A Note on AI Agents

(this is part of the materials used in my **Python for AI Projects course** 📘)

Last update: 24/04/2025

Created by: Thu Vu

Thuvu5

Table of Content:

Disclaimer

This document is something I created to share what I've learned about AI agents — it's meant for educational and informational purposes only. While I've done my best to ensure everything is accurate and up to date, I can't guarantee that every detail is perfect or complete.

Please don't treat this as professional or technical advice. Use it as a helpful guide, but always do your own research and think critically. I'm not responsible for any outcomes that result from how you choose to use this information.

Thanks for reading, and I hope you find it useful!

Introduction

Everywhere you look, people are talking about AI agents—it's one of the hottest topics right now.

So I decided to dive into this for you—so you don't have to spend weeks piecing it all together yourself. I went through everything: classic AI books, online courses, research papers, and countless YouTube videos, all to figure out what's *actually* going on with AI agents today.

And to put it to the test, I built one myself in Python to see if this stuff really works.

Big names like Satya Nadella (CEO of Microsoft) believe AI agents will be the future of how we interact with computers. Bill Gates even says the biggest tech race right now is all about who builds the best AI agent.

There's a good chance you've already been using ChatGPT, one of the most well-known AI agents. It can generate content, browse the web, conduct research, and execute Python code.

Other examples of specialized AI agents include:

- **Writer agents** that generate blog articles *(e.g., Jasper, Copy.ai, Notion AI)*
- **Coder agents** that write and debug code *(e.g., GitHub Copilot, Cursor, Replit Ghostwriter, Devin, Windsurf, Junie)*
- **Research agents** that analyze documents and extract insights *(e.g., Humata, Elicit, Perplexity AI Pro)*
- **Customer chatbot agents** that handle support inquiries and automate responses *(e.g., Intercom Fin, Drift, Ada, Zendesk AI)*

We'll get into why AI agents are such a hot topic in a bit. But as of today, real-world AI agents still have a lot of rough edges, and complex systems often require heavy human oversight. If we do manage to make them work seamlessly, though, it could be a game-changer—both exciting *and* a little terrifying for the future of work and society.

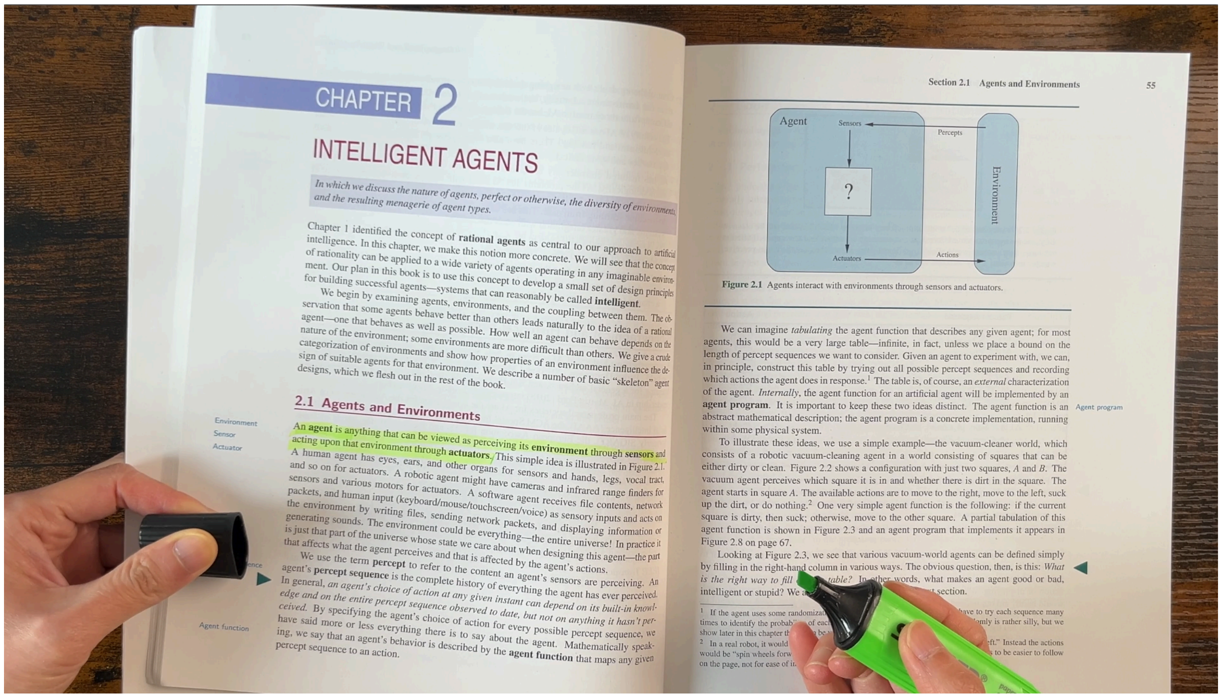
That's exactly why I want to help as many of you as possible get up to speed with the solid foundation and best insights on this fast-moving landscape.

Understanding AI Agents

AI agents are systems that perceive their environment and take actions to achieve a goal. While modern AI agents are often associated with Large Language Models (LLMs), the concept has been around for decades.

According to *Artificial Intelligence: A Modern Approach* by Stuart Russell and Peter Norvig:

> *"An agent is anything that can perceive its environment through sensors and act upon that environment through actuators."*



CHAPTER 2

INTELLIGENT AGENTS

In which we discuss the nature of agents, perfect or otherwise, the diversity of environments, and the resulting menagerie of agent types.

Chapter 1 identified the concept of **rational agents** as central to our approach to artificial intelligence. In this chapter, we make this notion more concrete. We will see that the concept of rationality can be applied to a wide variety of agents operating in any imaginable environment. Our plan in this book is to use this concept to develop a small set of design principles for building successful agents—systems that can reasonably be called **intelligent**.

We begin by examining agents, environments, and the coupling between them. The observation that some agents behave better than others leads naturally to the idea of a rational agent—one that behaves as well as possible. How well an agent can behave depends on the nature of the environment; some environments are more difficult than others. We give a crude categorization of environments and show how properties of an environment influence the design of suitable agents for that environment. We describe a number of basic "skeleton" agent designs, which we flesh out in the rest of the book.

2.1 Agents and Environments

Environment
Sensor
Actuator

An **agent** is anything that can be viewed as perceiving its **environment** through **sensors** and acting upon that environment through **actuators**. This simple idea is illustrated in Figure 2.1. A human agent has eyes, ears, and other organs for sensors and hands, legs, vocal tract, and so on for actuators. A robotic agent might have cameras and infrared range finders for sensors and various motors for actuators. A software agent receives file contents, network packets, and human input (keyboard/mouse/touchscreen/voice) as sensory inputs and acts on the environment by writing files, sending network packets, and displaying information or generating sounds. The environment could be everything—the entire universe! In practice it is just that part of the universe whose state we care about when designing this agent—the part that affects what the agent perceives and that is affected by the agent's actions.

ence

We use the term **percept** to refer to the content an agent's sensors are perceiving. An agent's **percept sequence** is the complete history of everything the agent has ever perceived. In general, *an agent's choice of action at any given instant can depend on its built-in knowledge and on the entire percept sequence observed to date, but not on anything it hasn't perceived.* By specifying the agent's choice of action for every possible percept sequence, we have said more or less everything there is to say about the agent. Mathematically speaking, we say that an agent's behavior is described by the **agent function** that maps any given percept sequence to an action.

Agent function

Section 2.1 Agents and Environments 55

Figure 2.1 Agents interact with environments through sensors and actuators.

We can imagine *tabulating* the agent function that describes any given agent; for most agents, this would be a very large table—infinite, in fact, unless we place a bound on the length of percept sequences we want to consider. Given an agent to experiment with, we can, in principle, construct this table by trying out all possible percept sequences and recording which actions the agent does in response.[1] The table is, of course, an *external* characterization of the agent. *Internally*, the agent function for an artificial agent will be implemented by an **agent program**. It is important to keep these two ideas distinct. The agent function is an abstract mathematical description; the agent program is a concrete implementation, running within some physical system.

Agent program

To illustrate these ideas, we use a simple example—the vacuum-cleaner world, which consists of a robotic vacuum-cleaning agent in a world consisting of squares that can be either dirty or clean. Figure 2.2 shows a configuration with just two squares, *A* and *B*. The vacuum agent perceives which square it is in and whether there is dirt in the square. The agent starts in square *A*. The available actions are to move to the right, move to the left, suck up the dirt, or do nothing.[2] One very simple agent function is the following: if the current square is dirty, then suck; otherwise, move to the other square. A partial tabulation of this agent function is shown in Figure 2.3 and an agent program that implements it appears in Figure 2.8 on page 67.

Looking at Figure 2.3, we see that various vacuum-world agents can be defined simply by filling in the right-hand column in various ways. The obvious question, then, is this: *What is the right way to fill* [illegible] *table?* In other words, what makes an agent good or bad, intelligent or stupid? We a [illegible] section.

[1] If the agent uses some randomizat [illegible] have to try each sequence many times to identify the probab [illegible] of eac [illegible] omly is rather silly, but we show later in this chapter th [illegible] be [illegible]

[2] In a real robot, it would [illegible] eft." Instead the actions would be "spin wheels forw [illegible] s to be easier to follow on the page, not for ease of i [illegible]

Image by author.

An **agent** can be a software system, a hardware system, a self-driving car, or even a robot. For this guide, we will focus on **software agents**, which are designed to accomplish tasks for users.

Here is a useful diagram showing what an AI agent is under the hood:

Thuvu5

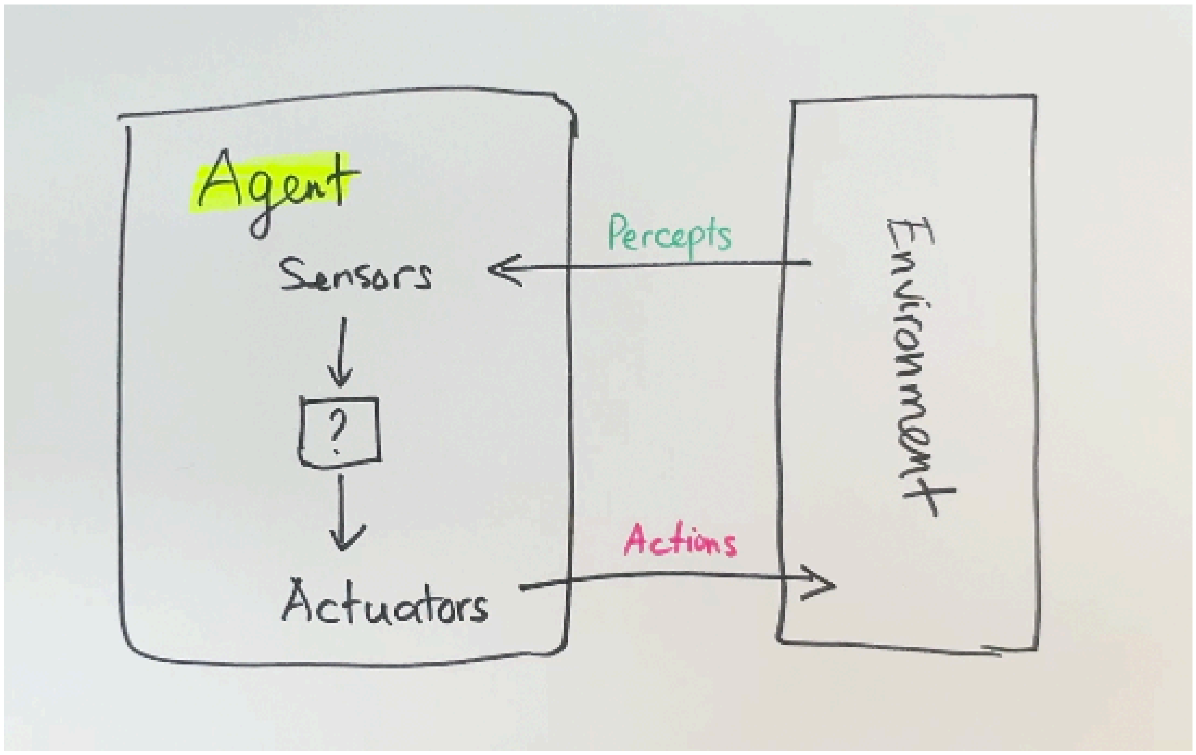


Diagram of an agent, based on the book "Artificial Intelligence: A Modern Approach (4th Edition). Image by author.

An Agent's Environment

An agent's environment is the system it interacts with and gathers information from.

- For a game agent, the game is its environment.
- A self-driving car agent's environment is the road system and its adjacent areas.
- For a robot, the environment is the physical world.
- For a travel booking AI Agent, the environment could be the travel booking system that the AI Agent uses to complete tasks.

Next, an agent gathers information and feedback from its environment through its **sensors**. For a robotic agent, this data might come from physical inputs like cameras and microphones. For a software agent, the sensor inputs could be text, voice, images,…or data obtained from an API.

Then, the agent does something with the input, and affects its environment through **actuators**. A robot's actuators might include motors for movement or grippers for handling objects. In software agents, actuators might involve actions like browsing the webs, modifying files, or executing commands in a system to accomplish a task.

Agents rely on **tools** to perform these actions. We'll talk more about tools in a bit.

Non-agentic vs. Agentic workflow vs. Fully autonomous agents

For a non-agentic workflow, you give an input to the system, and the system spits out the output, for example classifying text sentiment. This is very much like a straight line, input-output, done.

But with agentic workflow, the system can get feedback from the environment and iterate to improve the results. You can prompt the LLM multiple times, giving it opportunities to build step by step to higher-quality output.

Most people talk about agentic workflows, while there are certain agentic components, we humans still need to give feedback and help along the way for the agent to successfully complete a complex task. These are not fully autonomous agents that can figure out everything from start to end by itself.

At the time of writing, we still don't have fully autonomous agents yet, which require no human input or guidance,this is the next level and would likely need another technological breakthrough. But, if it works, it can be quite frightening as well, think about how much they can alter our lives, security concerns and job replacements.

Why AI Agents Are Trending

So if the concept of agents is nothing new, why have AI agents recently become such a buzz?

Just a few years ago, most AI systems were either narrow (doing single tasks) or required significant human orchestration.

Now, breakthroughs in AI – especially large language models – mean an AI agent can be instructed in natural language and reason about a wide variety of tasks. We no longer have to hard code the workflow, but the AI can now figure out the needed steps, what actions to take, which tools to use. So agents nwo become more **general-purpose**, and become more **autonomous.**

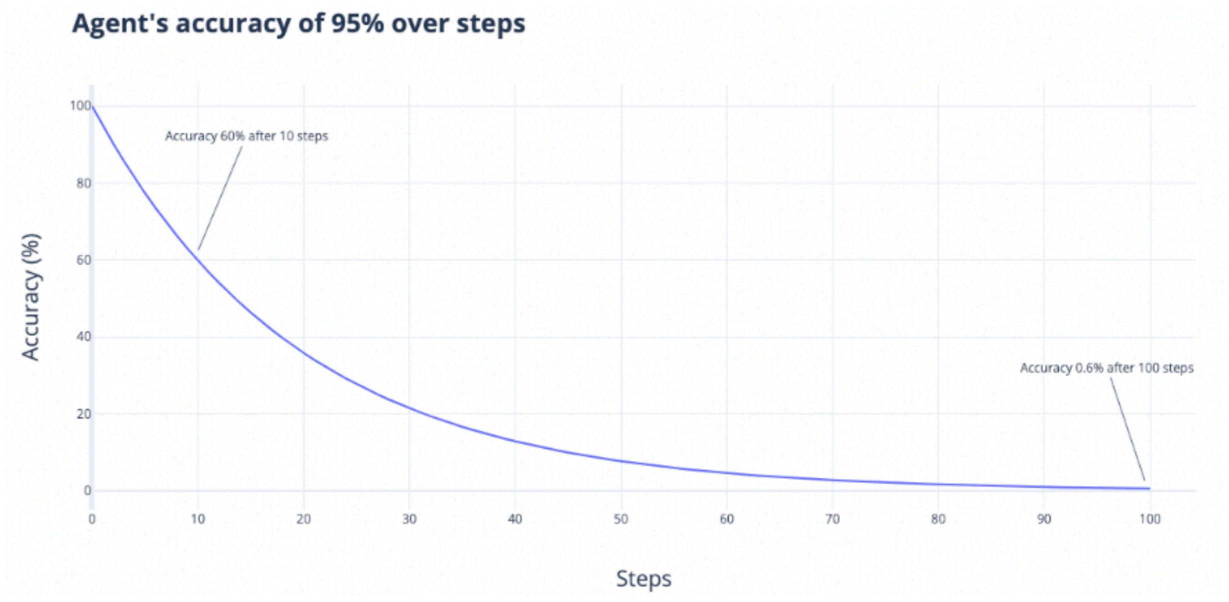
This has made the concept of “AI agents” much more accessible and useful than ever before.

An average user can now instruct an AI agent in plain English to do something (search the web, summarize a report, plan a route, etc.), and the agent can actually carry it out. Now you can **expect AI to not just answer questions, but to take actions** on your behalf, if you want it to.

Challenges of AI Agents

To successful accomplish a task, an AI agent must overcome two key challenges:

- **Compound mistakes**: an agent often needs to perform multiple steps to accomplish a task, and the overall accuracy decreases as the number of steps increases. If the model’s accuracy is 95% per step, over 10 steps, the accuracy will drop to 60%, and over 100 steps, the accuracy will be only 0.6%.



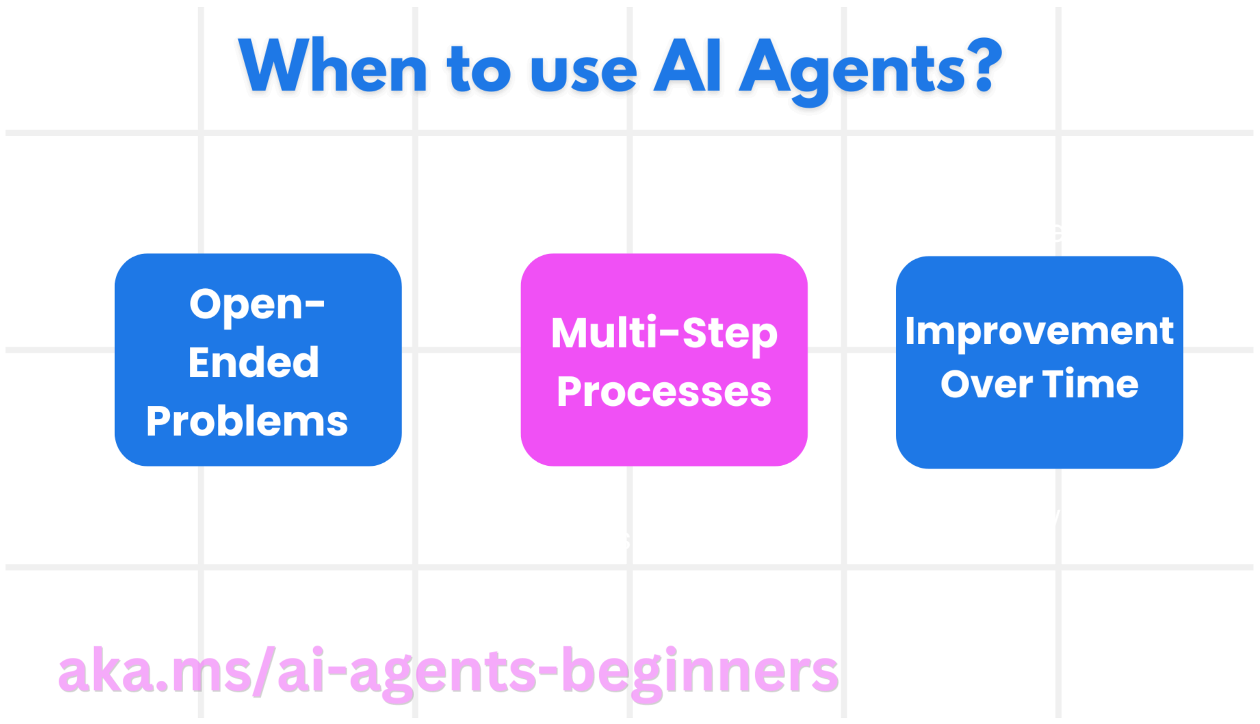
Example of diminishing accuracy. Image by author.

- **Higher stakes**: with access to tools, agents are capable of performing more impactful tasks, but any failure could have more severe consequences.

Thuvu5

That’s why more powerful models are often needed for agent use cases compared to non-agent use cases. With the current capability of AI models, in general, agents work best when they have a singular purpose, a narrow scope and a small number of tools.

Use Cases for AI Agents



AI agents are most useful in tasks involving:

- **Open-Ended Problems** - allowing the LLM to determine needed steps to complete a task because it can’t always be hardcoded into a workflow.
- **Multi-Step Processes** - tasks that require a level of complexity in which the AI Agent needs to use tools or information over multiple turns instead of single shot retrieval.
- **Improvement Over Time** - tasks where the agent can improve over time by receiving feedback from either its environment or users in order to provide better utility.

Design Patterns in AI Agent

According to Andrew Ng, AI agents follow four key design patterns:

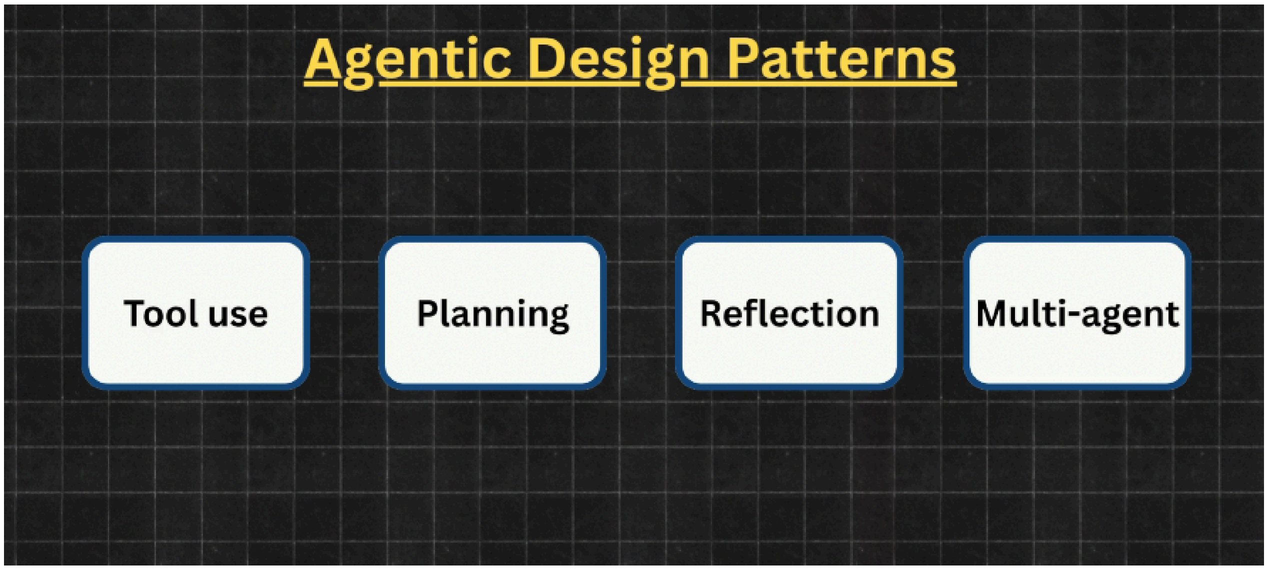


Image by author.

1. Tools

Let's start by looking into tool use, that is the ability of agents to interact with tools.

Just to be very clear: A system doesn't need access to external tools to be an agent. However, without external tools, the agent's capabilities would be limited.

By itself, a LLM model can typically perform one action— generate text. However, external tools make them vastly more capable and useful. Because tools help an agent both perceive the environment and act upon it.

Research has shown that Tool use can significantly boost a model's performance compared to just prompting or even fine tuning.

Chameleon (Lu et al., 2023) shows that a GPT-4-powered agent, augmented with a set of 13 tools including knowledge retrieval, a query generator, an image captioner, a text detector, and Bing search, can outperform GPT-4 alone with only few-shot prompting or even fine-tuning on several benchmarks.

There are many possible tools you can give your agent. They generally fall into 3 categories:

- **Knowledge augmentation tools:** An important category of tools includes those that help augment the knowledge of your agent, for example through web browsing to access up-to-date information, access to data APIs or private information.
- **Capability extension tools:** For example, AI models are notorious for being bad at math. If you ask a model what is the compound interest of $1000 after 10 years with interest of 5%, the model will likely fail. However, an agent with access to a calculator can easily do this for you. Another powerful tool is code interpreter. This allows your AI agent to actually analyze data, build out a website or a predictive model. For ChatGPT, the image generator Dalle 3 allows it to generate not only text but also images, turning it into a multimodal model. Other simple tools that can significantly boost a model's capability include a calendar, timezone converter, unit converter (e.g., from lbs to kg), and translator that can translate to and from the languages that the model isn't good at.
- **Tools that let your agent act upon its environment:** This is a little bit of a risky domain. Think of an agent that automates the whole customer outreach workflow: researching potential customers, finding their contacts, drafting emails, sending emails, reading responses, following up, extracting orders, updating your databases with new orders, etc. The question is: do you trust the agent to do these actions? What are the security concerns?

So what do tools actually look like? For software agents, tools are simply functions. Invoking a tool is, therefore, often called ***function calling***.

2. Planning

This is when you can give an AI, an LLM a certain task that you want done– such as planning a weekend trip to Paris– and it's able to figure out the exact small steps and the necessary tools that it needs in order to accomplish these steps. So the agent is free to decide how to approach the problem, autonomously determining the sequence of steps and actions given the tools it has access to. You can think of it as a problem solving machine.

The simplest way to turn a large language model into a plan generator is with prompt engineering. you can prompt the LLM to break down this goal and come up with a plan with small manageable steps on how to tackle it.

Imagine you're planning a weekend trip to Paris. So the AI model may come up with

Step 1: The AI searches for flights, checking prices and availability.

Step 2: It finds hotels based on budget and preferences.

Step 3: It creates a sightseeing itinerary with must-visit landmarks.

Step 4: It recommends restaurants and local experiences.

The agent will also decide which tools to use in each step in the plan.

Now, it's worth noting that planning is hard.

If you ever have to plan a party with a group of friends, you probably know how hard it is - I personally hate to do this 🙈. There are many things to take into account, many constraints and preferences.

So in reality, for complex tasks, humans can be involved at any stage to aid with the planning process and mitigate risks.

- A human expert can provide a plan, validate a plan, or execute parts of a plan. For example, for complex tasks for which an agent has trouble generating the whole plan, a human expert can provide a high-level plan that the agent can expand upon.
- If a plan involves risky operations, such as updating a database or merging a code change, the system can ask for explicit human approval before executing or defer to humans to execute these operations. To make this possible, you need to clearly define the level of automation an agent can have for each action.

Can LLMs plan?

An open question is how well foundation models can plan. Many researchers believe that foundation models, at least those built on top of autoregressive language models, cannot. Meta's Chief AI Scientist Yann LeCun states unequivocably that autoregressive LLMs can't plan , and can't really reason (2023).

So whether this is really true is still debatable. However, *in general, stronger models are better at planning*.

3. Reflection and error correction

Reflection is a design pattern where you're simply asking AI to more carefully look through its own results and find ways to improve the output. Reflection is a simple, easy to implement way to improve the quality and success rate of an agent.

Reflection and error correction are two different mechanisms that go hand in hand.

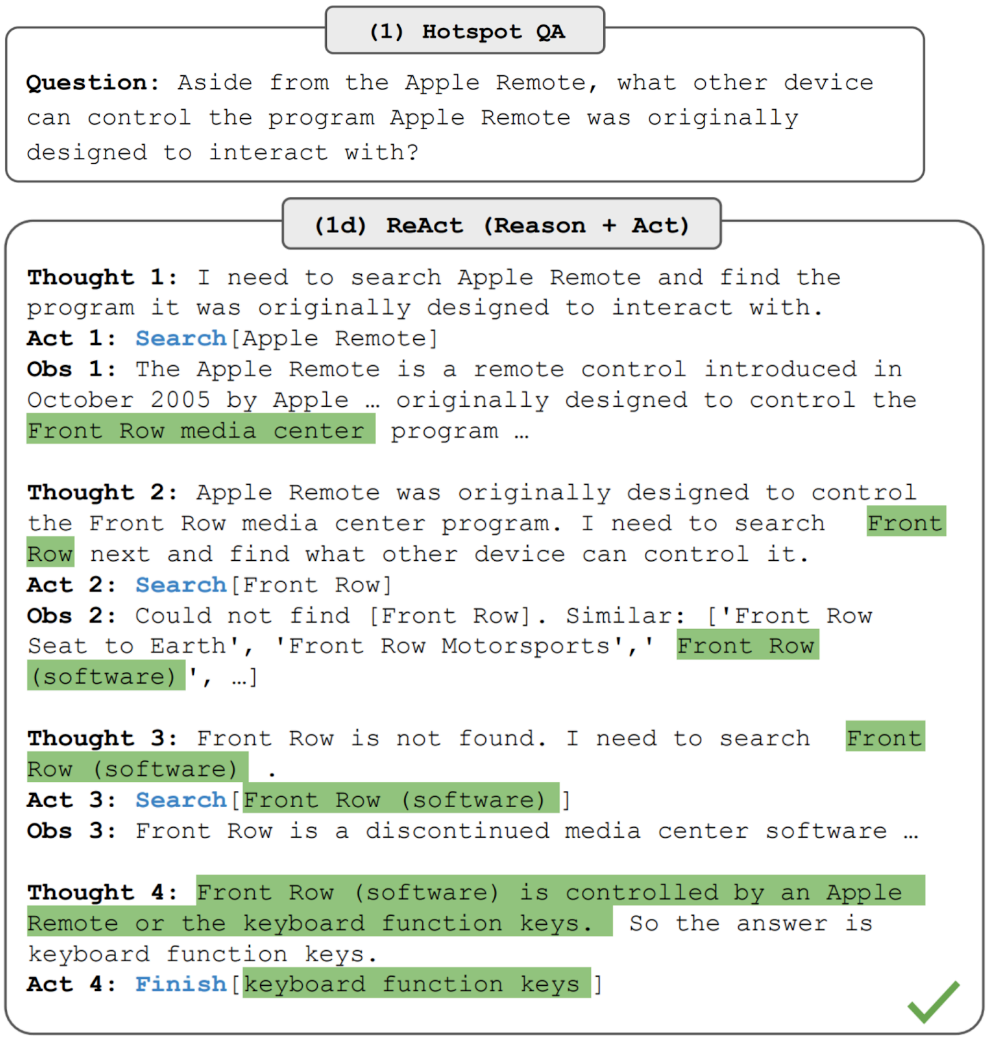
3.1 Self-Reflection Prompts:

You can add a reflection mechanism for your agent by using a self-reflection prompt:

Yao et al., 2022 proposed an approach called ReAct which stands for reason and act. You can implement this approach by prompting an agent to generate outputs in the following format:

At each step, the agent is asked to explain its thinking (), take actions, then analyze observations (reflection), until the task is considered finished by the agent.

```
Thought 1: …
Act 1: …
Observation 1: …
… [continue until reflection determines that the task is finished] …
Thought N: …
Act N: Finish [Response to query]
```



```
(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 ]
```

Source: Yao et al., 2022

3.2 Secondary agent for review:

Another approach to implement reflection is to use a multi-agent setting. We'll talk more about multi-agent collaboration in a bit.

The idea is that we'll create another agent, the critic agent, to evaluate the output generated by the coder agent. For example, the critic agent may notice that there's an error in the code output, and it will suggest how to fix it. The coder agent will take the critic, generate new code and the output until the critic age determines that the task is finished.

4. Multi-agent system

Multi-agent systems are basically just a way to get multiple AI agents to work together toward a common goal.

Think about how we humans collaborate at work. Sure, one person can handle simple tasks, but when things get complicated, we team up. One person does the research, another writes up the findings, and someone else checks everything for accuracy. Together, we create something better than any of us could alone.

🤔 *Why Go with Multi-Agents?*

Well, when we build applications with a single agent (like using an LLM to handle all the thinking and planning), things tend to get messy as the app grows. It's kind of like asking one person to do everything in a company.

You might run into problems like:

- Your agent has too many tools and gets confused about which one to use.
- There's just too much information for one agent to keep track of.
- You need different types of expertise in your system (planning, research, math skills, etc.).

The solution? Break things up! Split your application into smaller, independent agents and let them work together as a **multi-agent system**. But this raises an important question: when exactly should you use this approach?

🌟 *Advantages of Multi-agent systems*

Multi-agent systems really shine in these situations:

- **Large workloads** - When there's tons of work to do: If you've got a mountain of tasks, multiple agents can divide and conquer. Imagine you need to process thousands of customer reviews - instead of one agent slowly working through them all, you could have ten agents each handling a portion of the data.

- **Complex tasks** - When tasks get complicated: Complex jobs are usually made up of many smaller tasks. Take building a software app - you could have one agent writing code, another testing it, and a third creating documentation. Much better than one overwhelmed agent trying to juggle everything.
- **Diverse expertise** - When you need different kinds of expertise: Different agents can be specialists in different areas. For a research project, you might want one agent that's amazing at finding relevant studies, another that's great at analyzing data, and a third that excels at explaining findings in plain English.

Now that we understand when to use multi-agent systems, let's explore why they're often better than relying on a single agent, even a very sophisticated one.

🤖*Why Multiple Agents Beat a Single Super-Agent*

A single agent works fine for simple stuff, but having a team of agents offers some big advantages:

- **Specialization** - Each agent can focus on what it does best. It's like the difference between a general practitioner and a specialist doctor - sometimes you really need that focused expertise.
- **Modularity & Scalability** - Working with separate agents is like using LEGO blocks - you can build, test, and improve each piece separately. Need to scale up? Just add more agents rather than making your single agent more complicated.
- **Fault Tolerance** - With multiple agents, one agent having a bad day doesn't crash your entire system.
- **Control** - You decide exactly how your agents talk to each other, which makes it easier to understand what's happening and fix problems.

These advantages sound great in theory, but let's look at a concrete example to really understand how this works in practice.

Example: Planning a Vacation

Imagine you're building a system to help people plan vacations.

With a single agent, you'd need one super-agent that knows about flights, hotels, car rentals, local attractions, weather patterns, travel insurance... everything! That's a lot for one agent to handle well.

But with multiple agents, you could have:

- A flight expert that knows all about airlines, routes, and getting the best deals.
- A hotel guru that understands accommodation options and what makes a great place to stay.
- A local transport specialist who knows how to get around at the destination.
- An activities expert who can suggest the perfect things to do based on your interests.
- A coordinator who talks to the user and keeps everything organized.

Each agent focuses on what it does best, and together they create a better experience than one jack-of-all-trades agent could.

And to do these tasks, each agent can have access to their own tools or they can also share certain tools together. It all depends on how we design the system.

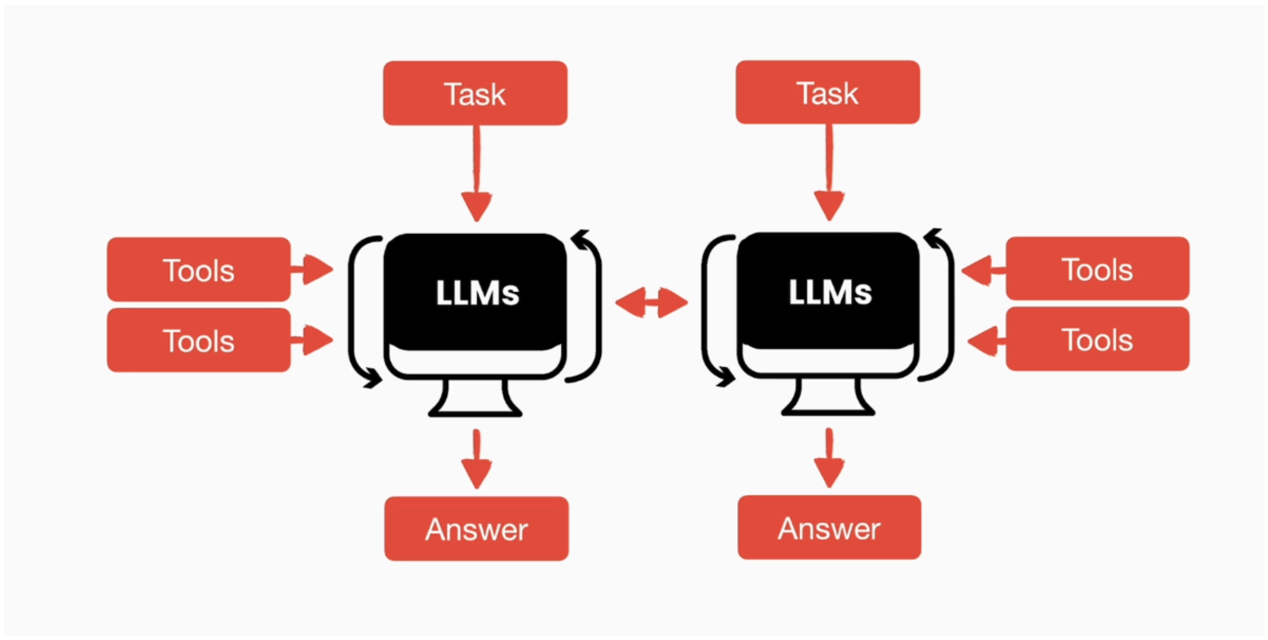


Image source: CrewAI

Now that we understand the benefits of multi-agent systems, the next question is: how do we organize these agents to work together effectively?

Different Ways to Organize Your Multi-Agent System

There are several popular ways to structure your multi-agent systems.

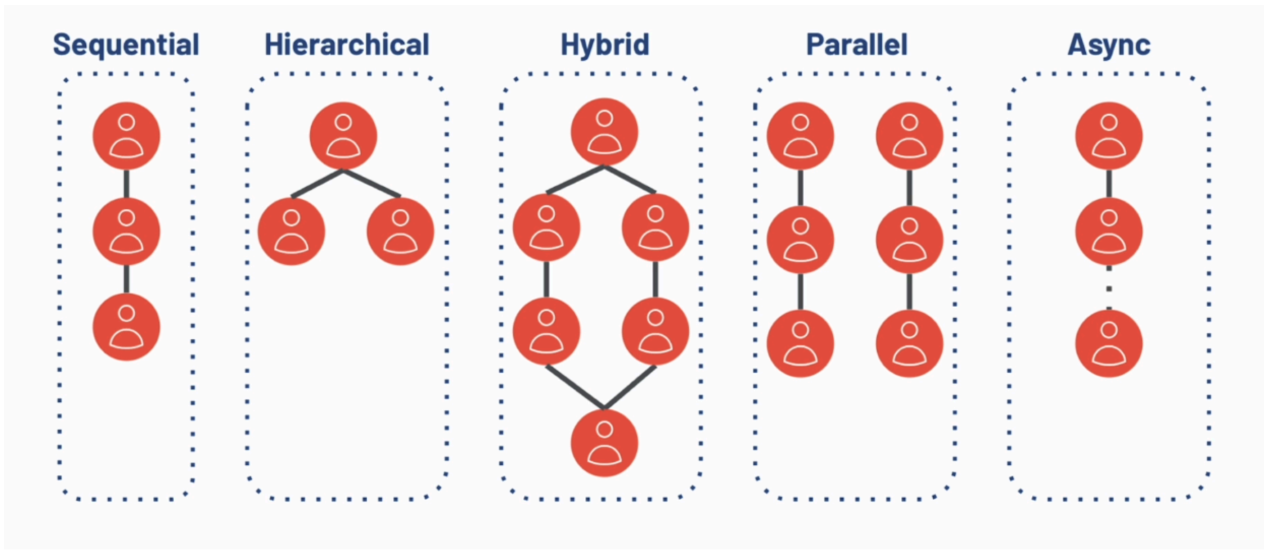
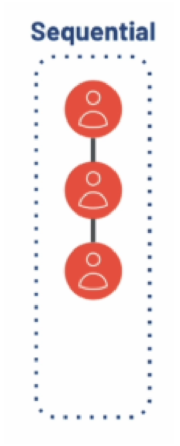


Image source: CrewAI

Let's look at each pattern and consider when you might want to use it.

1. Sequential Pattern



This is the simplest approach, agents work one after another, like an assembly line in a factory. This pattern is also called the handoffs pattern where one agent hands off control to another once it's done executing its own part.

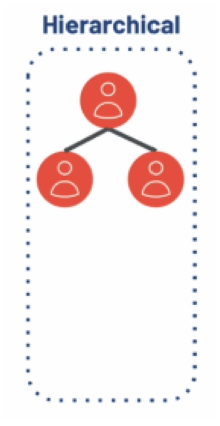
Example: Processing documents

- First agent pulls text from scanned documents
- Second agent summarizes what's important
- Third agent picks out key action items
- Fourth agent saves everything neatly in a database

Thuvu5

This works great when your task has clear steps that need to happen in a specific order. But what if your task is more complex and needs different kinds of expertise working in coordination?

2. Hierarchical Pattern



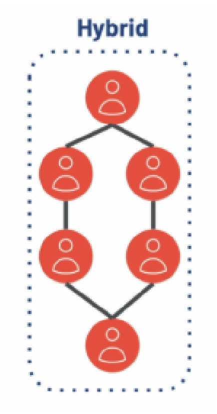
Next up is the hierarchical pattern, which is different from the sequential approach because it has a central coordinator. Here, you have a boss agent that oversees specialized worker agents. The workers do their individual tasks and report back to the boss, who puts everything together.

Example: Creating a business report

- Manager Agent: Gets the assignment and hands out tasks
- Team member 1: Digs into market trends using specialized tools
- Team member 2: Looks at what customers are saying in your feedback database
- Team member 3: Checks how products are performing across the company
- Final team member: Takes all this info and crafts it into a useful report with recommendations

This setup shines when you can break a complex task into independent pieces that need to be combined at the end. But what about situations where you need both hierarchy and continuous communication between agents?

3. Hybrid Systems



Sometimes, you need more flexibility than either the sequential or hierarchical patterns can provide. That's where hybrid systems come in. This approach mixes the assembly line and manager approaches. Agents work both top-down and in sequence, creating adaptable systems that can handle changing situations.

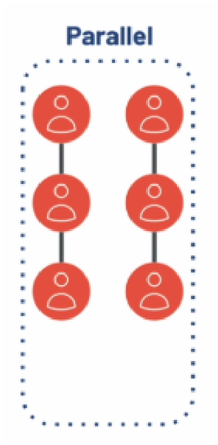
Example: Self-driving car

- Head agent: Plans the overall route and strategy
- Team members: Handle moment-to-moment tasks like watching sensors, avoiding obstacles, and checking road conditions

What makes this hybrid is the constant communication. As the car moves and traffic changes, all agents are constantly talking to each other and adjusting their actions. It's not just following a plan - it's adapting in real-time.

This approach works great for robotics, navigation, and other situations where things are constantly changing. However, sometimes the main challenge isn't complexity but sheer volume of work. That's where our next pattern comes in.

4. Parallel Systems



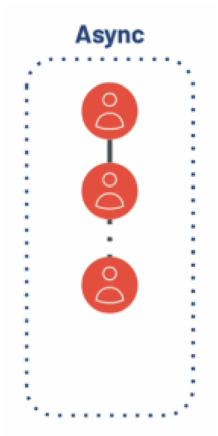
When you have a massive amount of work to do, the parallel pattern can be a lifesaver. In this setup, multiple agents tackle different parts of a big job at the same time, like a group of friends cleaning different rooms in a house.

Example: Analyzing massive datasets

- Different agents process separate chunks of data simultaneously
- Their results get combined at the end for a complete analysis

This is super helpful when you're dealing with huge amounts of information that can be split up. But what about situations where you don't know when problems will arise and need agents that can respond to events as they happen?

5. Asynchronous Systems



Our final basic pattern is the asynchronous system, which is particularly good at handling unpredictable environments. These systems have agents that work independently and respond to events as they happen, rather than following a strict order.

Example: Cybersecurity monitoring

- One agent constantly watches network traffic
- Another keeps an eye on user behavior
- A third randomly tests different security scenarios

When any agent spots something fishy, they can trigger responses from other parts of the system.

This approach is perfect for real-time monitoring and situations where you need to react to unpredictable events.

How to build an AI agent

There are really two main ways to go about it, and which one you pick depends a lot on your background and what kind of project you're dreaming up.

Option 1: Low-Code or No-Code Tools – Quick and Easy

These are perfect for folks who don't have much programming experience but still want to build something powerful. Platforms like ***N8N, Flowise, Bubble,*** and ***Replit AI*** let you design AI agents visually—think drag-and-drop components, simple configurations, and easy integration with popular tools like Slack, Google Sheets, or WhatsApp.

You can also connect directly to LLMs like GPT-4 or Claude to add intelligence to your workflows.

This approach is fast, user-friendly, and great for quickly getting something up and running. But, of course, there are a few trade-offs. Most of these platforms come with subscription fees, which you'll be paying on top of any API costs for LLMs.

And when something doesn't work quite right, it can be frustrating—you might not have full control or visibility into what's going wrong. It's also harder to customize deeply when you're relying on a pre-built system.

Option 2: Code It Yourself – More Control, More Fun, but More Complex!

Now, if you're comfortable with Python—or even just excited to learn—you can build your own AI agent from scratch.

Building it yourself means you're not tied to any platform's limitations or pricing, and you get to understand exactly how everything works under the hood. Personally, I find it a lot more fun and rewarding too.

The good news is that there are some awesome Python libraries that make this way easier than it used to be:

- **LangChain** is a go-to for chaining together LLM-powered actions.
- **CrewAI** helps set up multiple agents that can collaborate with specific roles.
- **Microsoft Autogen** is great for designing conversations between multiple agents.
- **LlamaIndex** is perfect if you want your agent to dive into large sets of documents.
- **OpenAI's Agents SDK**, which I really like because it's simple, clean, and works smoothly if you're already using OpenAI's models. This is also what I've been using for my AI agent projects.

Wrapping up

In this guide, we've covered what AI agents are, different use cases, agent design patterns and structures. This is by no means a complete document on agents. And with the speed of change nowadays, there will be massive developments to come in the coming months and years.

I hope this guide gives you some better idea of what AI agents are all about as of today.

P.S.: If you're interested in learning more about building AI projects with Python (and all the foundations in **Python/ Machine learning/ Deep learning**), feel free to check out my comprehensive course.

There have been 250+ people joining this small community 🎉, receiving direct support from me.

Otherwise, you can keep watching my free content on Youtube, where I also provide in-depth project tutorials, tips for building skills and careers in Data science and AI.