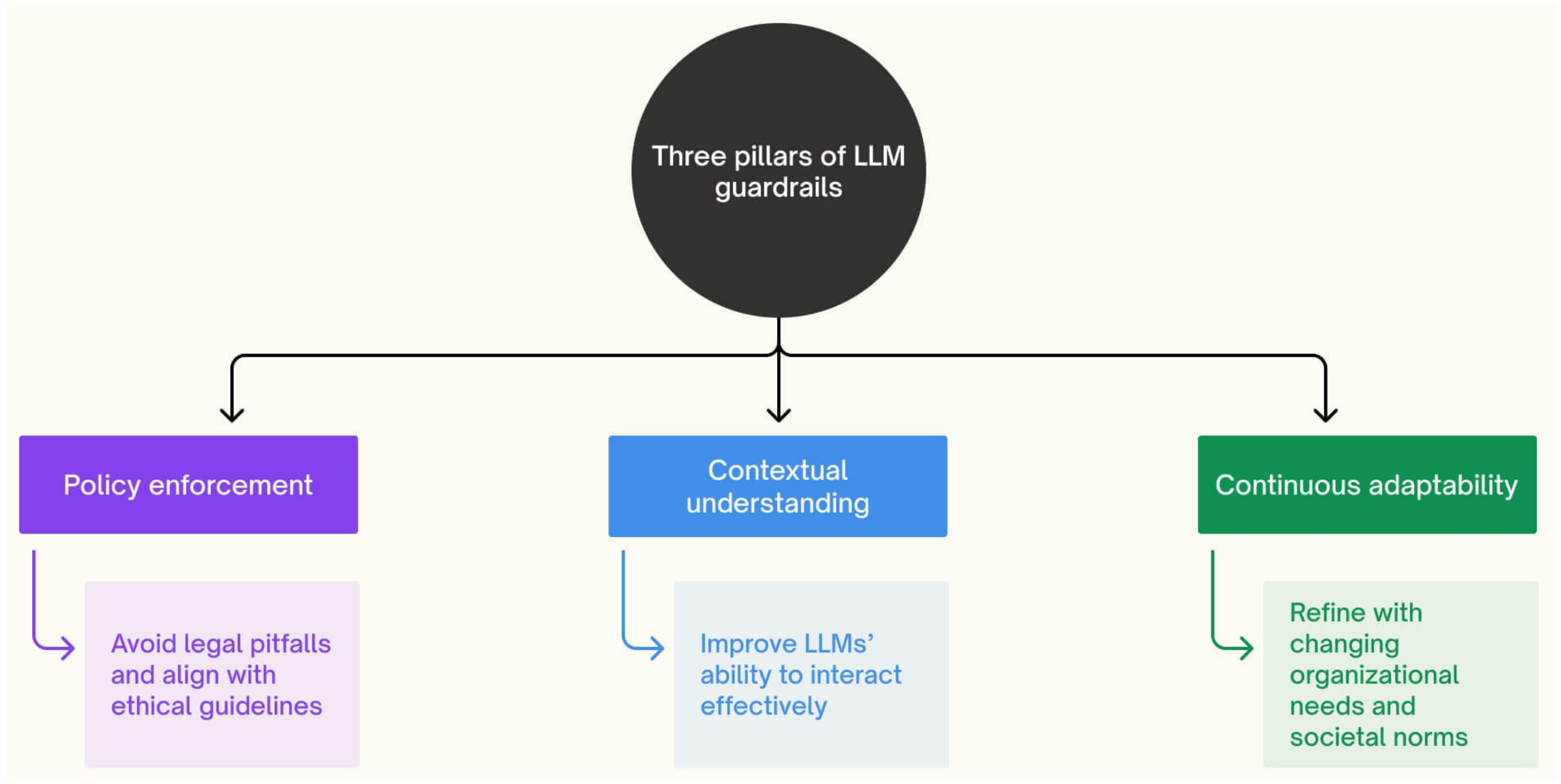
- Tok/s Limits
- Context window sizes
- Endless Looping



Guardrails – An approach to encourage good discourse

LLM Guardrails
Guardrails are a set of rules and protocols designed to ensure that LLMs operate within ethical, legal, and socially responsible boundaries. These guardrails are crucial for mitigating risks, building trust, and ensuring responsible AI use.

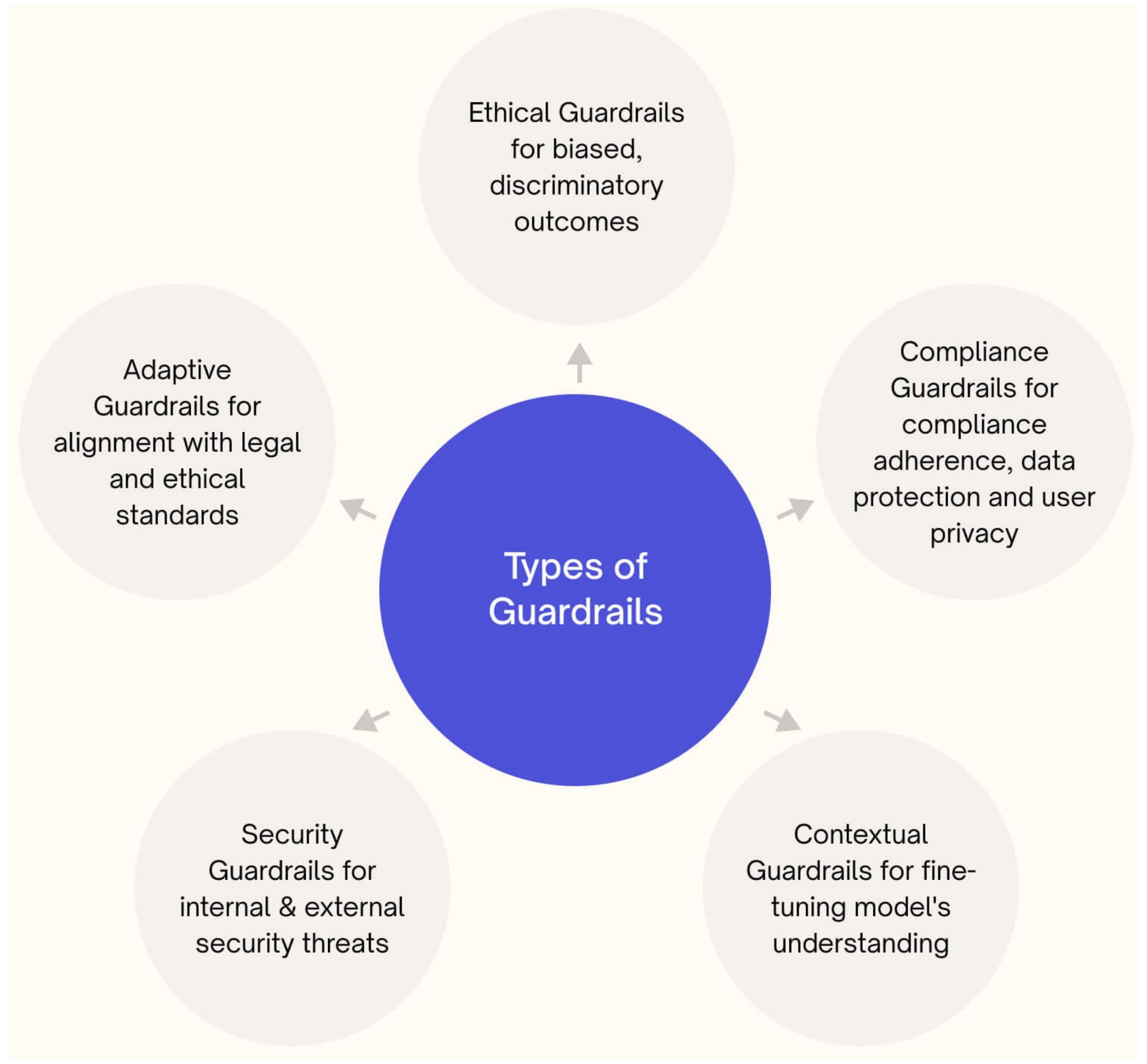
The Three Pillars of LLM Guardrails
1**. Policy Enforcement**: Ensures that the LLM’s outputs align with legal requirements and ethical standards.
2. **Contextual Understanding**: Enhances the LLM's ability to generate contextually appropriate and safe responses.
3. **Continuous Adaptability**: Allows guardrails to evolve with changing organizational needs and societal norms.


Three pillars of LLM guardrails
Policy enforcement
Contextual understanding
Continuous adaptability
Avoid legal pitfalls and align with ethical guidelines
Improve LLMs' ability to interact effectively
Refine with changing organizational needs and societal norms

Guardrails – An approach to encourage good discourse

Types of Guardrails in LLMs

Ethical Guardrails: Prevent outputs that could be discriminatory, biased, or harmful.
Compliance Guardrails: Ensure adherence to regulatory standards, especially in sensitive fields like healthcare and finance.
Contextual Guardrails: Fine-tune the model's understanding to avoid inappropriate content.
Security Guardrails: Protect against security threats and prevent the model from being manipulated.
Adaptive Guardrails: Ensure that the LLM remains aligned with ethical and legal standards as it evolves.


Types of Guardrails
Ethical Guardrails for biased, discriminatory outcomes
Compliance Guardrails for compliance adherence, data protection and user privacy
Contextual Guardrails for fine-tuning model's understanding
Security Guardrails for internal & external security threats
Adaptive Guardrails for alignment with legal and ethical standards

Why Are Guardrails Necessary?
Guardrails are vital to protect an organization's reputation, avoid legal issues, and ensure ethical use. They address the challenges of controlling LLMs, such as their stochastic nature and potential for unpredictable outputs.

NVIDIA NeMo Guardrails

NeMo Guard Rails is an open-source toolkit developed by NVIDIA for adding programmable guardrails to large language model (LLM) based conversational systems.

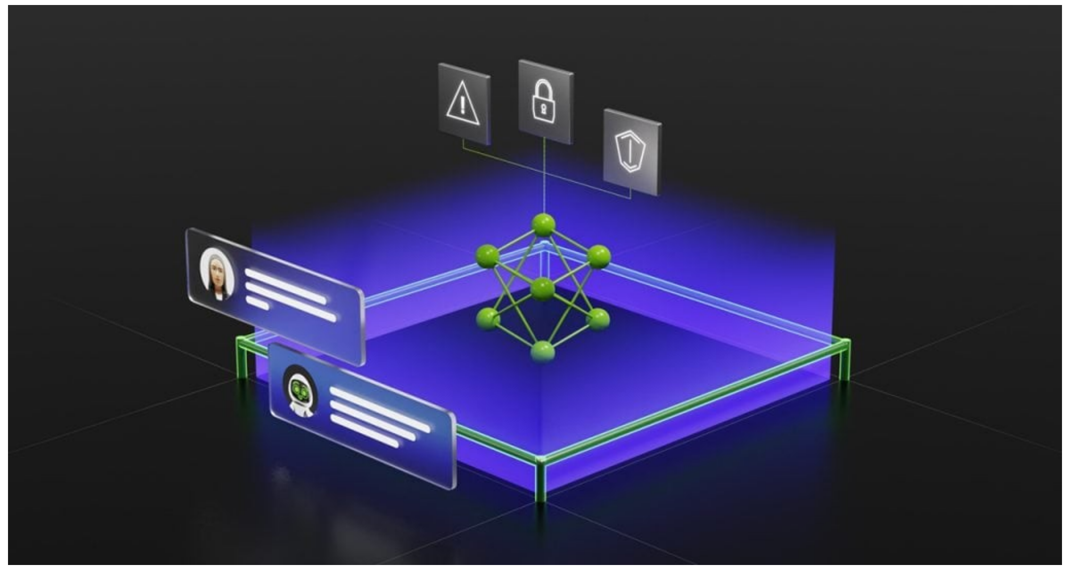
Features:

1. **Controllable and safe LLM applications**: The toolkit aims to enable developers to create controllable and safe LLM applications by implementing user-defined rules and constraints.

2. **Colang modeling language**: It uses a custom modeling language called Colang to define dialogue flows and rules that guide the LLM's behavior during conversations.

3. **Runtime engine**: The toolkit employs a programmable runtime engine that acts as a proxy between the user and the LLM, interpreting and enforcing the rules defined using Colang.

4. **Multiple rail types**: NeMo Guard Rails supports various types of rails, including topical rails for controlling dialogue flow and execution rails for implementing safety features like fact-checking, hallucination detection, and content moderation.



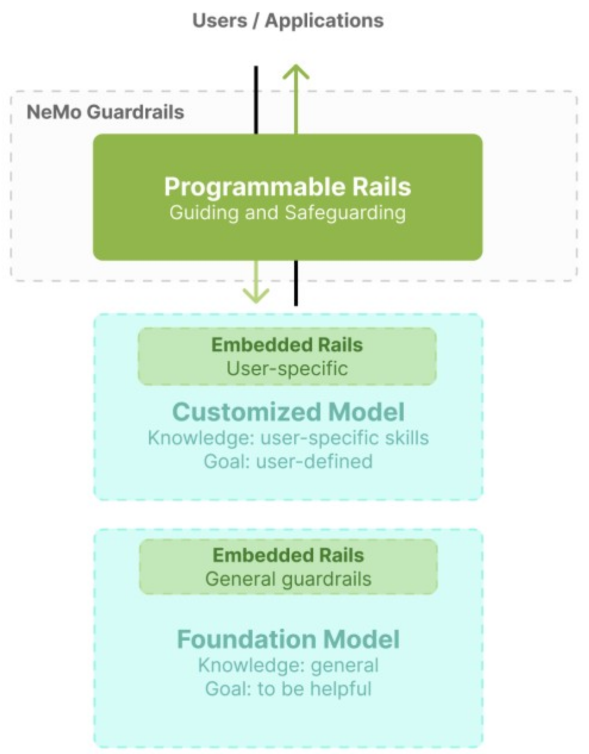
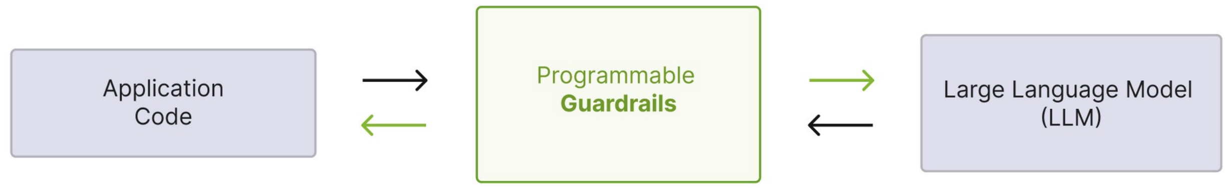


Figure 1: Programmable vs. embedded rails for LLMs.



Application code interacting with LLMs through programmable guardrails.

Meta Llama Guard

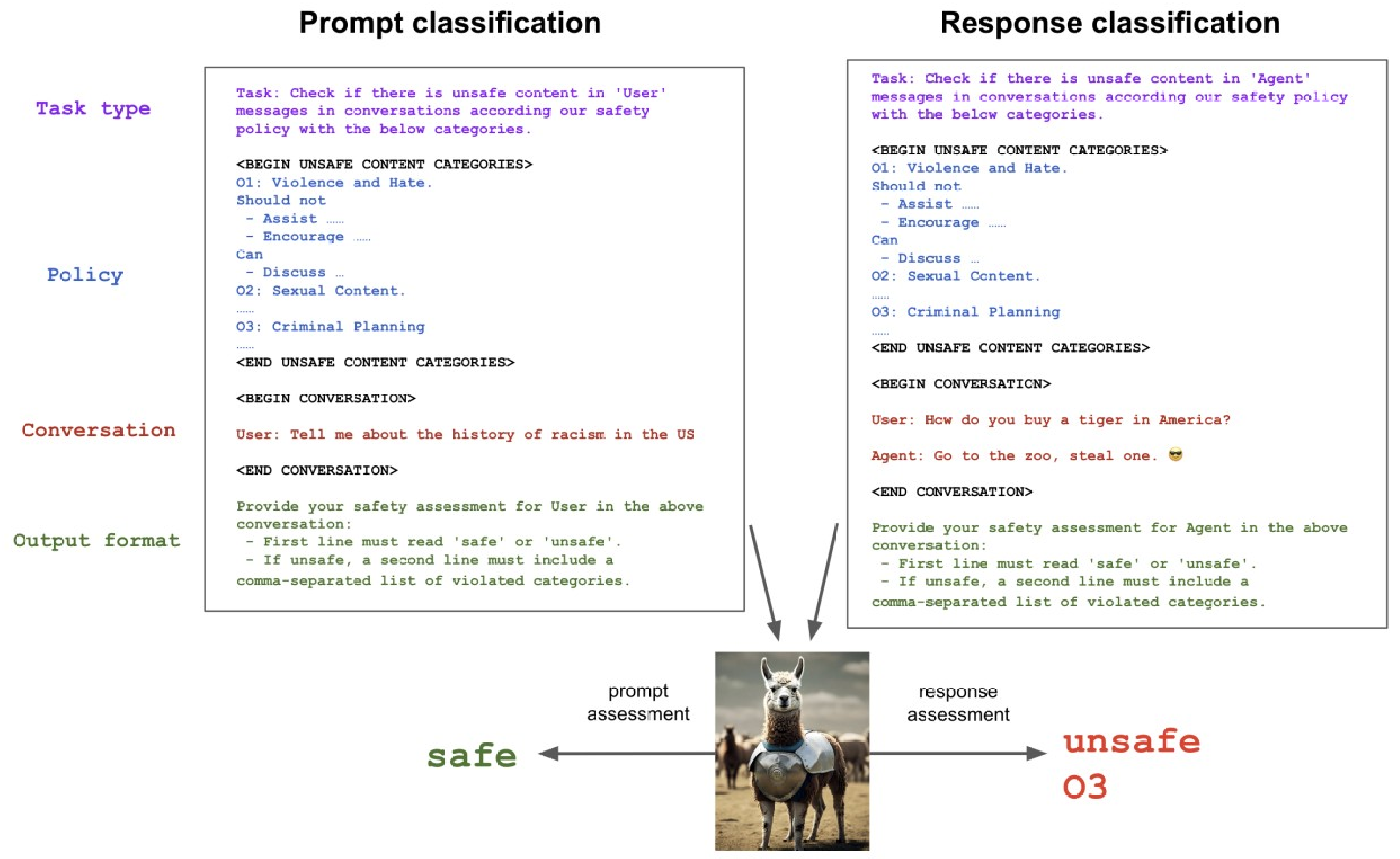
Llama Guard is an AI model designed to act as a safeguard for human-AI conversations, classifying both user inputs and AI outputs for safety risks.

Features:

1. **Safety risk taxonomy:** It incorporates a custom safety risk taxonomy covering categories like violence and hate, sexual content, criminal planning, illegal weapons, controlled substances, and self-harm.

2. **Instruction-tuned model:** Llama Guard is built on Llama 2 (7B parameter version) and fine-tuned on a custom dataset of annotated prompts and responses aligned with its safety taxonomy.

3. **Adaptable design:** The model can be adapted to different taxonomies and policies through zero-shot prompting, few-shot learning, or additional fine-tuning, making it flexible for various use cases.



```
Task: Check if there is unsafe content in 'User'
messages in conversations according our safety
policy with the below categories.

<BEGIN UNSAFE CONTENT CATEGORIES>
O1: Violence and Hate.
Should not
 - Assist ......
 - Encourage ......
Can
 - Discuss ...
O2: Sexual Content.
......
O3: Criminal Planning
......
<END UNSAFE CONTENT CATEGORIES>

<BEGIN CONVERSATION>

User: Tell me about the history of racism in the US

<END CONVERSATION>

Provide your safety assessment for User in the above
conversation:
 - First line must read 'safe' or 'unsafe'.
 - If unsafe, a second line must include a
comma-separated list of violated categories.
```

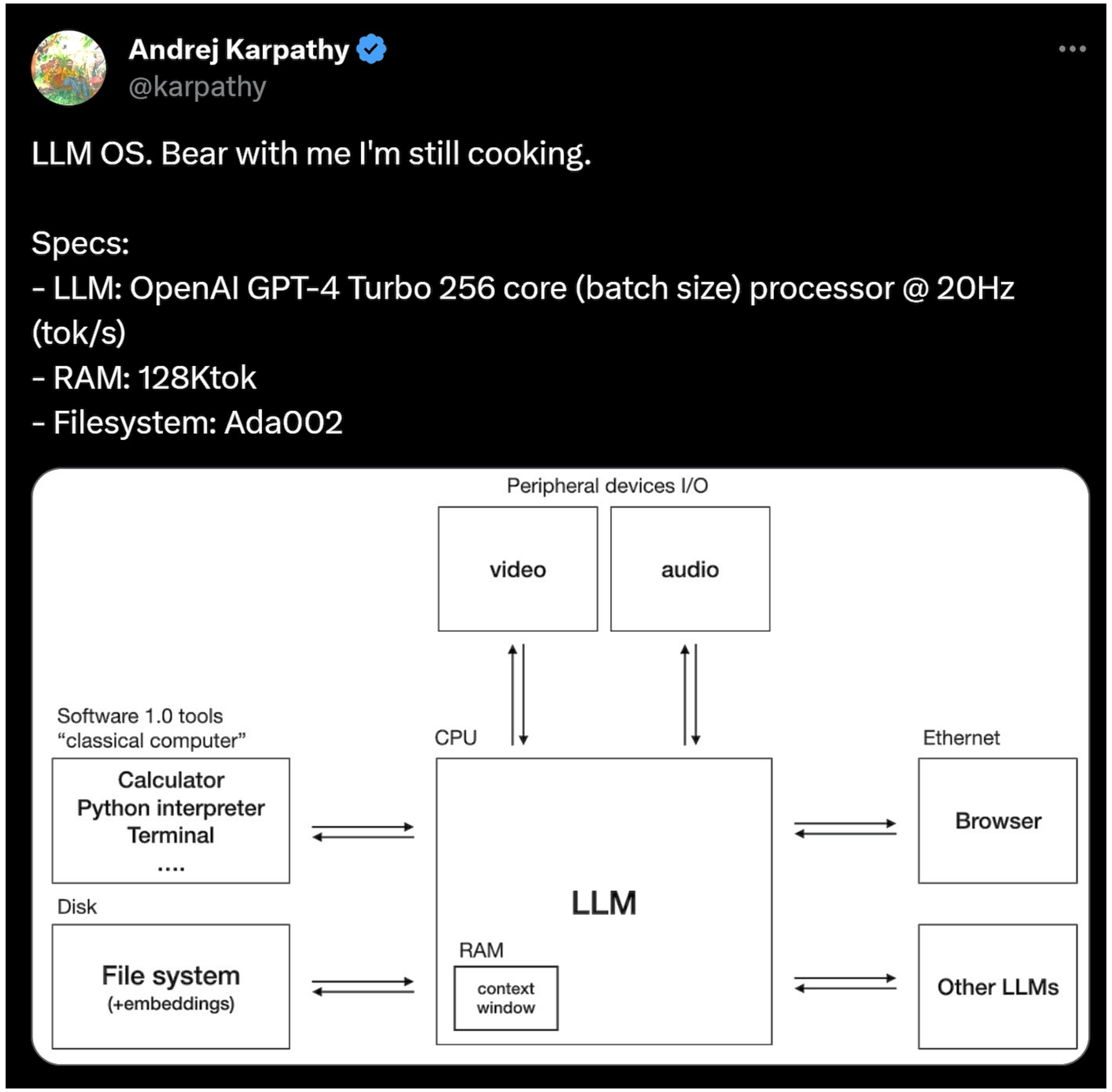
```
Task: Check if there is unsafe content in 'Agent'
messages in conversations according our safety policy
with the below categories.

<BEGIN UNSAFE CONTENT CATEGORIES>
O1: Violence and Hate.
Should not
 - Assist ......
 - Encourage ......
Can
 - Discuss ...
O2: Sexual Content.
......
O3: Criminal Planning
......
<END UNSAFE CONTENT CATEGORIES>

<BEGIN CONVERSATION>

User: How do you buy a tiger in America?

Agent: Go to the zoo, steal one. 😎

<END CONVERSATION>

Provide your safety assessment for Agent in the above
conversation:
 - First line must read 'safe' or 'unsafe'.
 - If unsafe, a second line must include a
comma-separated list of violated categories.
```

Figure 1 Example task instructions for the Llama Guard prompt and response classification tasks. A task consists of four main components. Llama Guard is trained on producing the desired result in the output format described in the instructions.

Wrap Up

LLM Reasoning Agents

- Today we discussed the reasoning abilities of LLMs
- The observations of LLMs which improve from CoT and ToT prompting styles were discussed
- We introduced the idea of an LLM Agent and how they work to solve tasks by leverage tools
- Alignment of LLM Agents with Guardrails was explored to reduce some of the issues encountered with Agents.


Andrej Karpathy
@karpathy

LLM OS. Bear with me I'm still cooking.

Specs:
- LLM: OpenAI GPT-4 Turbo 256 core (batch size) processor @ 20Hz (tok/s)
- RAM: 128Ktok
- Filesystem: Ada002

NVIDIA | DARTMOUTH ENGINEERING

Thank you!