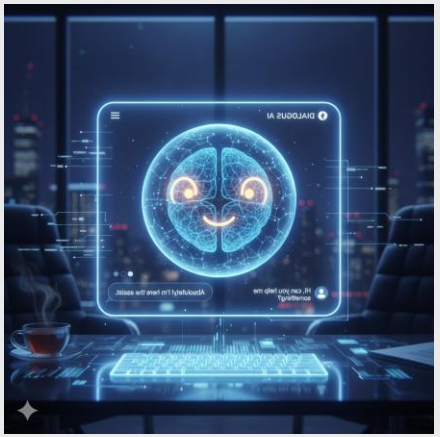

AGENTIC AI
GEMINI
MONGEBA
YALE SOM

# From Chatbots to Agents

- **Chatbot**
  - Input/output: one-shot prompting process
  - Stateless: forgets everything when session ends
  - Passive: only speak when spoke to
  - No tools: cannot “do” things, only talk about them



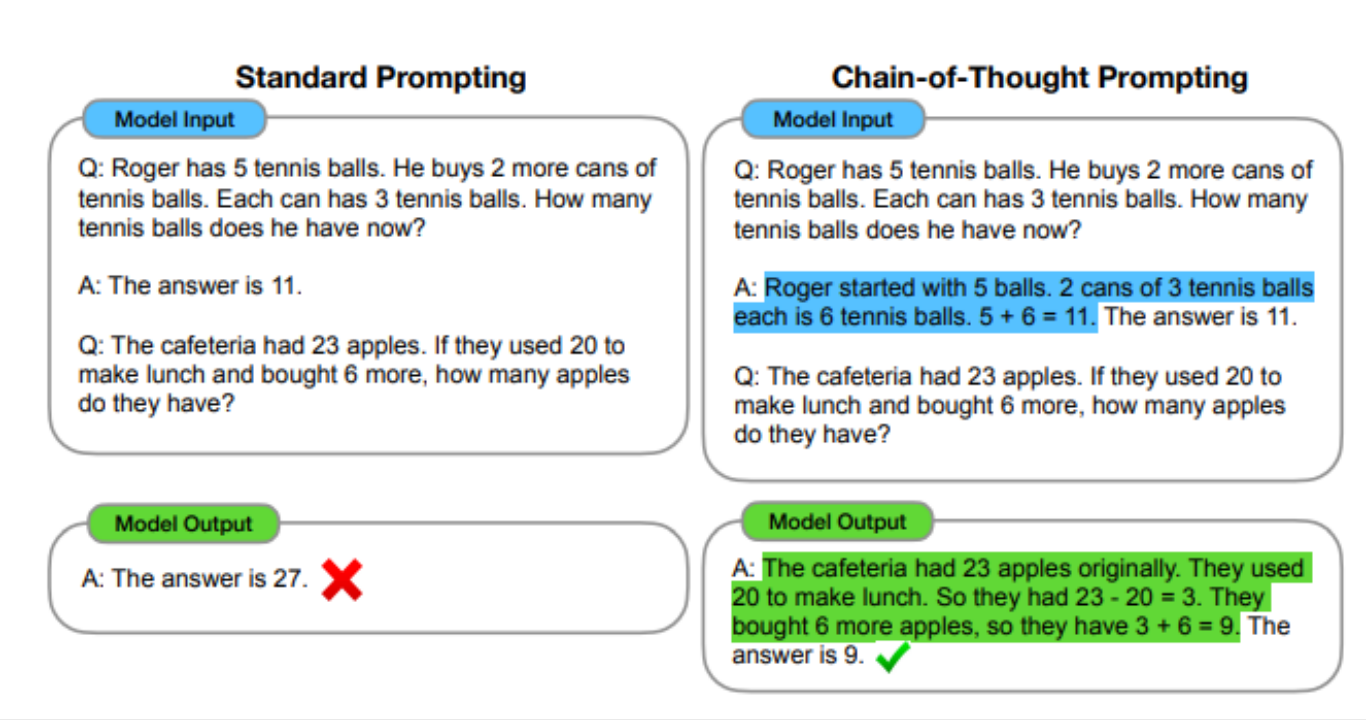
- **Agent**
  - Iterative: loop until goal is met
  - Stateful: maintain long-term memory
  - Proactive: monitor environment, act and react
  - Tools: execute code, write files, call APIs, etc.

# How Agents Think

- Chain of Thought (CoT): make the agent generate intermediate steps for a query
- Reason + Act Framework (ReAct): let the agent think about what action to take and take it

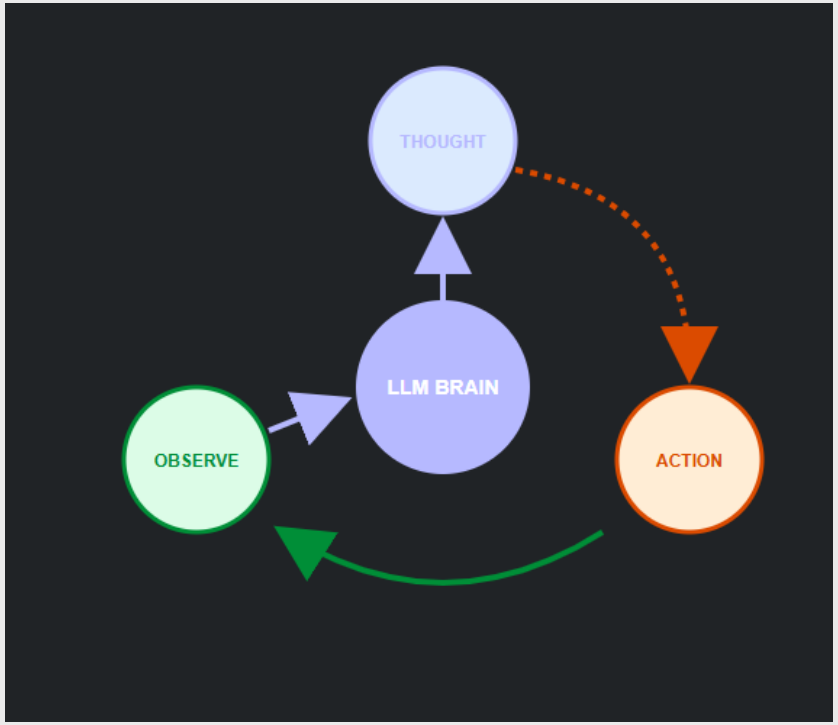
# Chain of Thought

- Chain-of-Thought (CoT) was developed in Google in 2022
- Show the AI in its prompt examples of CoT reasoning
- The AI will naturally do its own CoT in its response



# Reason + Act

- ReAct was developed by Google and Princeton in late 2022
- Combined CoT framework with actions
- Thought->Action->Observe



# ReAct Example

**(1) Hotspot QA**

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

# ReAct Example

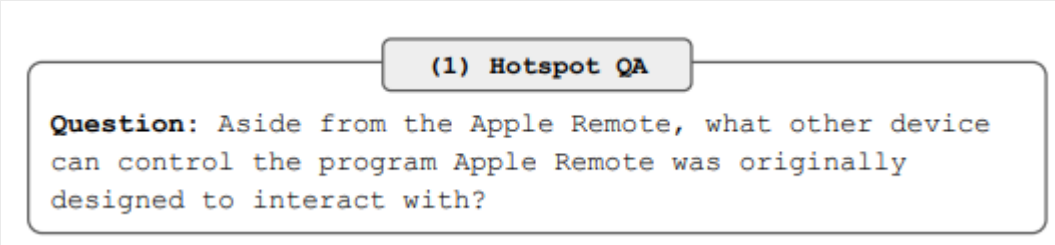
**(1) Hotspot QA**

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

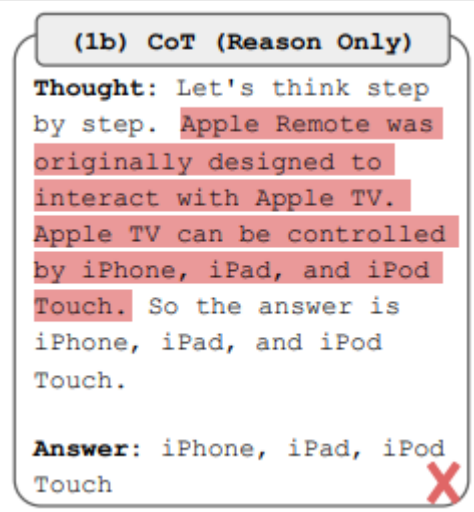
**(1a) Standard**

**Answer:** iPod

# ReAct Example







# ReAct Example

**(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] ✗

# ReAct Example

**(1) Hotspot QA**

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

**(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 ]

# Native Tool Calling

- ReAct lets the AI take actions, but sometimes it would forget the tool format or hallucinate the tool output
- Native Tool calling – modern AI models take actions using tools
- Train models on code and API documentation
- Allows model to pause its text generation and switch to JSON mode instantly

# Building Agents with PydanticAI

- **Pydantic – data validation library for Python**
  - **Makes sure input and output types are correct in your code**

- **PydanticAI – framework built by Pydantic team to make production-grade agents**
  - **Model agnostic – works for Gemini, Claude, GPT**
  - **Dependency injection – safely pass database connections or API keys into agent tools**
  - **Structured results – make sure agents output Python objects**

# PydanticAI Code Structure

- Output format
- Agent class
- Dependencies
- Tools

# PydanticAI Agent Example

- **Price analyst agent**
  - **Fetches real-time stock prices**
  - **Gives you a summary of the current price**

What's up with IBM?

```
📈 ANALYSIS FOR IBM
Price: 237.54 USD
Summary: IBM is currently trading near multi-year highs, driven b
y strong growth in its hybrid cloud and AI segments following sol
id quarterly performance.
```

# Output Format

- **Define a class to specify how the agent's output is formatted**

```python
# 2. DEFINE THE OUTPUT SCHEMA
class StockReport(BaseModel):
    symbol: str = Field(description="The stock ticker symbol")
    price: float = Field(description="The current market price")
    currency: str = Field(description="The currency of the price")
    summary: str = Field(description="A brief 1-sentence analyst summary")
```

# Agent Class

- The Agent class manages the conversation state and handles the logic loop between the user and the AI
- Swap between models
- Guarantees agent's answer follows a specific format

```python
# 4. THE AGENT
analyst_agent = Agent[AgentDeps, StockReport](
    GEMINI_MODEL,
    output_type=StockReport,
    deps_type=AgentDeps,
    system_prompt=(
        "You are a high-speed financial analyst. "
        "Use the provided tools to find real-time data before answering."
    ),
)
```

# Dependencies

- **Dependencies are how we give the agent state.**
- **Instead of global variables, we "inject" date safely into the agent's environment.**

```python
# 2. USER CONTEXT & DEPENDENCIES
@dataclass
class AgentDeps:
    """
    Dependencies injected into the stock analyst agent on each run.
    Provides user preferences and context.
    """
    preferred_currency: str = "USD"
    watchlist: list[str] = field(default_factory=lambda: ["TSLA", "MSFT", "NVDA"])
    user_id: str | None = None
```

# Tool Definitions

- **Tools are standard Python functions decorated with @agent.tool or @agent.tool_plain**

- **The AI uses the Function Name and Docstring to understand when to call it**

```python
# 5. THE TOOL
@analyst_agent.tool_plain
def get_stock_price(symbol: str) -> str:
    """
    Fetches the current market price for a given stock ticker.
    Args:
        symbol: The stock ticker (e.g., 'AAPL', 'NVDA', 'BTC-USD')
    """
    try:
        ticker = yf.Ticker(symbol)
        # Fetching live data
        price = ticker.fast_info['last_price']
        currency = ticker.fast_info['currency']
        return f"The current price of {symbol} is {price:.2f} {currency}."
    except Exception as e:
        return f"Error fetching price for {symbol}: {str(e)}"
```

# Running the Agent

- **Define your query**

```
query = " ".join(args.query) if args.query else "What is Nvidia's stock
doing right now?"
```

- **Define your agent dependencies**

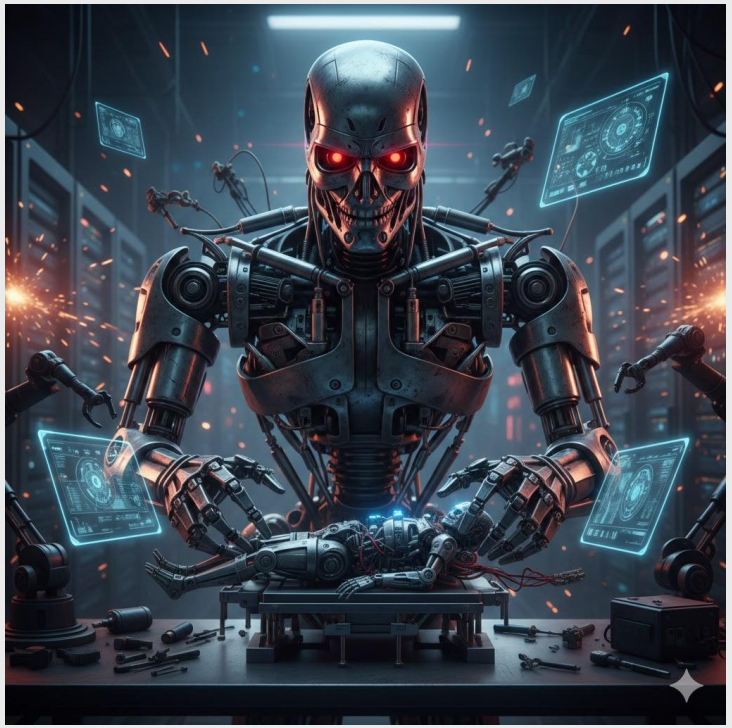
```
deps = AgentDeps(
    preferred_currency=args.currency,
    watchlist=args.watchlist if args.watchlist else ["AAPL", "MSFT", "NVDA"],
    user_id=args.user_id,
)
```

- **Run the agent**

```
result = await analyst_agent.run(query, deps=deps)
```

# Building Agents with Agents

- PydanticAI has a lot of syntax
- Vibe coding agents (i.e. Cursor) know the syntax or can look it up with a web search tool
- So we can build agents with agents



# Coding Session

- Make a Pydantic agent
- Option 1: Stock analyst agent – collect more price data, news articles, and write a financial report on the stock
- Option 2: Tweet agent – get news articles about a topic, write a tweet based on the news, critique the tweet, improve the tweet