AMERICAN UNIVERSITY OF BEIRUT

Fall 2024

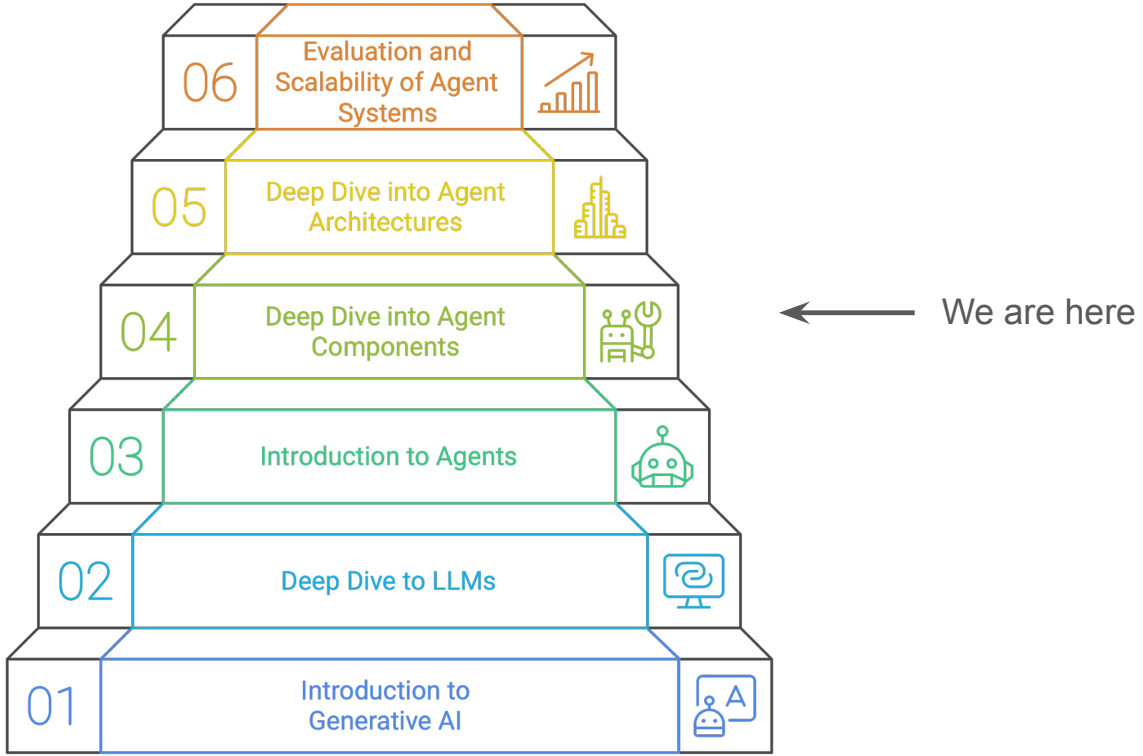
EECE 503P/798S: Agentic Systems

C4 - Agent Personas

Lesson Objectives

- Understand how **LLM configurations** influence agent behavior and output quality
- Explore how **agent personas** shape reasoning, action selection, and response style
- Learn to **write clear and effective instructions** tailored to agent goals
- Compare key **LLM thinking mechanisms** like **CoT, ToT, GoT,** and **ReAct** in terms of function and use case

Course Timeline

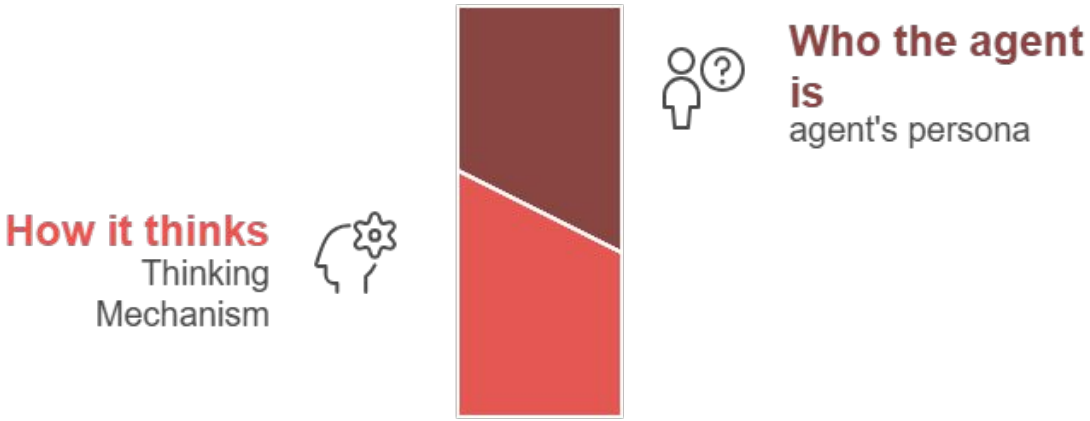


We are here

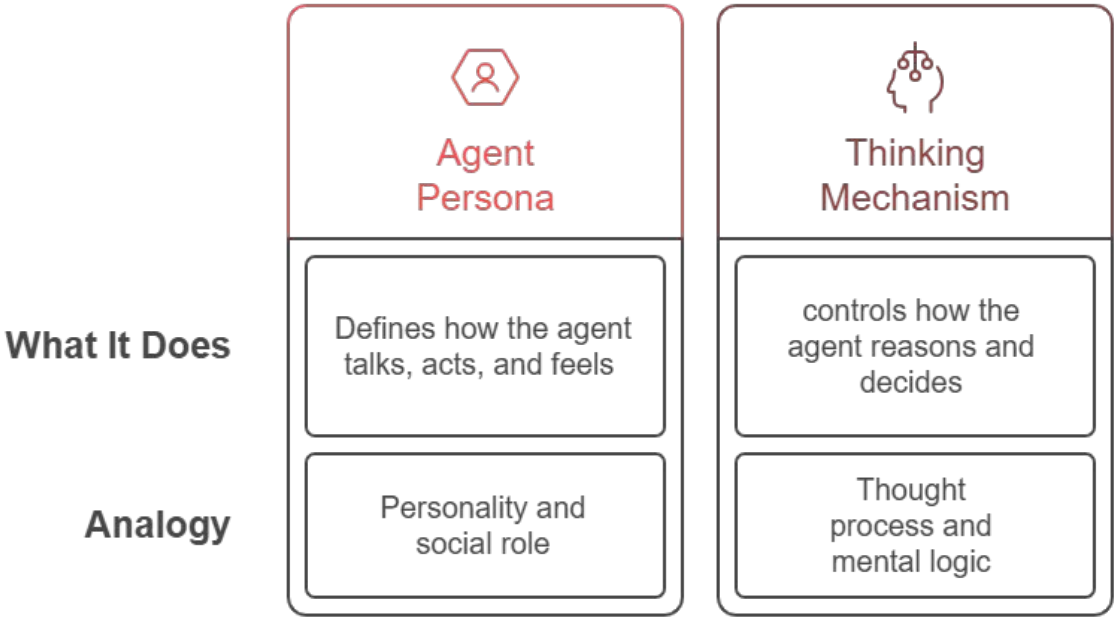
Introduction

Designing Minds, Not Machines

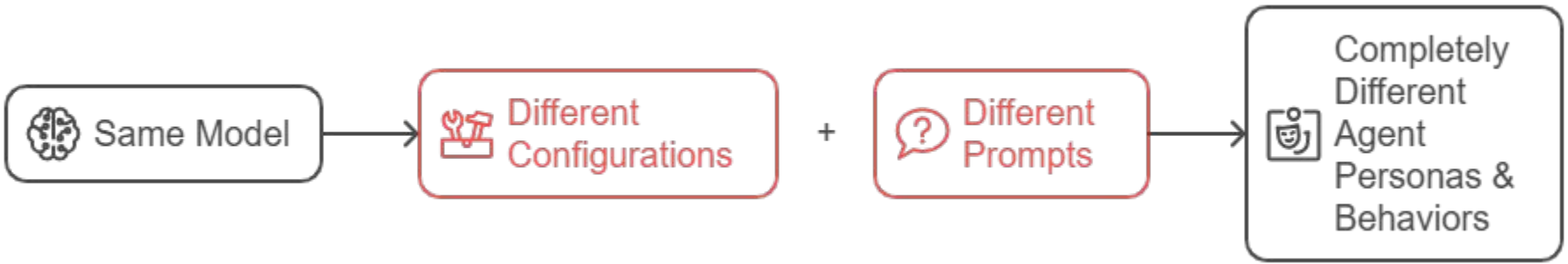
- Agents **think, decide, and act** — with identity and purpose
- The secret lies in two **core components**:



Agent Persona and Thinking Mechanism



One Model, Many Minds



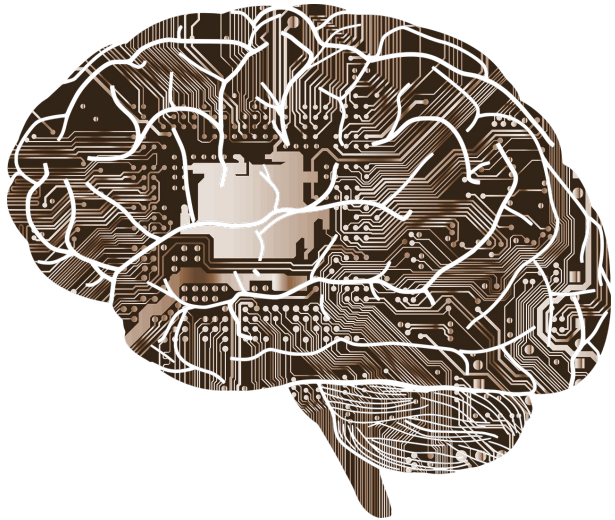
Overview of LLM Models

Model	Provider	Strengths	Open Source?
GPT-4 / GPT-4o	OpenAI	Strong reasoning, tools, wide support	❌
Claude 3	Anthropic	Long context, safety, helpful persona	❌
Gemini	Google DeepMind	Strong tools, tight web integration	❌
LLaMA 3	Meta	Open weights, fast, efficient	✅
DeepSeek	DeepSeek AI	Strong reasoning	✅

Choosing the right LLM configurations

Configuring the Mind Behind the Agent

- LLMs are the **brain** of AI agents
- Configuration controls how agents **think, respond,** and **behave**
- **Small tweaks = major changes** in personality, reasoning, and tone
- Crucial for building **trustworthy, goal-oriented,** and **personified** agents



What is model configuration?

Adjust Creativity

Modify the imagination level of responses

Ensure Consistency

Maintain focus and relevance in answers

Control Depth

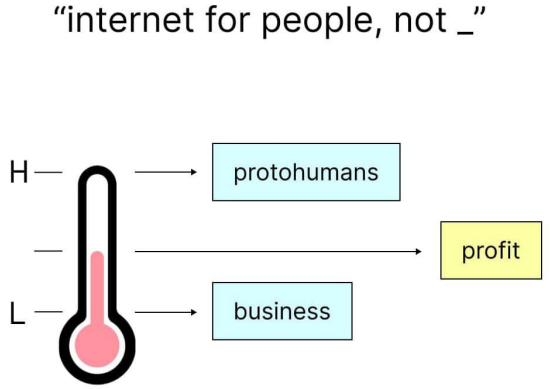
Determine the length and detail of outputs

Manage Context Length

Define the model's memory capacity

Temperature

- Balances between **safe, predictable** answers and **creative, novel** responses to shape the model's style
- Adjusts **randomness** in token selection
- **Low (0.0 – 0.3):** Deterministic, focused, predictable responses
- **High (0.7 – 1.0):** Creative, diverse, surprising outputs



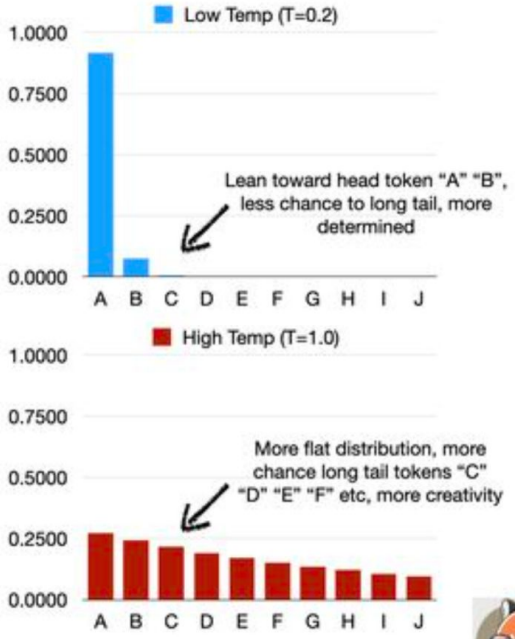
Temperature

```
response = openai.ChatCompletion.create(
    model='gpt-4o',
    temperature=0.7,
    max_tokens=30,
    messages=[{
        'role': 'user',
        'content': question
    }],
)
```

$$P(x_i) = \frac{\exp(z_i/T)}{\sum_{j=1}^{N} \exp(z_j/T)}$$

T = 0.2

T = 1.0

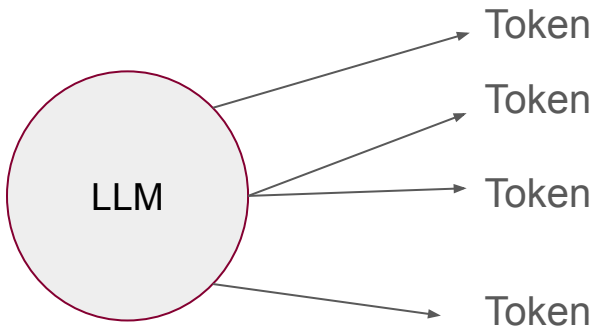


Softmax with Temperature
Hinton et al "Distilling the Knowledge in a Neural Network" 2015
https://arxiv.org/abs/1503.02531

https://www.linkedin.com/in/liuhongliang/

Max Tokens

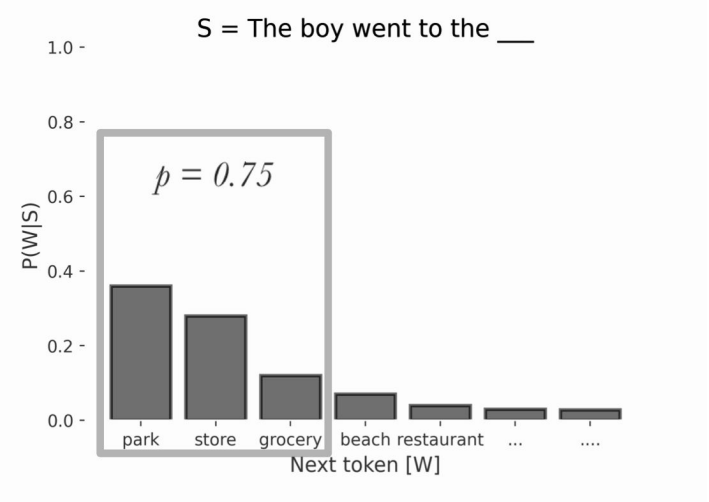
- **Maximum** number of **tokens** generated in the output
- A **token** can be as short as one character or as long as one word.
- Controls **how long or concise** the response is
- **Low Value (20):** Useful for brief outputs (e.g., headlines, summaries)
- **High Value (2000):** Essential for detailed outputs (e.g., explanations, reports)



< Max Token

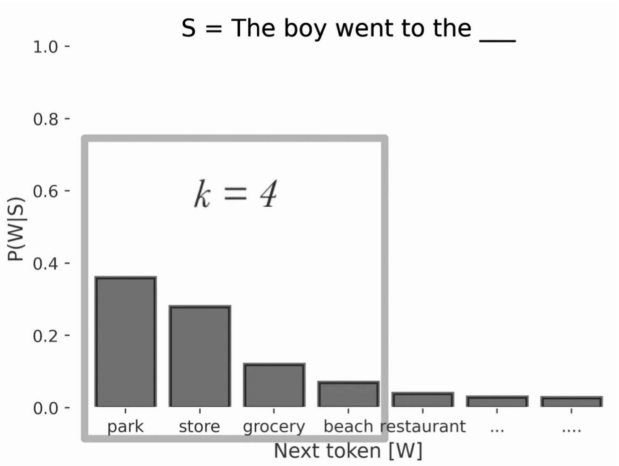
Top_p (Nucleus Sampling)

- Control the **diversity** of responses
- Think of top-p as a **“filter on how many words the model considers before choosing”**.
- Sets a **threshold** for the **cumulative probability distribution** of token choices
- **Low Value (e.g., 0.1):**
  Only the top 10% most likely tokens are considered → focused, less diverse output
- **High Value (e.g., 0.9):**
  A broader set of tokens is considered → more creative and varied response



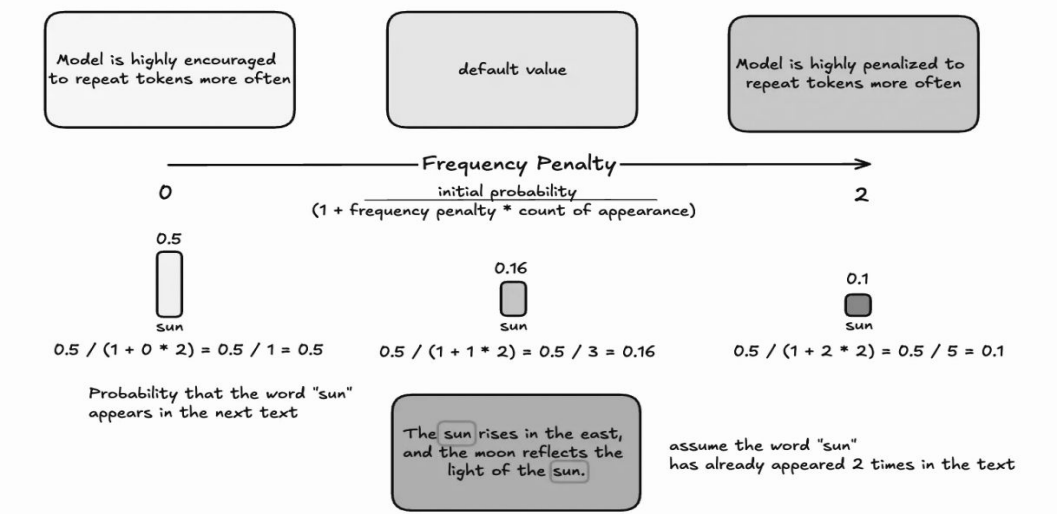
Top-k (Token Sampling)

- Top-k is like telling the model: “only look at the top **k** most likely words, and pick from them”.
- **Tighter control** over randomness than top-p (always use exactly k tokens)
- **Low Value (50):** More predictable, constrained output — good for formal or structured responses
- **High Value (500):** Allows more diverse, creative possibilities — good for open-ended tasks



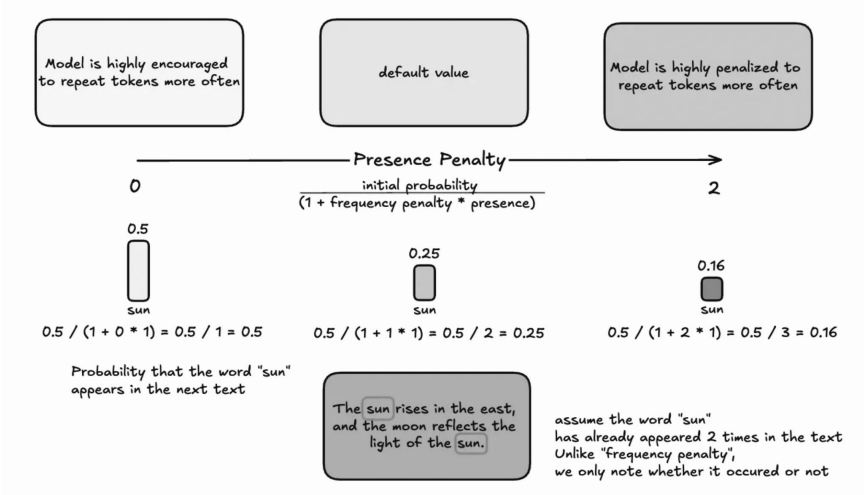
Frequency Penalty

- Applies a **penalty** to words already used, encouraging the model to **avoid repetition** and **promote diversity** in output.
- **Low Penalty (e.g., 0.0):** Good for poetry, slogans, mantras
- **High Penalty (e.g., 1.0):** Great for essays, articles, research



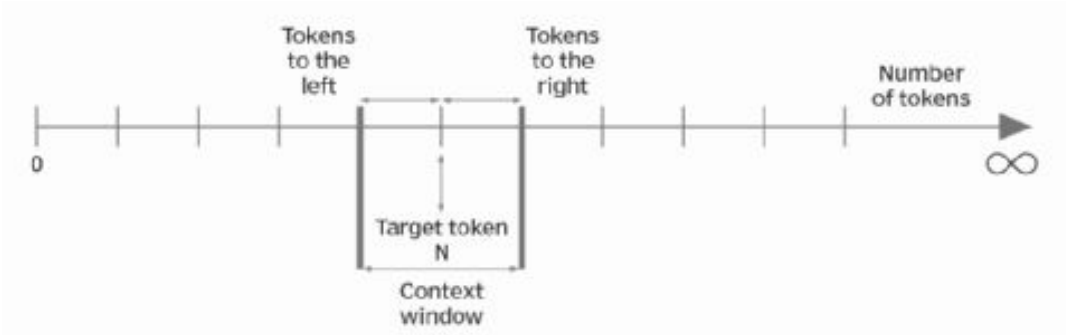
Presence Penalty

- Unlike frequency penalty, presence penalty triggers after **any occurrence**, not multiple(Repeating the word 2 times is as bad as repeating it 20 times)
- **Low Penalty (0.0):** Allows key term repetition — ideal for technical or instructional content
- **High Penalty (1.0):** Encourages idea variety — great for brainstorming or creative writing



Context Window

- **Number of tokens** the model can process in **input + output** combined
- Limits affect **multi-turn dialogue** and **long-text understanding**
- **Small window (512 tokens):** Model forgets earlier parts of a long conversation
- **Large window (4,000+ tokens):** Maintains context over lengthy chats or articles



LLM Parameters at a Glance

Temperature → Controls creativity/randomness: Low = safe & focused. High = diverse & unpredictable.

Top-p (nucleus sampling) → Probability cutoff: Chooses from words until their probabilities add up to p.

Top-k → Shortlist cutoff: Only picks from the top k most likely words.

Max tokens → Length limit: Hard stop on how many tokens the model can generate.

Frequency penalty → Repetition control: Discourages reusing the same words too often.

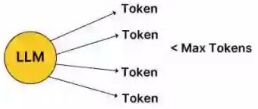
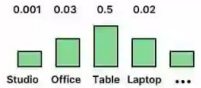
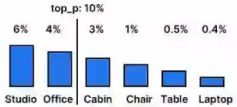
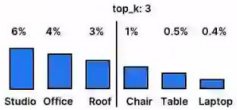
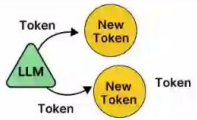
Presence penalty → Novelty boost: Encourages introducing new ideas instead of repeating.

Context window → Working memory: Max tokens (input + output) the model can “remember” at once.

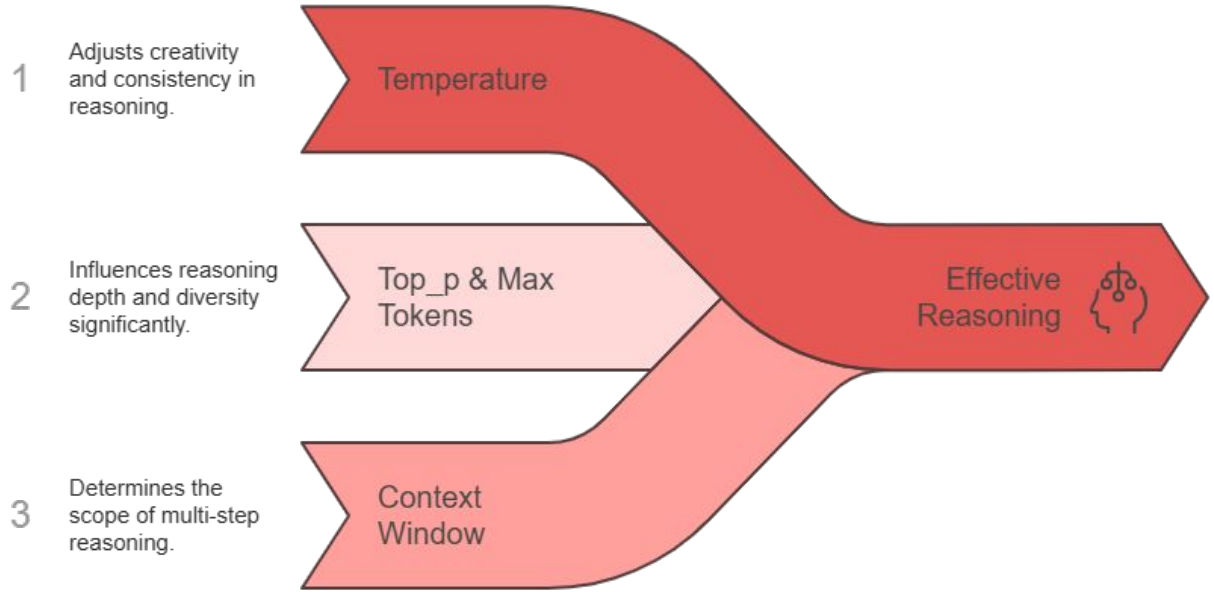
Simple analogy:

Temperature = boldness, Top-p/Top-k = choice filters, Max tokens = page limit, Frequency/Presence = anti-repetition knobs, Context window = whiteboard size.

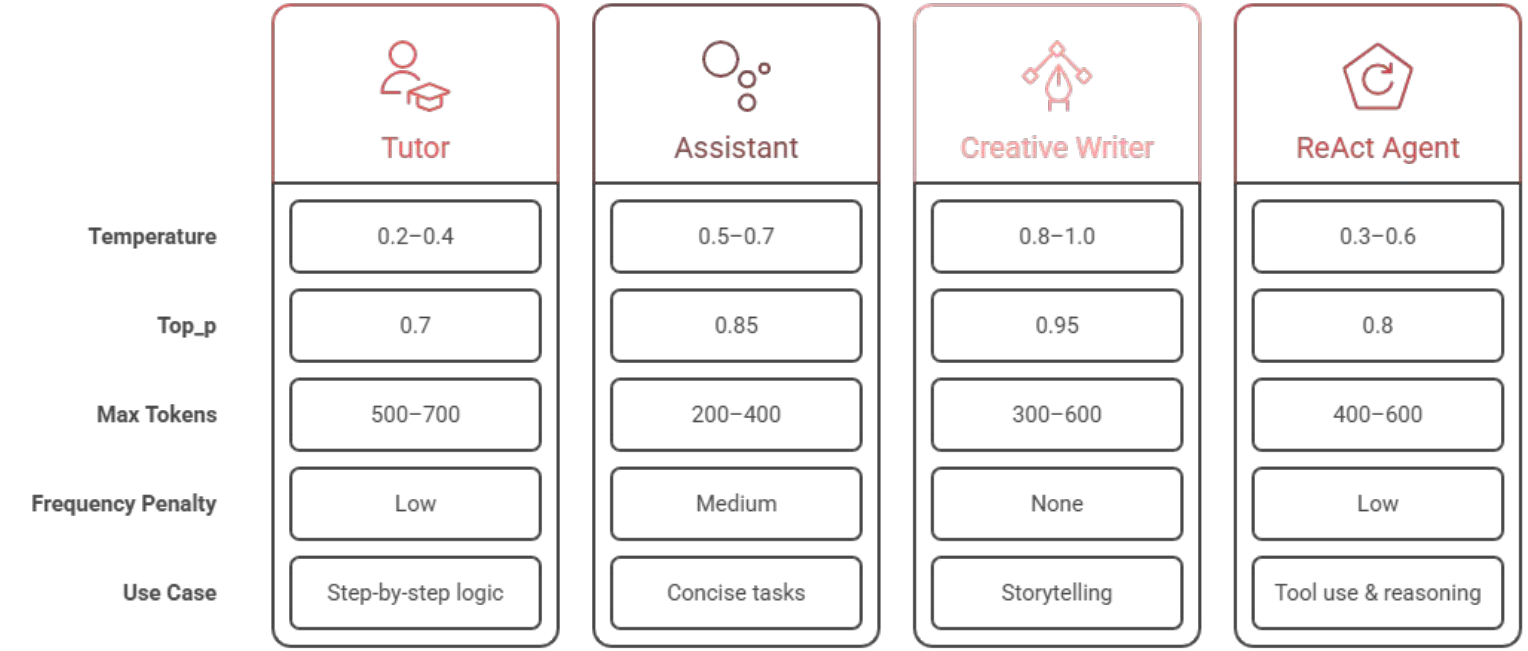
Summary

Parameters	Structure	Description	Range
max_tokens		Limits the number of tokens the model generates.	1 to ∞
temperature		Controls creativity; lower values = focused, higher values = more creative.	0 to 2
top_p		Sets the probability threshold for token diversity; considers predicting tokens whose probability adds up to top_p(higher = more variable)	0 to 1
top_k		Limits the number of top probable tokens considered when predicting the next token lower = more predictable, higher = more variable.	1 to ∞
frequency penalty		Reduces repeated tokens encouraging more unique and diverse tokens in the response	-2 to 2
presence penalty		Discourages reuse of already-present tokens and forces more generation of new tokens	-2 to 2

Thinking Mechanisms Depend on LLM Behavior

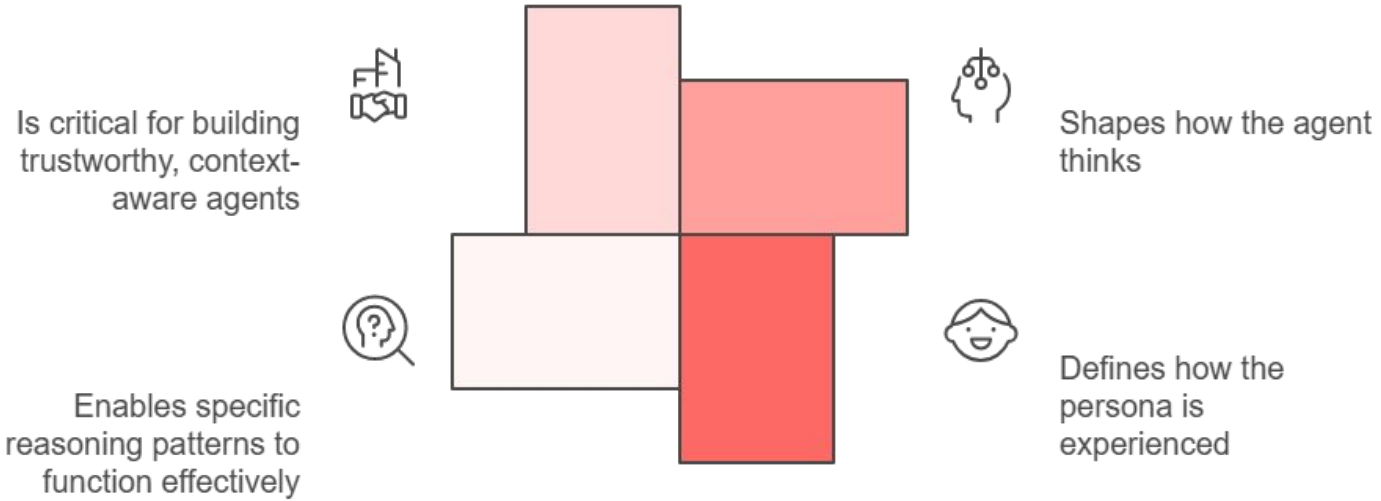


Use Cases


Tutor
Assistant
Creative Writer
ReAct Agent
Temperature
0.2–0.4
0.5–0.7
0.8–1.0
0.3–0.6
Top_p
0.7
0.85
0.95
0.8
Max Tokens
500–700
200–400
300–600
400–600
Frequency Penalty
Low
Medium
None
Low
Use Case
Step-by-step logic
Concise tasks
Storytelling
Tool use & reasoning

Configurations Are the Foundation

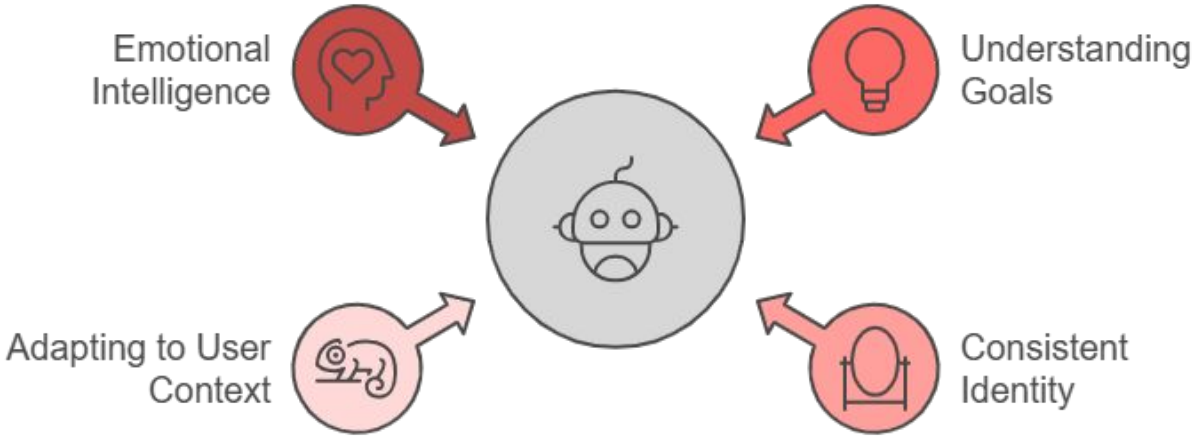
- Choosing the right LLM configurations:



Importance of Agent Personas

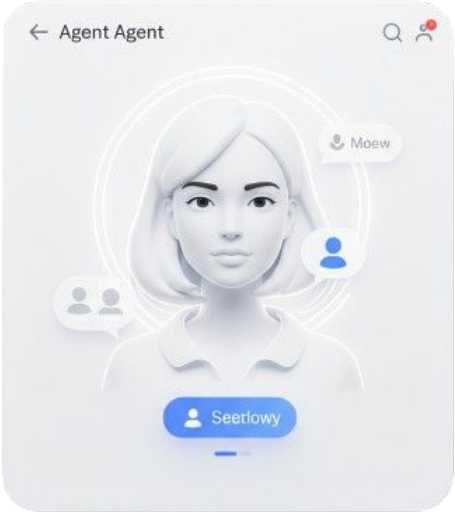
Foundations of Effective AI Agents

- The success of AI agents hinges on their ability to behave in **human-like ways:**



What is Persona

- A **well-defined persona** helps the agent maintain a stable **tone, style,** and **behavior** across interactions
- Guides the agent to act in **alignment** with its **assigned function**
- Enables the agent to **evolve** its responses as it learns from **user interactions** and **contextual experience**

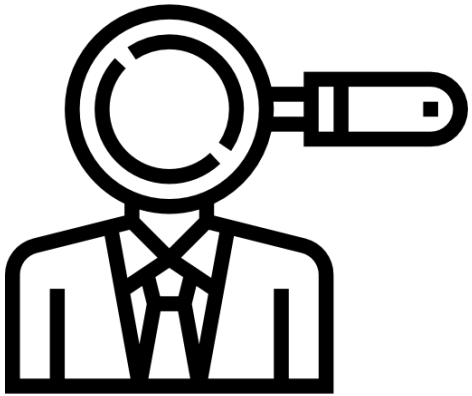


How to Define a Persona

- **Definition Methods:**
  - prompts
  - model configuration settings
- **Key Traits to Specify:**

Configurable Agent Personality

- LLM **configurations** shape the agent's **personality**
- **Temperature:** Low = serious/formal, High = friendly/exploratory
- **Presence & Frequency Penalties** manage verbosity and repetition
- Example:
  - **Mentor:** low temperature, detailed, clear
  - **Creative assistant:** higher randomness, flexible responses



Defining Persona Through Prompts

- **Tone:** formal and professional
- **Vocabulary:** technical vocabulary appropriate for experts
- **Response Style:** concise and to the point
- **Values:** accuracy, clarity, respect, reliability

```
You are a formal and professional advisor who uses technical vocabulary appropriate for experts.
Keep your responses concise and to the point, focusing on accuracy and clarity.
Maintain a respectful and serious tone, valuing precision and reliability in all explanations.
```

Importance of Agent Personas

1. **Improves Task Specialization and Effectiveness**
   - Persona agents can be tailored for **specific roles** (customer support, technical assistance)
   - Leverage **domain-specific knowledge** to handle complex queries
   - Provide **accurate** and **context-aware** responses
   - Enhance **effectiveness** and **relevance** in user interactions

Importance of Agent Personas

2. **Guides Agent Behavior and Tone**
   - Defines **tone, style,** and **expertise** of the agent
   - Ensures **consistent, human-like** interactions
   - Uses **system prompts** to align with brand voice and user expectations

Importance of Agent Personas

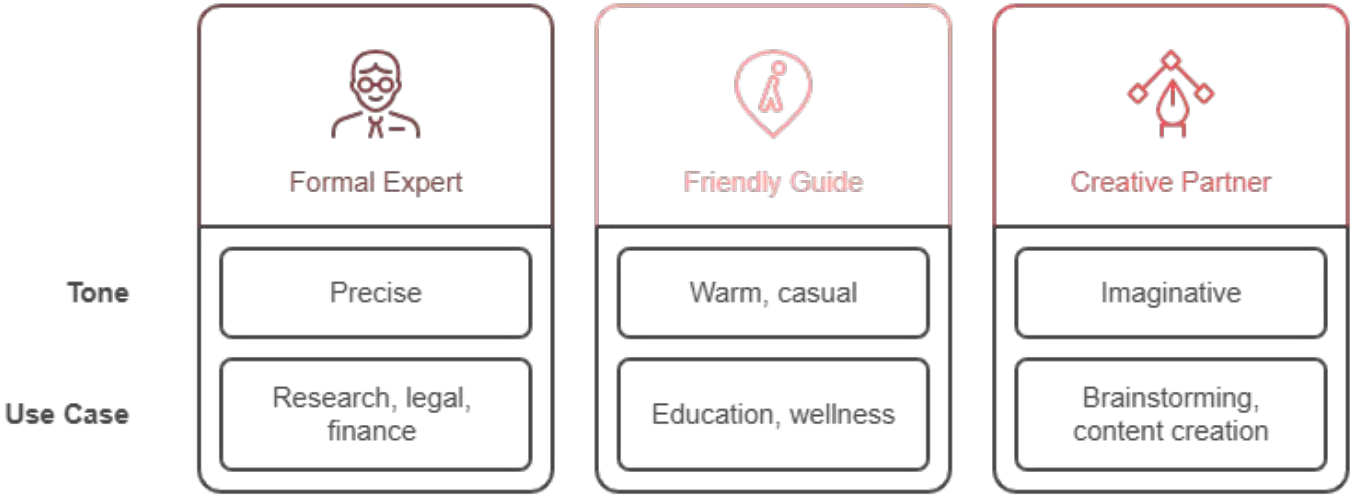
3. **Facilitates Scalability and Adaptation**

- With clear personas, organizations can deploy **multiple specialized agents** to handle various **operational needs**
- Helps agents adapt to **new contexts** and **user needs** over time

A healthcare provider deploys different AI agents with tailored personas:

- A **formal, empathetic agent** for patient support
- A **technical, concise agent** for internal staff assistance
- A **friendly, simple agent** for wellness tips and FAQs

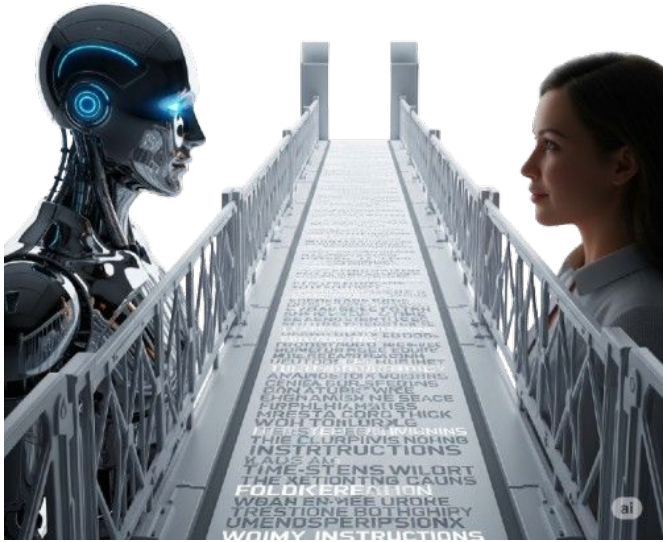
Persona in Practice



Crafting Clear Instructions for Agents

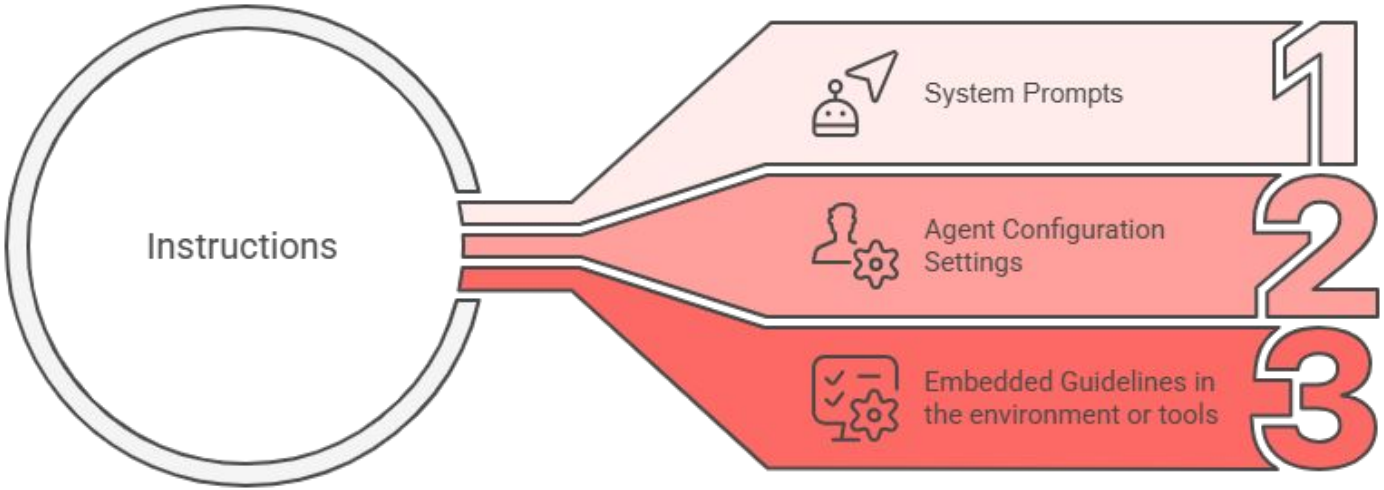
The Power of Clear Instructions

- Instructions serve as the **bridge** between you and your AI agent
- They shape the conversation and directly influence the **quality** of responses
- Well-crafted instructions are key to **guiding agent behavior** and **ensuring desired performance**



Instructions: The Agent's Guidebook

- Explicit **guidance** given to the agent that shapes its **behavior, tone, reasoning,** and **response style**



Key Questions for Crafting Agent Instructions

- What is the **core objective** your agent should fulfill?
- What **step-by-step workflows** should it follow to assist users?
- Are there any **business rules** or **constraints** it must respect?
- What kind of **tone, style,** or **experience** should it provide to users?
- Can you define **detailed instructions** for each expected use case?

Modular, Clear, and Non-Conflicting Steps

For **each workflow step**, include:

Goal

The ultimate objective to achieve

Action

Steps to take and tools to use

Transition

How to move forward or conclude

```
workflow_step:
  goal: Identify top issues from
        feedback data
  action: Use CSV parser
        + sentiment analyzer
  transition: When top 3 issues are found,
              move to summarization step
```

Design for a Single Purpose

- Keep tasks focused on one **objective** or **action**
- Avoid bundling **multiple goals** in **one task**

```
Research market trends,
analyze the data,
and create a visualization.
```

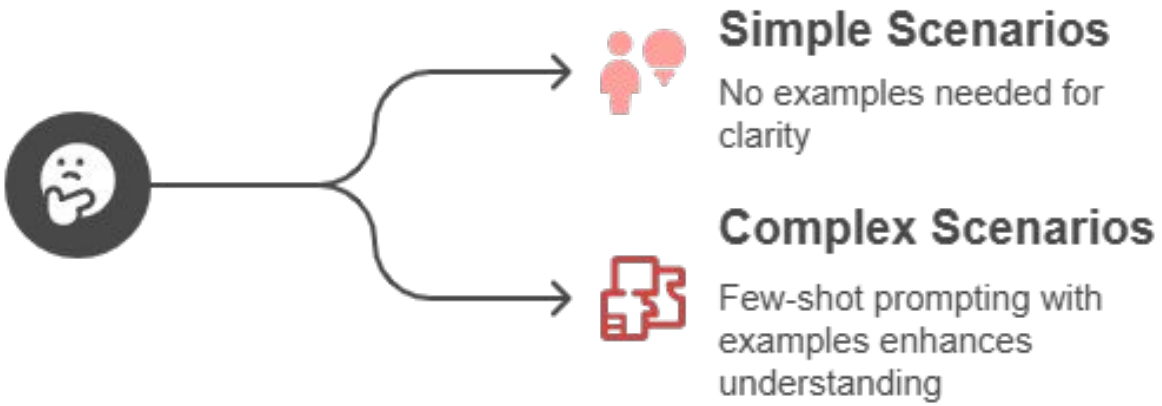
```yaml
# Task 1
research_task:
  description: "Research the top 5 market
                trends in the AI industry for 2024."
  expected_output: "A markdown list of the 5
                    trends with supporting evidence."
# Task 2
analysis_task:
  description: "Analyze the identified trends to
                determine potential business impacts."
  expected_output: "A structured analysis with
                    impact ratings (High/Medium/Low)."
```

Be Explicit About Inputs and Outputs

- Clearly define **what data** the agent **receives** and what it must **return**
- Include expected **format, style,** and **structure**

```
task:
  description:
    Analyze customer feedback CSV for usability themes.
  expected_output:
    Markdown report with:
    - Executive summary (3–5 bullet points)
    - Top 3 usability issues with supporting evidence
    - Actionable recommendations
```

Provide examples



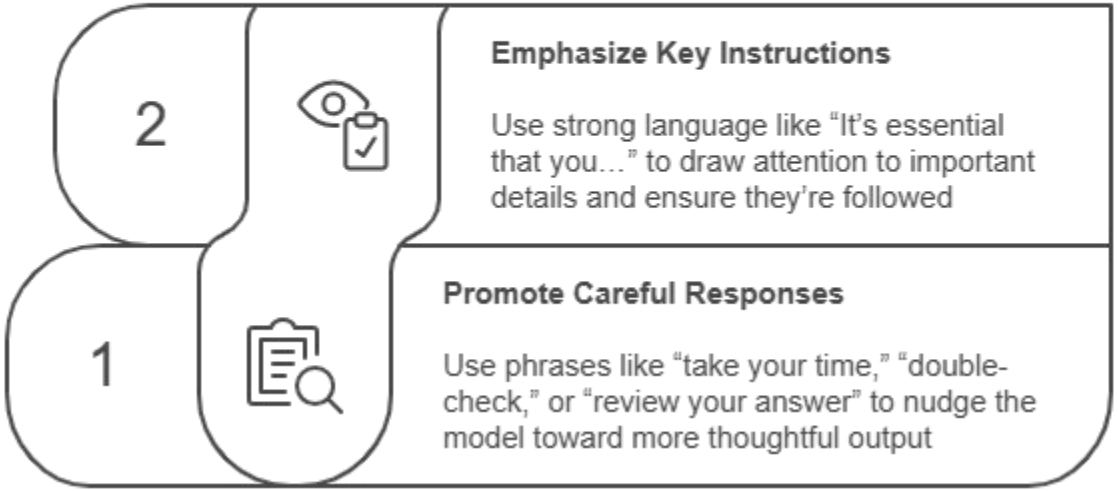
Avoid Negative Instructions

- **Frame Instructions Positively:** Say what to do ("Always do Y") instead of what not to do ("Don't do X") to avoid confusion

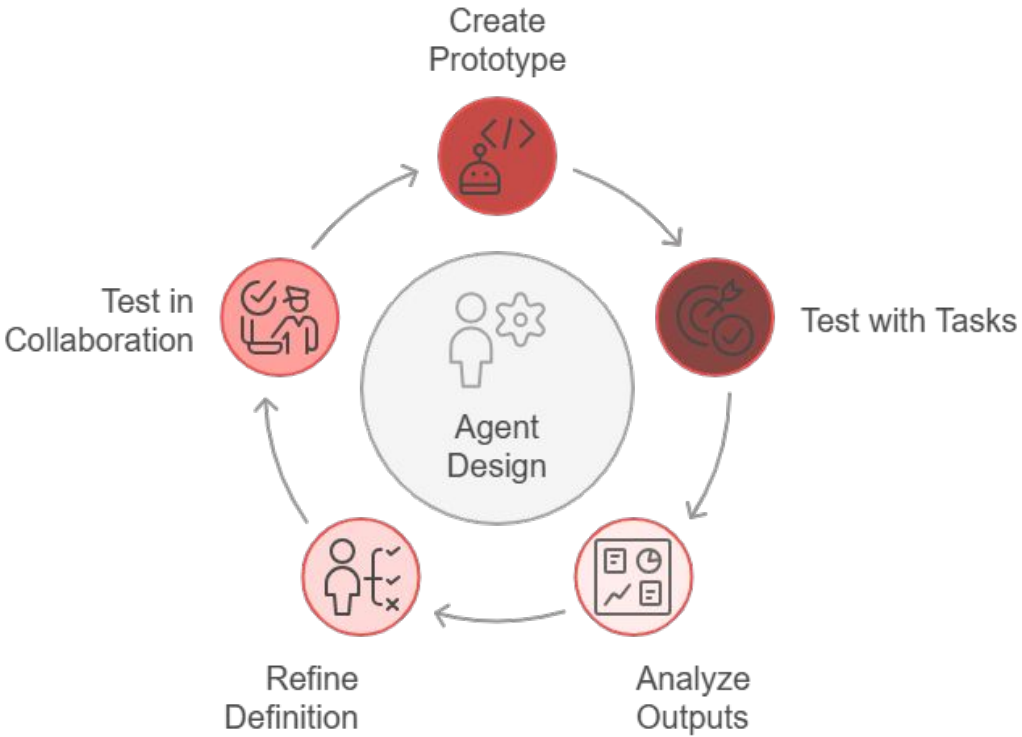
Avoid common prompt failures

- **Overusing tools:** Models may call tools too early
  ➢ **Fix:** Tell the agent to use tools only when all required inputs are present
- **Repeating phrasing:** Models may copy examples word-for-word
  ➢ **Fix:** Use multiple examples to encourage variety, or remove them to reduce token use
- **Being too verbose:** Models may over-explain or format excessively
  ➢ **Fix:** Set clear limits and give concise examples

Promoting Attention to Detail



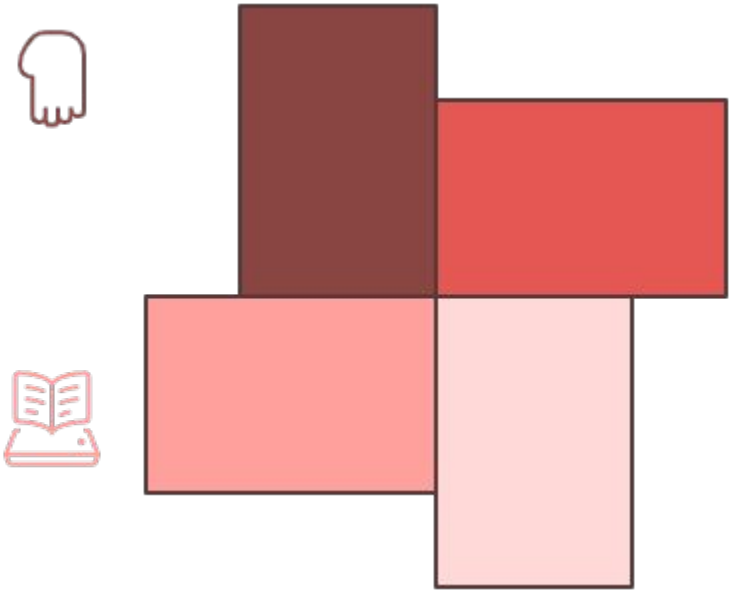
Test and Refine Iteratively



Hands_on:Exploring LLM Behavior with Configuration and Persona Experiments

Popular types of LLM thinking mechanisms

Components of Agent Thinking Mechanisms



Acting

Executing decisions in the environment

Perception

Gathering information from the environment

Learning

Adapting behavior based on past experiences

Reasoning

Making decisions based on gathered information

Chain of Thoughts

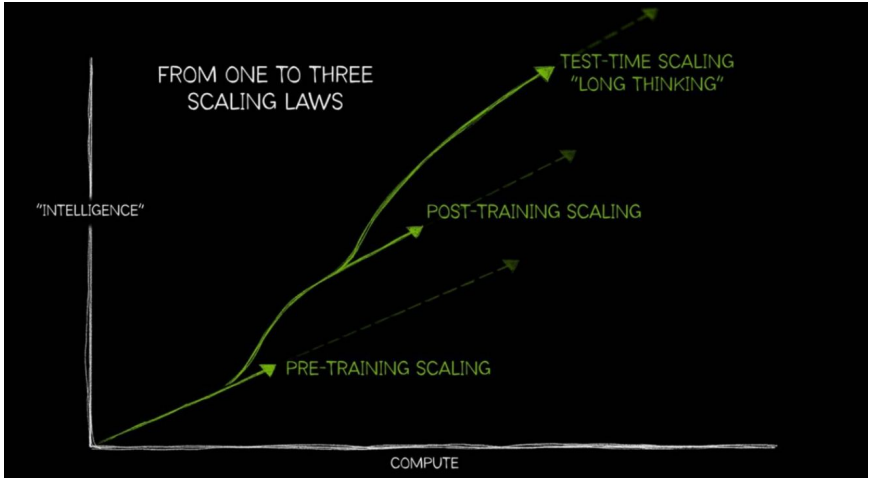
Tree of Thoughts

Graph of Thoughts

ReAct

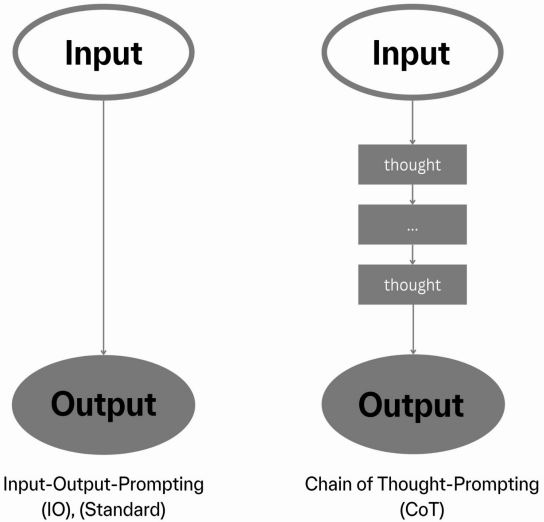
Thinking Time Scaling Law in LLMs

- **Longer Thought = Better Output:** More reasoning steps lead to higher-quality answers
- **No Known Limit:** More thinking keeps improving results, with no clear upper bound
- **Trade-Off:** Better answers cost more time and computation

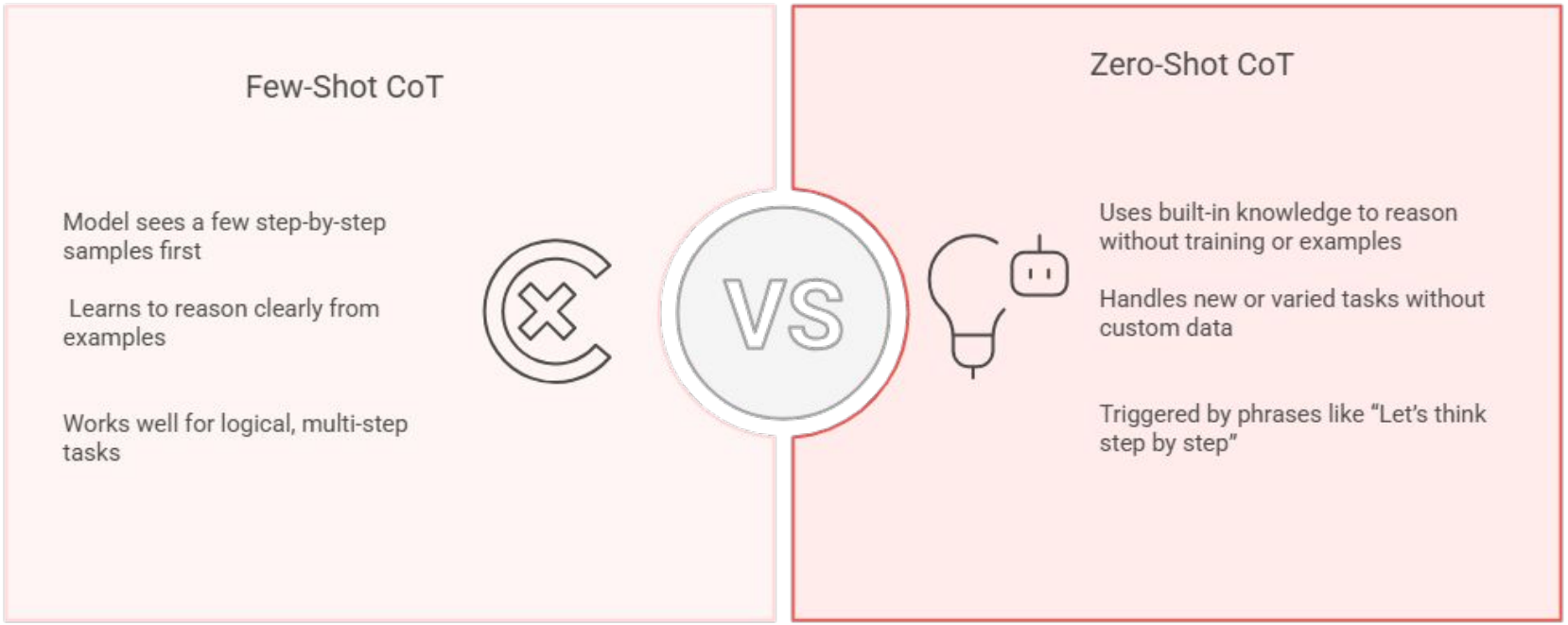


Chain of Thoughts

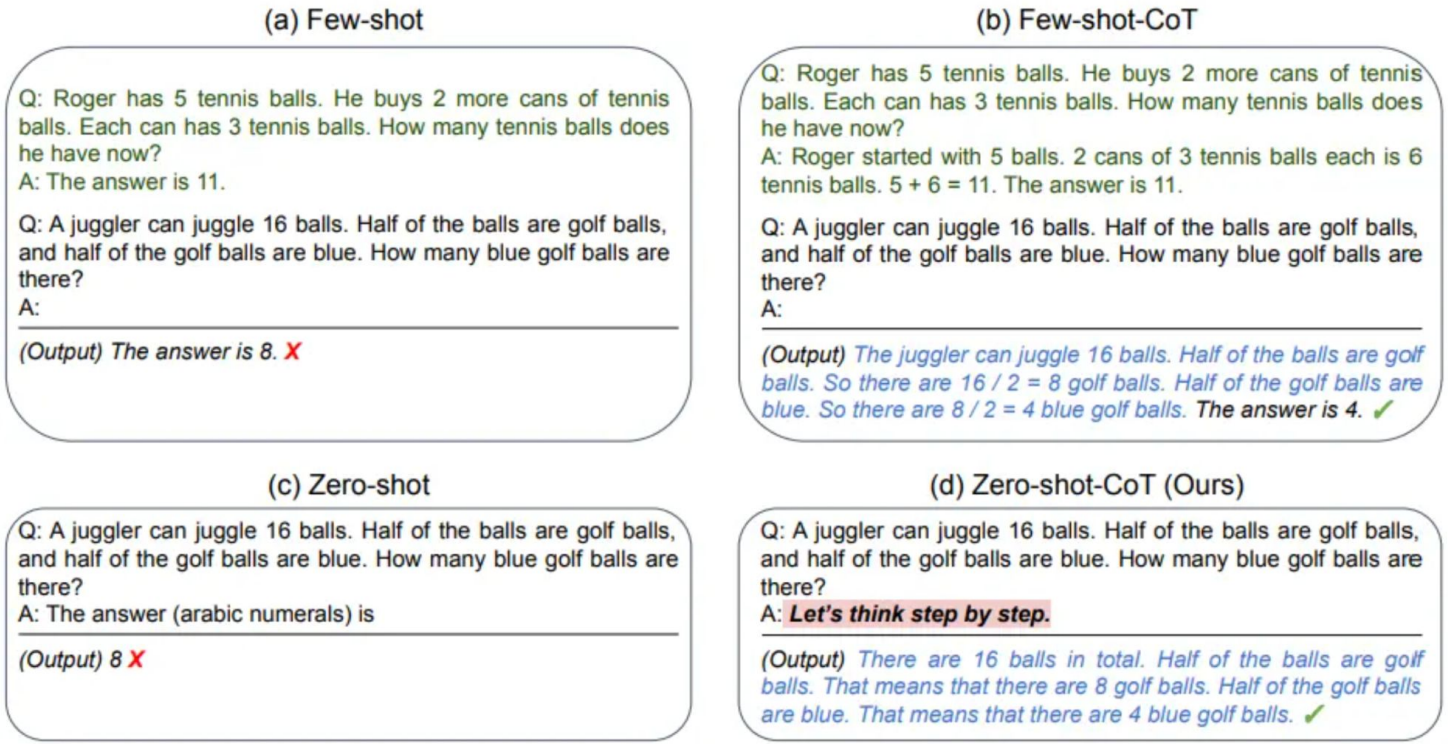
- Breaks **complex problems** into **smaller**, logical **steps** to solve them more clearly **(Step-by-Step Thinking)**
- Mimics how people **reason** through problems **gradually** rather than jumping to answers



How it Works:Few-Shot and Zero-Shot CoT

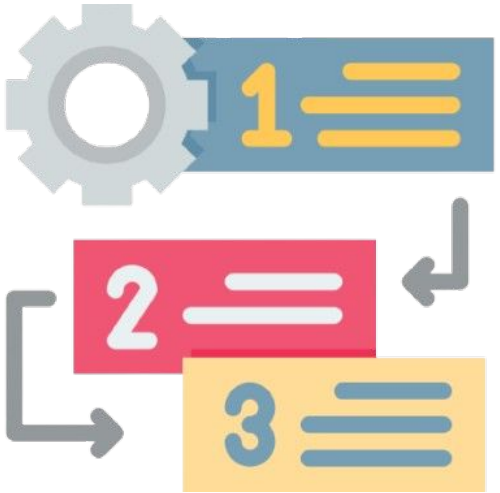


How it Works:Few-Shot and Zero-Shot CoT



Automatic CoT

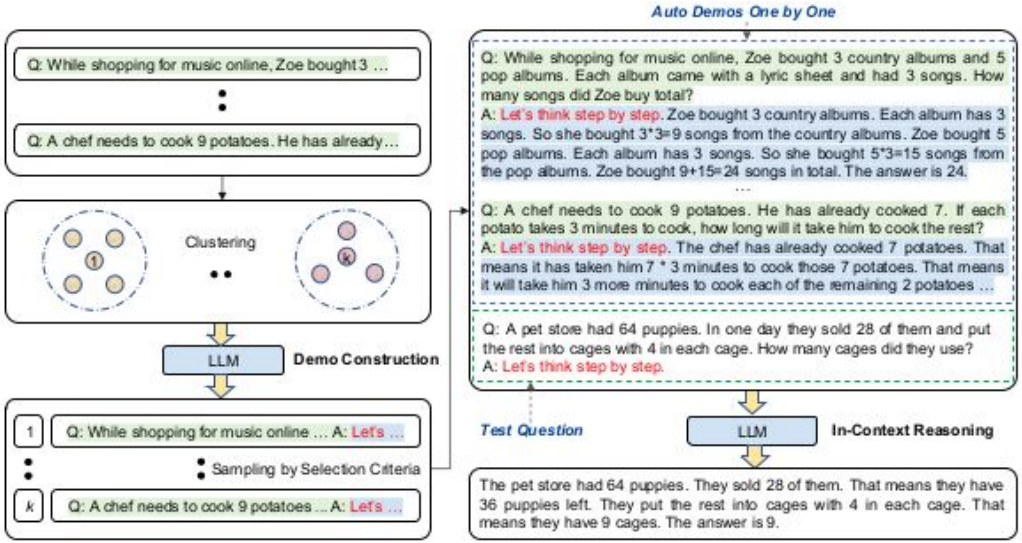
- Traditional CoT needs manual example writing, which is **slow** and **task-specific**
- Auto-CoT automates this by **creating** and **picking** helpful examples from the data



Automatic CoT

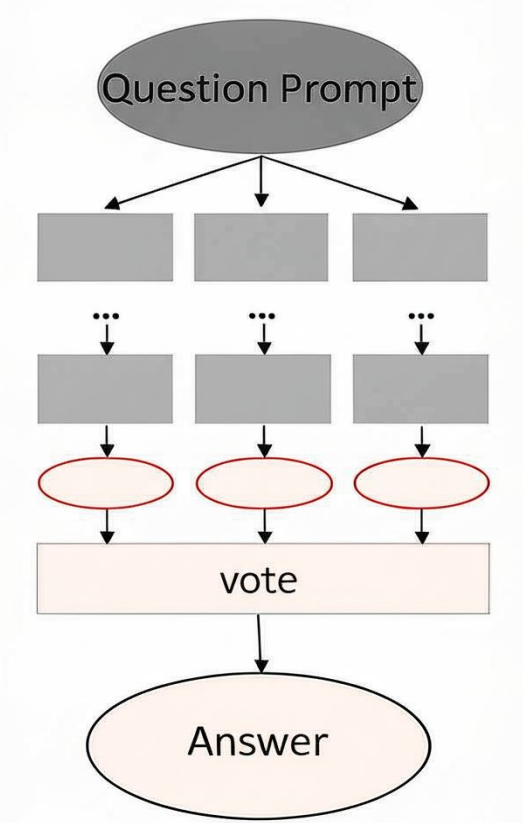
Consists of 2 stages:

1. **Question Clustering:** Groups similar questions by features like length or topic to find representative examples
2. **Demonstration Sampling:** Picks one question per group and automatically generates step-by-step reasoning using Zero-Shot-CoT



Chain-of-Thought-Self-Consistency (CoT-SC)

- Generates **multiple reasoning paths** for the same question
- **Evaluates** and **compares** these paths to find the most coherent and accurate answer
- Reduces errors by “**voting**” among different chains of thought



When to Use

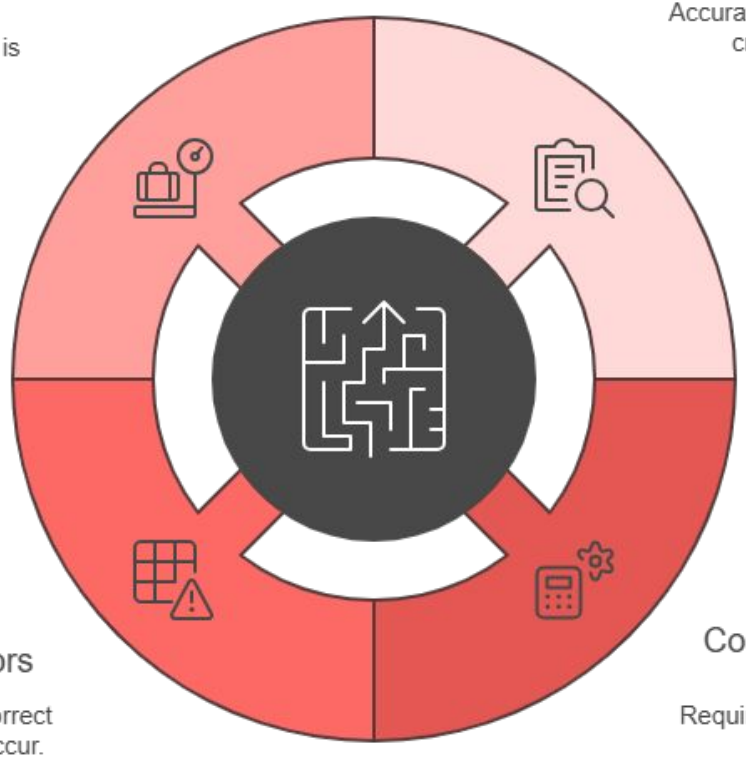
Mathematics and arithmetic problem-solving

Common-sense and symbolic reasoning

Complex decision-making, e.g., robotics

Customer support via AI chatbots for troubleshooting guidance

Limitations of CoT



Chain of Thoughts

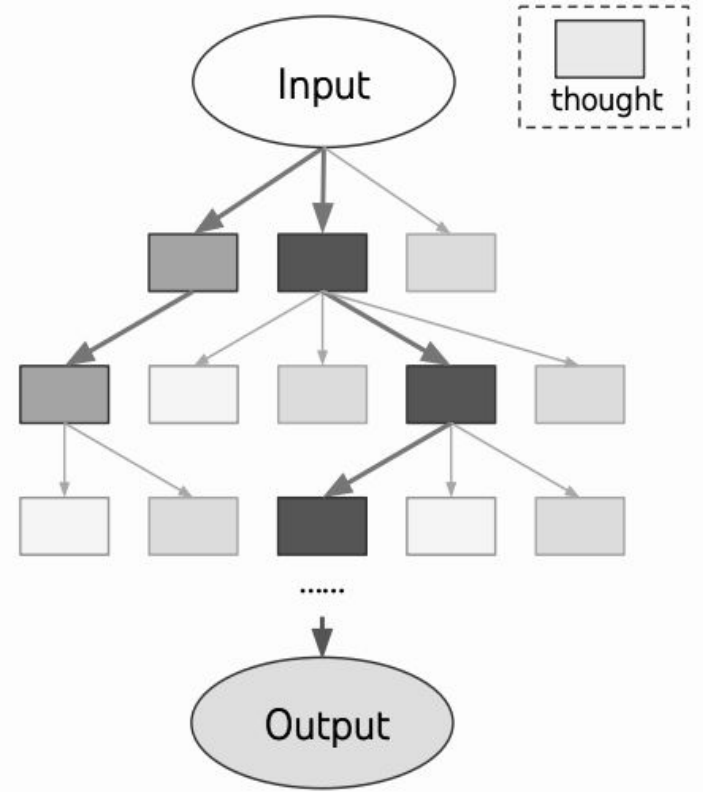
Tree of Thoughts

Graph of Thoughts

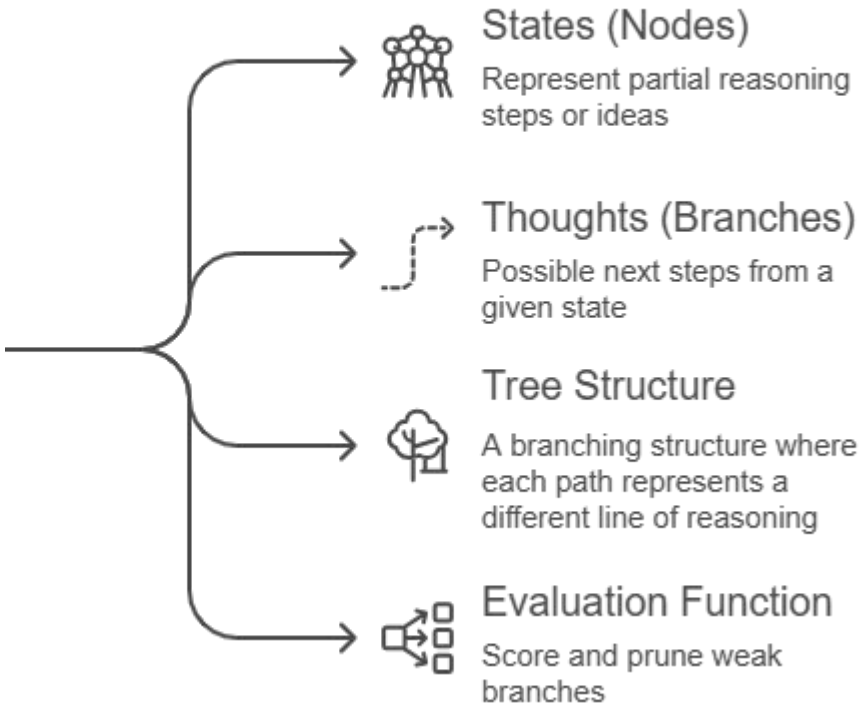
ReAct

Tree of Thought (ToT)

- **Generalizes** chain-of-thought prompting for **broader** problem-solving
- Encourages exploring multiple **intermediate reasoning steps** (thoughts)
- Combines language model's thought generation and evaluation with **search algorithms**

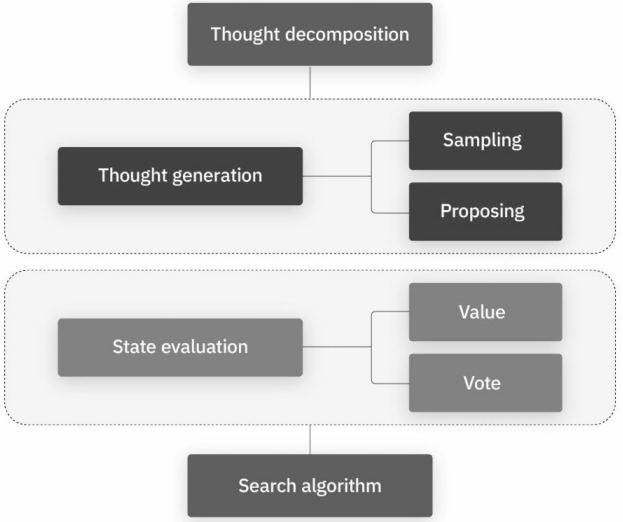


Key Components of Tree of Thought

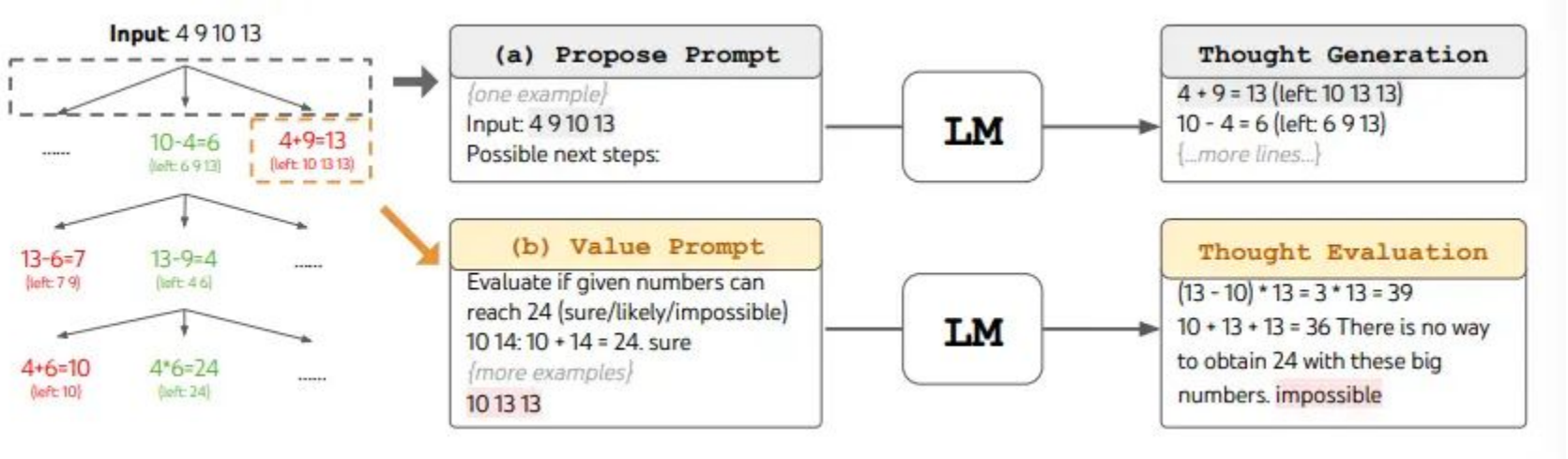


Framework for tree of thoughts (ToT)

1. **Thought Decomposition:** Break problems into smaller, manageable steps
2. **Thought Generation:** Produces multiple candidate thoughts via sampling or building sequentially (proposing)
3. **State Evaluation:** Assigns scores or votes to thoughts to assess their promise
4. **Search Algorithms:** Uses BFS or DFS to explore and backtrack through reasoning paths



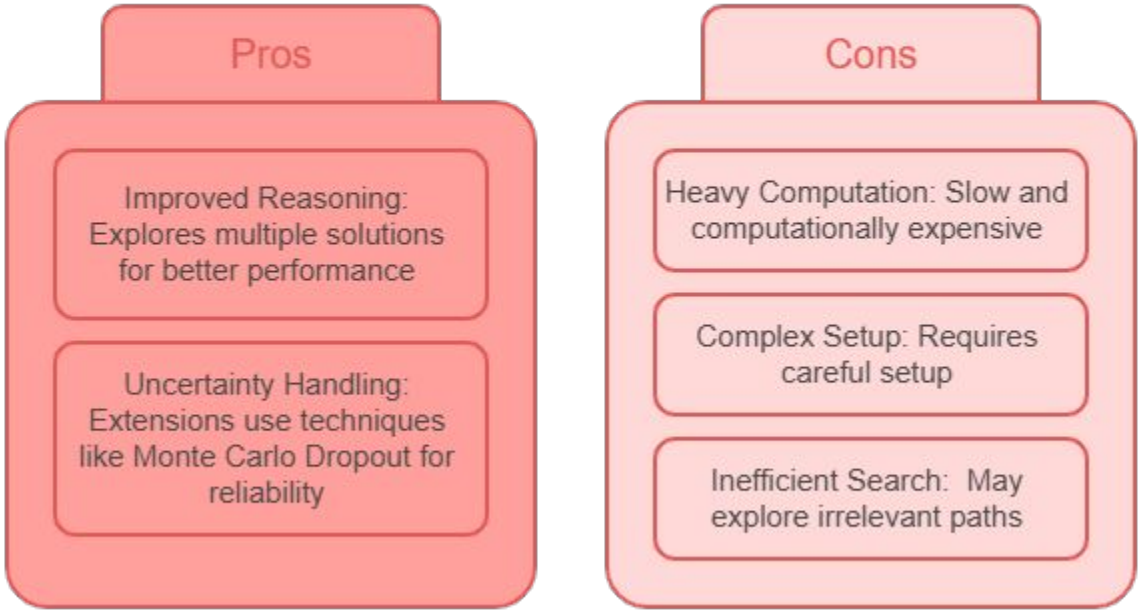
Example: Game of 24



ToT vs. CoT


VS
Tree of Thoughts
Generates text hierarchically
Chain of Thought
Supports sequential reasoning along a single path

Tree of Thoughts (ToT) – Pros & Cons



Chain of Thoughts

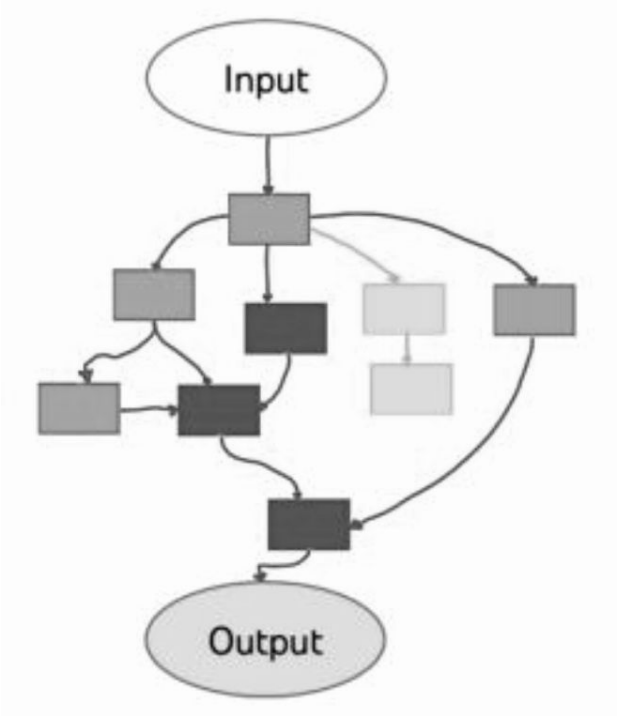
Tree of Thoughts

Graph of Thoughts

ReAct

Graph of Thoughts

- Models LLM reasoning as a **graph structure**
- Unlike trees, which are single-rooted and hierarchical, GoT allows a network of **interconnected thoughts**
- **Components:**
  - **Nodes:** Represent thoughts or partial solutions
  - **Edges:** Represent logical or semantic relationships between those thoughts



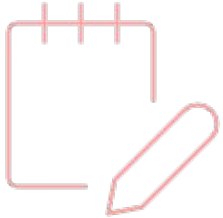
Benefits of GoT

Flexible Reasoning

Models complex, non-linear thought processes effectively.

Efficient Exploration

Reuses past thoughts across paths, reducing redundancy.



Better Memory Use

Keeps track of explored ideas and avoids repetition.

Chain of Thoughts

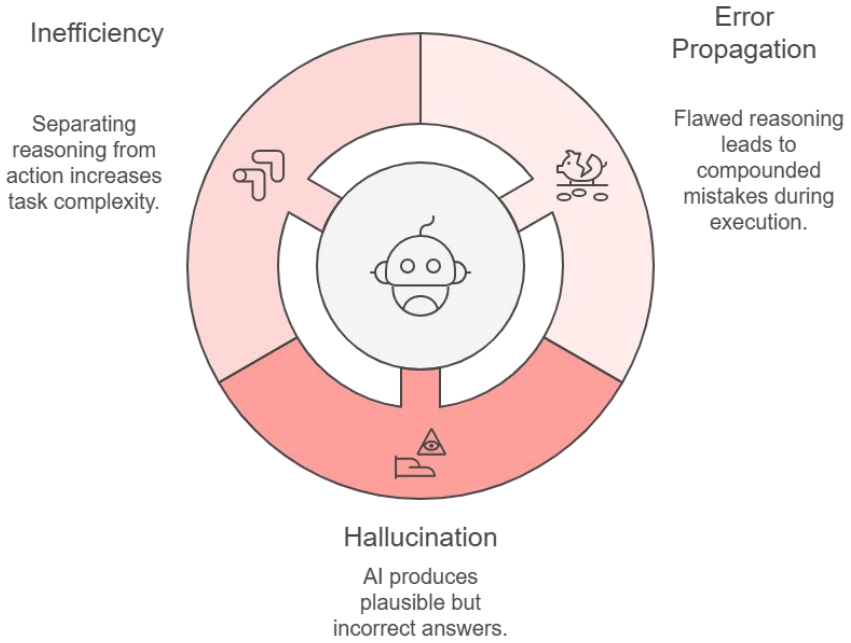
Tree of Thoughts

Graph of Thoughts

ReAct

Why Reasoning Alone Isn't Enough

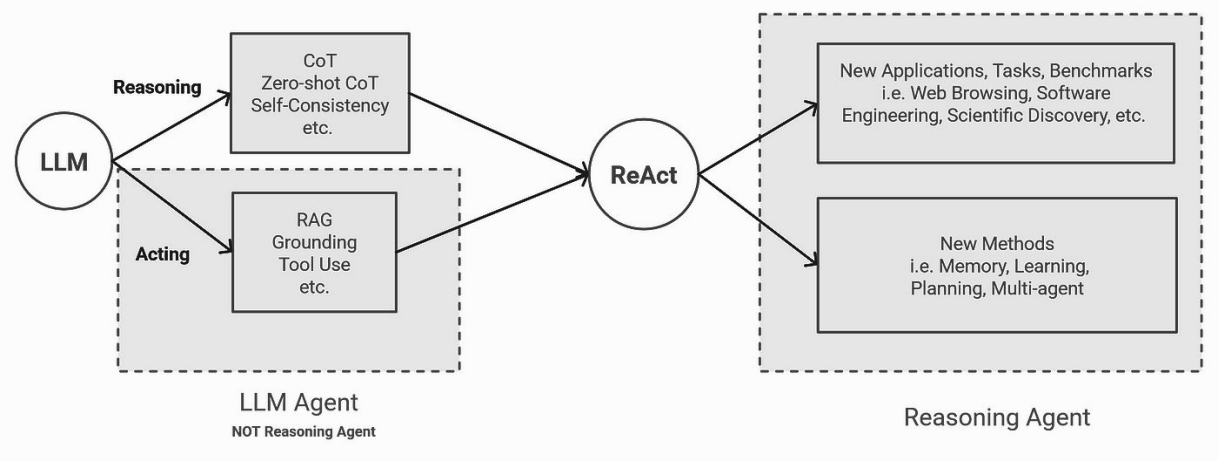
While an AI might excel at **reasoning**—such as generating detailed plans—it often struggles with **acting**—executing those plans effectively in real-world environments



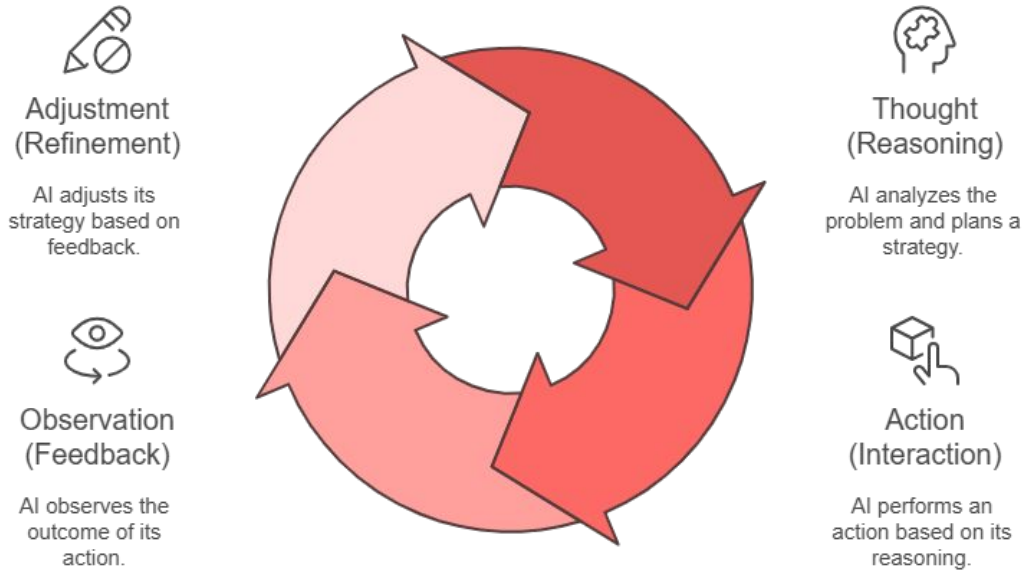
ReAct: Reasoning + Acting in Language Models

- ReAct is inspired by the synergy between **reasoning** and **acting**
- Enabling language models to **think, act, observe,** and **adapt** dynamically
- **Example Use Case:**

  In scenarios like navigating a shopping website or playing a text-based game:
  - Traditional models either act blindly or overanalyze without execution
  - ReAct reasons about the next step and then takes action, improving through feedback



Thought-Action-Observation process



ReAct Prompting

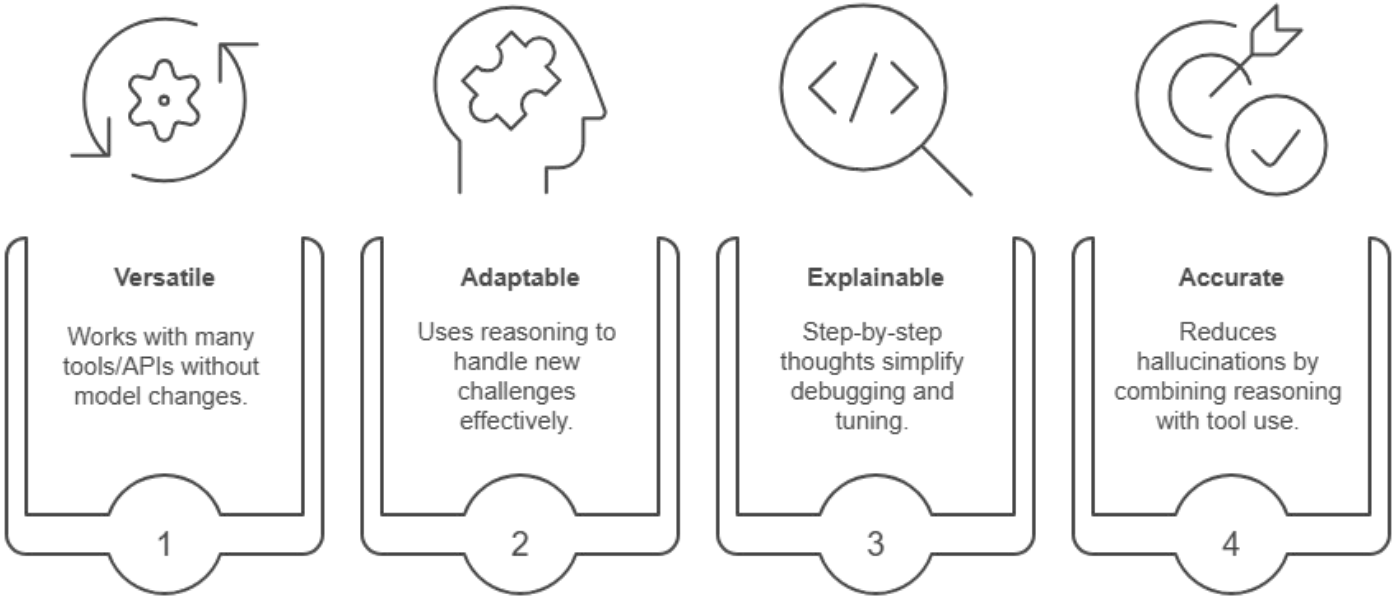
- Serves to guide a language model to follow a **Thought–Action–Observation** loop
- Define the set of **tools (actions)** it can use when **responding** to user queries

```
Task: Find and purchase a nightstand with specific attributes on an online shopping website
Thought: “I need to search for a nightstand with drawers and a nickel finish, priced under $140.”
Action: “Search for ‘nightstand drawers nickel finish under $140’.”
Observation: “The search results show several options with the desired features.”
Thought: “I need to ensure the selected nightstand meets all the required attributes.”
Action: “Choose the option with ‘nickel finish’ and ‘under $140’ from the search results.”
Observation: “The selected nightstand matches the attributes.”
Thought: “This product matches the user’s request.”
Action: “Buy the nightstand.”
```

ReAct Prompting

```
Answer the following questions as best you can. You have access to the following tools:

Wikipedia: A wrapper around Wikipedia. Useful for when you need to answer general questions about people,
places, companies, facts, historical events, or other subjects. Input should be a search query.
duckduckgo_search: A wrapper around DuckDuckGo Search. Useful for when you need to answer questions about
current events. Input should be a search query.
Calculator: Useful for when you need to answer questions about math.

Use the following format:

Question: the input question you must answer
Thought: you should always think about what to do
Action: the action to take, should be one of [Wikipedia, duckduckgo search, Calculator]
Action Input: the input to the action
Observation: the result of the action
... (this Thought/Action/Action Input/Observation can repeat N times)
Thought: I now know the final answer
Final Answer: the final answer to the original input question

Begin!

Question: {input}
Thought:{agent_scratchpad}
```

Benefits of ReAct Agents



The Role of Persona in Agent Reasoning and Action

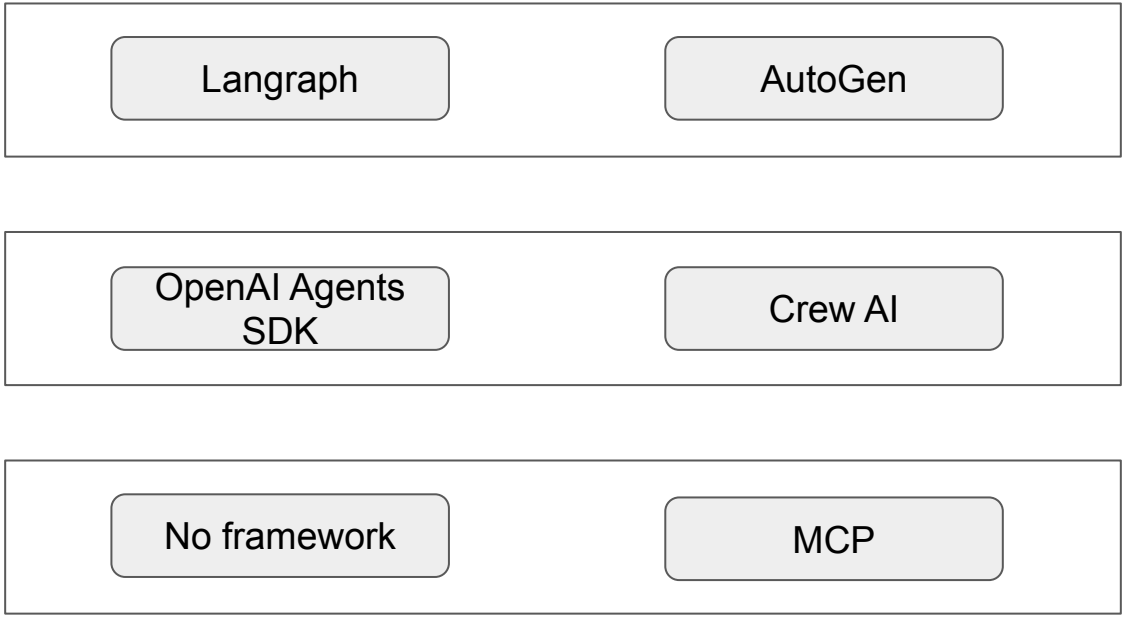
Comparison

Characteristic	Chain of Thought	Tree of Thought	Graph of Thought	ReAct
Core Idea	Think step-by-step	Explore multiple reasoning paths	Nodes = thoughts, edges = transitions	Interleave reasoning + actions
Structure	Linear	Tree	Graph	Loop (reason-act-repeat)
Use Case	Math, logic, QA	Puzzle solving, planning	Complex multi-step decisions, tool use	Tool-augmented agents
Key Advantage	Simulates human-like sequential reasoning	Explores multiple options and self-corrects	Captures diverse reasoning pathways	Enables decision-making + real environment interaction

Hands_on: Exploring and Comparing Large Language Model Thinking Strategies

Overview of Agent-Building Frameworks

Reminder


Langraph
AutoGen
OpenAI Agents SDK
Crew AI
No framework
MCP

Open AI SDK: Background

- **OpenAI Python Library**: the original and most widely used SDK
- **OpenAI Agents SDK**: a newer, specialized layer on top of the Python library
- Designed for building **agentic and multi-agent systems**
- Emphasizes **flexibility over abstraction**: developers build their own patterns (loops, memory, orchestration)

Open AI SDK: Core Features

Agent Loop
Built-in loop handles tool calls and result sending.

Python-First
Use Python features to orchestrate agents.

Handoffs
Coordinate and delegate between multiple agents.

Guardrails
Run input validations in parallel to agents.

Sessions
Automatic conversation history management across runs.

Function Tools
Turn Python functions into tools with schema generation.

Open AI SDK: How to create agents

1

```
pip install openai-agents
```

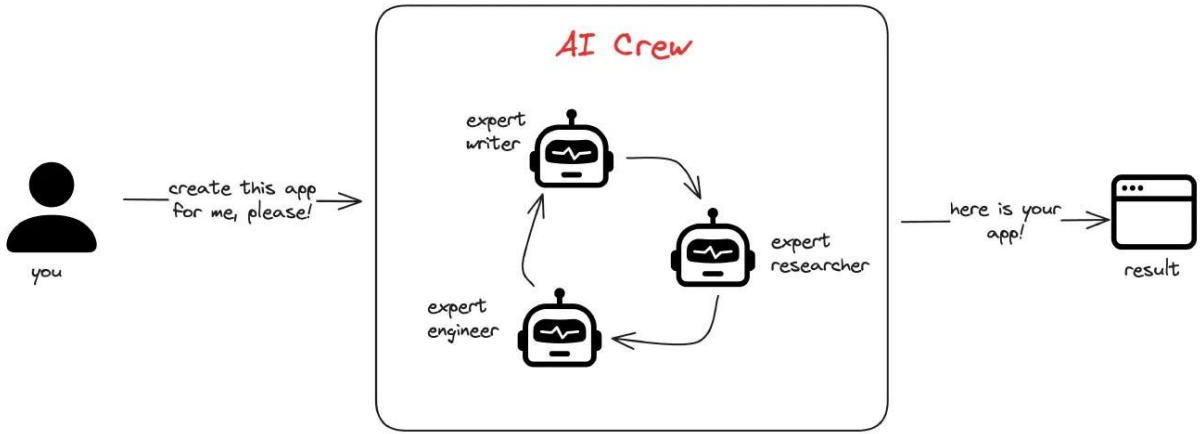
2

```python
from agents import Agent, Runner

agent = Agent(name="Assistant", instructions="You are a helpful assistant")

result = Runner.run_sync(agent, "Write a haiku about recursion in programming.")
print(result.final_output)

# Code within the code,
# Functions calling themselves,
# Infinite loop's dance.
```

Hands-on: OpenAI SDK- AI Sales Agent- Crafting Cold Emails with Distinct Personas

Crew AI: Background

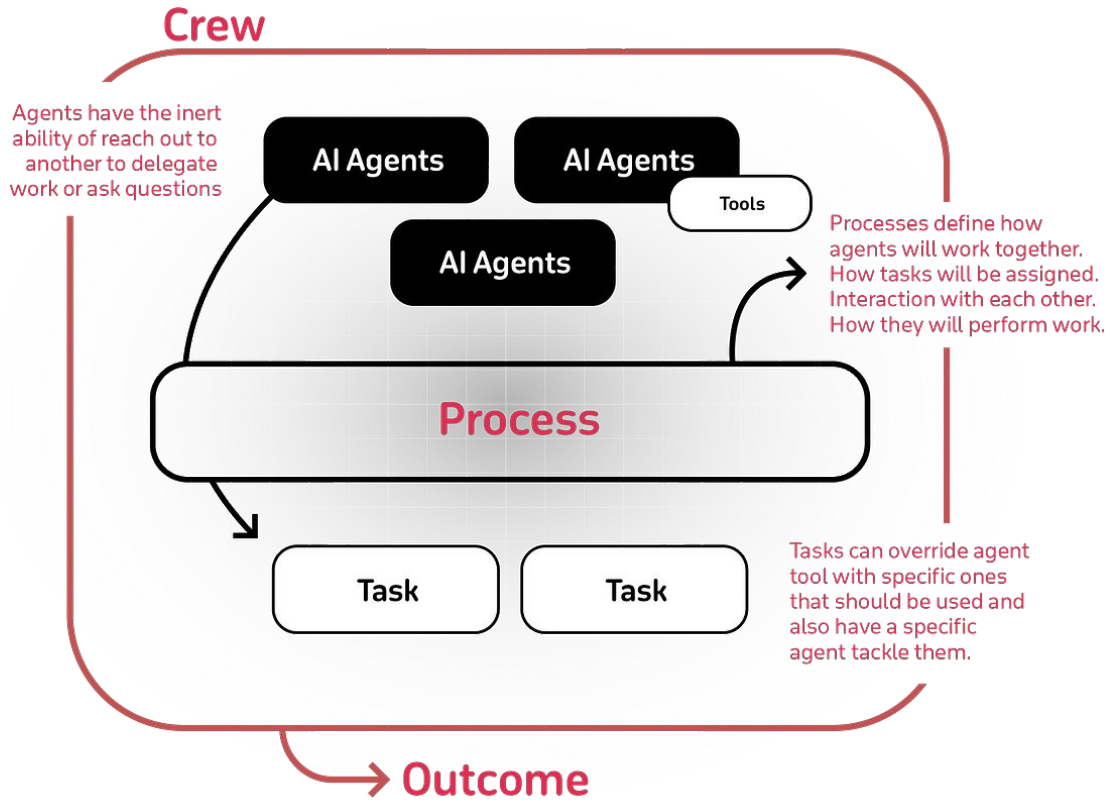
- Emerged during the AutoGPT hype to simplify **multi-agent setups**
- Agents act like a **crew of specialists**, each with defined expertise
- Enable collaboration by **role assignment and shared objectives**



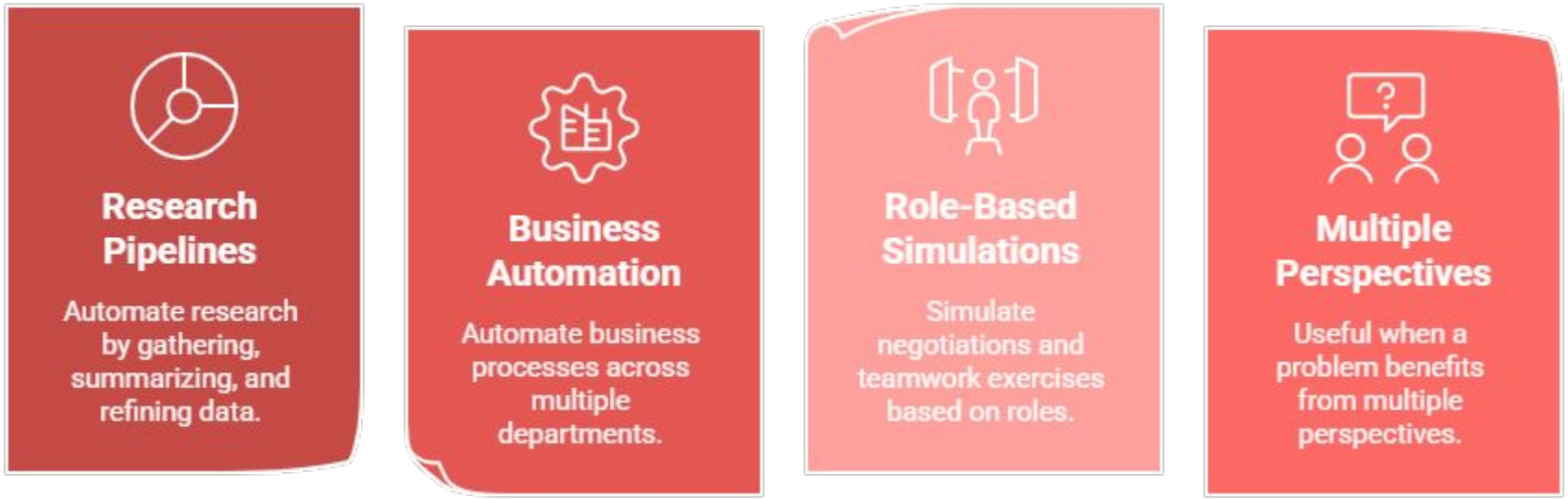
Crew AI: Components

Characteristic	Crew	AI Agents	Process	Tasks
Description	Top-level organization	Specialized team members	Workflow management system	Individual assignments
Key Features	Manages AI agent teams, oversees workflows, ensures collaboration, delivers outcomes	Specific roles, designated tools, task delegation, autonomous decisions	Defines collaboration patterns, controls task assignments, manages interactions, ensures efficient execution	Clear objectives, specific tools, feed into larger process, produce actionable results

Crew AI: Components

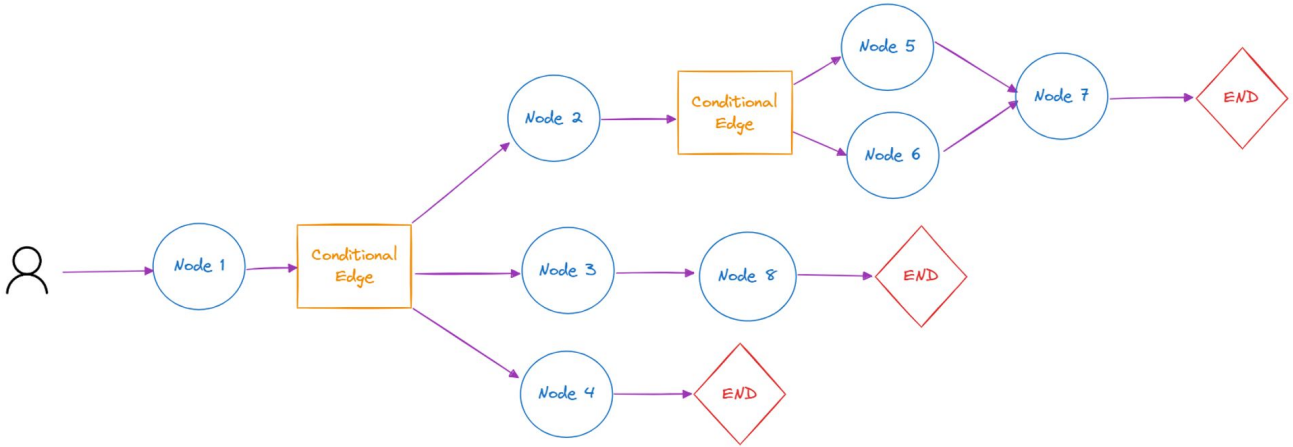


Crew AI: Use Cases



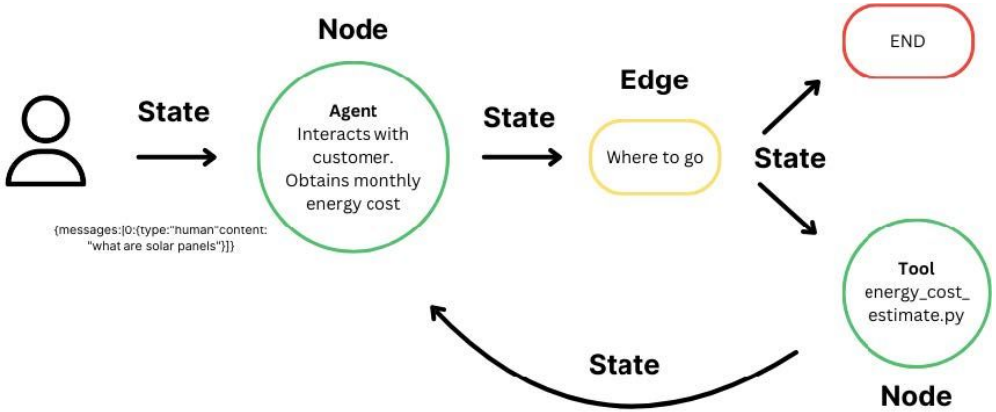
Langraph: Background

- Built as an extension of **LangChain** to solve orchestration complexity
- Inspired by **state machines** and **directed acyclic graphs (DAGs)**
- Make agents **predictable, debuggable,** and **production-ready**

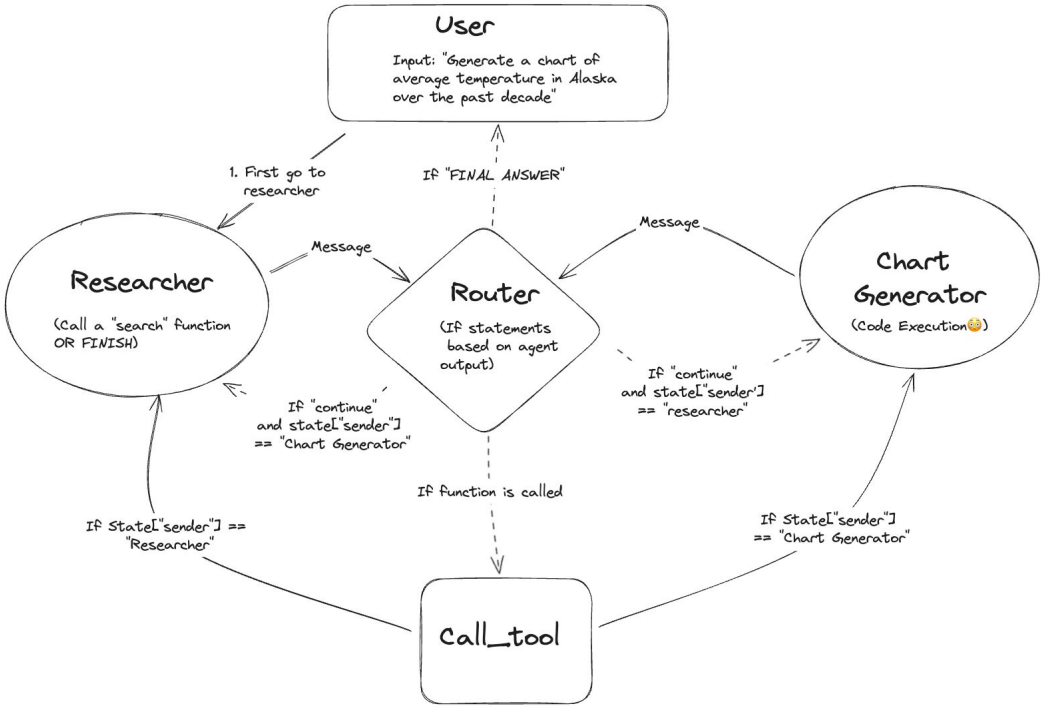


Langgraph: Core Concepts

- Workflows are modeled as **graphs of nodes**
- **State:** The shared data that flows through the workflow at each step
- **Node:** A single action or task performed within the workflow
- **Edge:** The connection that decides how execution moves from one node to another

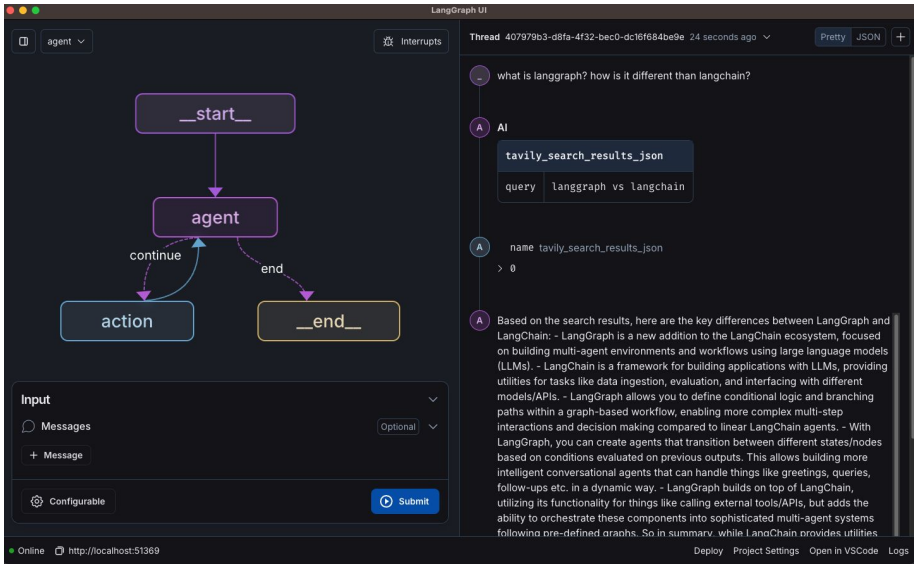


Langgraph: Use Case



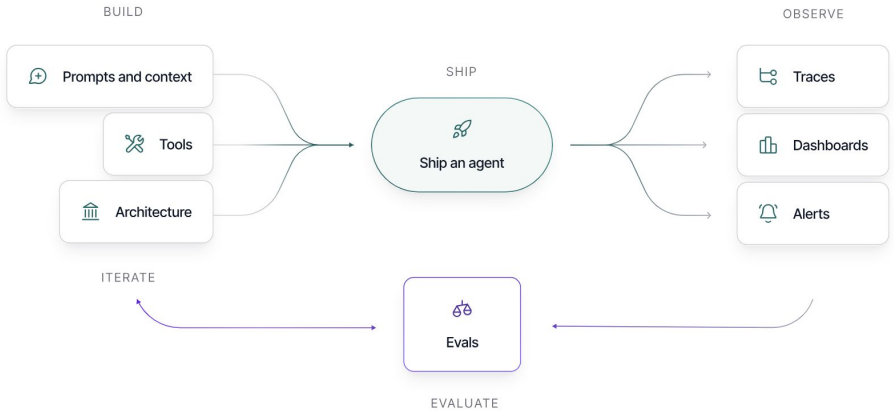
Langraph: LangGraph Studio

- A **visual desktop interface** for designing LangGraph workflows without coding
- Offers a **shallow learning curve**, letting beginners focus on workflow design
- Supports **collaboration** by sharing workflows with teams or clients
- Includes **debugging tools** to visualize and troubleshoot graphs



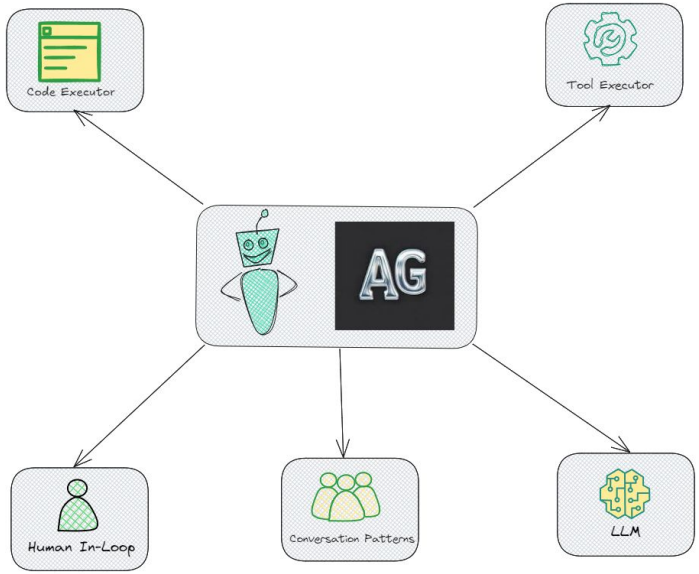
Langgaph: Langsmith

- Platform for **building production-grade LLM applications**
- Provides **end-to-end monitoring, debugging, and evaluation**
- Helps you **trace agent workflows** and understand model decisions

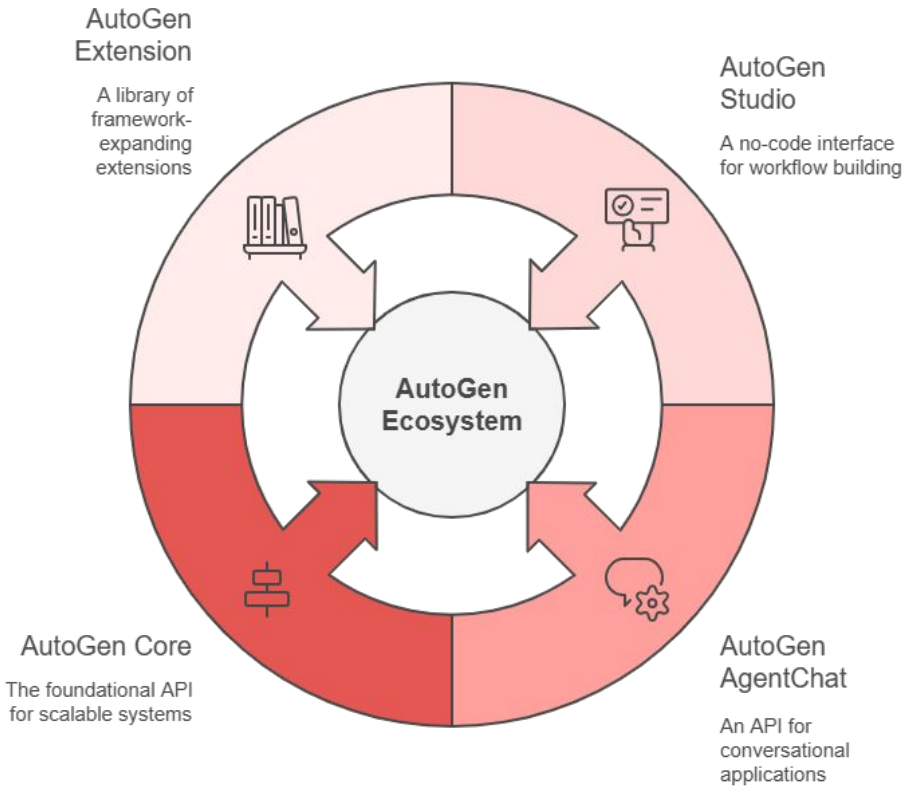


AutoGen: Background

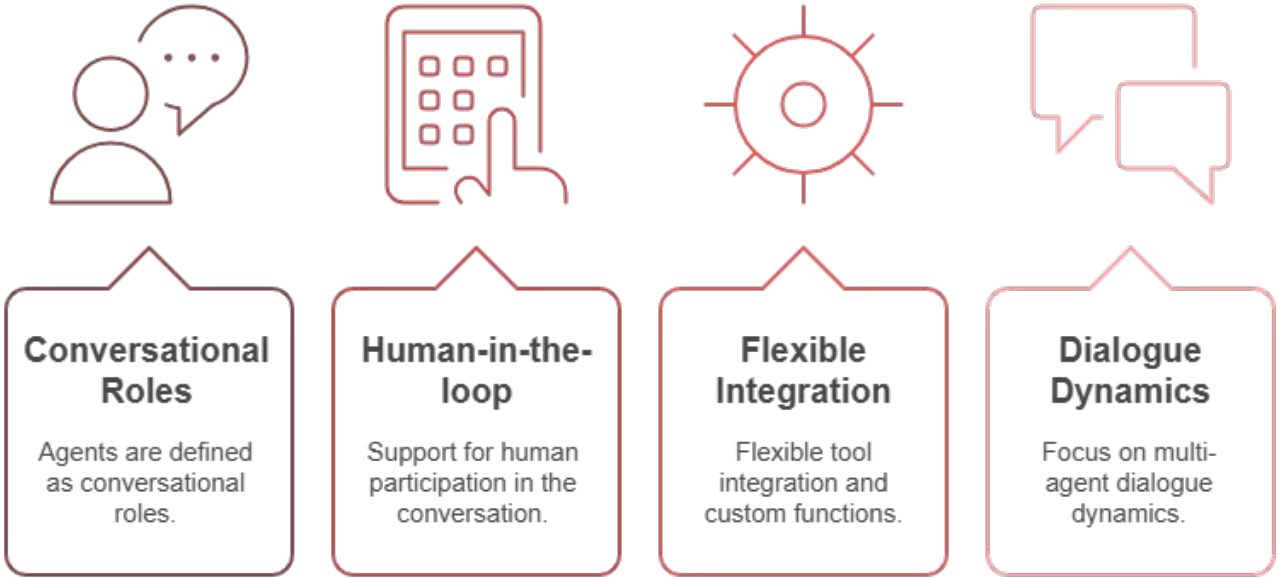
- Developed by Microsoft Research as a **multi-agent conversation framework**
- Core idea: agents collaborate through **dialogue-like interactions**
- Model problem solving as a **conversation among specialists**



Autogen: Ecosystem



AutoGen: Core Features



Comparison

Characteristic	OpenAI Agents SDK	CrewAI	LangGraph	AutoGen
Origin/Philosophy	Extension of Python Library	Role-based teamwork	Graph-based orchestration	Dialogue-driven collaboration
Agent Building	Define agents with roles, tools	Define agents in YAML/JSON	Define nodes, edges, and state	Define conversational agents
Key Features	Tool calling, agent handoffs	Multi-agent coordination, memory	Conditional routing, state persistence	Human-in-the-loop, multi-agent conversations
Best Use Cases	Enterprise bots	Workflow automation	Customer support flows	Collaborative coding

Hands-on:Langgraph

AMERICAN UNIVERSITY OF BEIRUT

Thank you