

AI Agent course

What is an Agent?

Agent is: an AI model capable of reasoning, planning, and interacting with its environment. we call it Agent because it has *agency*, aka it has the ability to interact with the environment.



An Agent is a system that leverages an AI model to interact with its environment in order to achieve a user-defined objective. It combines reasoning, planning, and the execution of actions (often via external tools) to fulfill tasks.

The most common AI model found in Agents is an LLM (Large Language Model), which takes **Text** as an input and outputs **Text** as well. Well known examples are **GPT4** from **OpenAI**, **LLama** from **Meta**, **Gemini** from **Google**, etc. These models have been trained on a vast amount of text and are able to generalize well.

It's also possible to use models that accept other inputs as the Agent's core model. a Vision Language Model (VLM), which is like an LLM but also understands images as input.

An Action, for instance, can involve the use of multiple Tools to complete. Tools enable Agents to interact with the real world and complete tasks.

an Agent is a system that uses an AI Model (typically an LLM) as its core reasoning engine, to:

- **Understand natural language:** Interpret and respond to human instructions in a meaningful way.

- **Reason and plan:** Analyze information, make decisions, and devise strategies to solve problems.
- **Interact with its environment:** Gather information, take actions, and observe the results of those actions.

LLMs enable the Agent to interpret, plan, and decide on the next steps.

An LLM is a type of AI model that excels at **understanding and generating human language**. They are trained on vast amounts of text data, allowing them to learn patterns, structure, and even nuance in language. These models typically consist of many millions of parameters.

Most LLMs nowadays are **built on the Transformer architecture**—a deep learning architecture based on the “Attention” algorithm, that has gained significant interest since the release of BERT from Google in 2018.

There are 3 types of transformers:

1. Encoders

An encoder-based Transformer takes text (or other data) as input and outputs a dense representation (or embedding) of that text.

- **Example:** BERT from Google
- **Use Cases:** Text classification, semantic search, Named Entity Recognition
- **Typical Size:** Millions of parameters

2. Decoders

A decoder-based Transformer focuses **on generating new tokens to complete a sequence, one token at a time**.

- **Example:** Llama from Meta
- **Use Cases:** Text generation, chatbots, code generation
- **Typical Size:** Billions (in the US sense, i.e., 10^9) of parameters

3. Seq2Seq (Encoder–Decoder)

A sequence-to-sequence Transformer *combines* an encoder and a decoder. The encoder first processes the input sequence into a context representation, then the decoder generates an output sequence.

- **Example:** T5, BART

- **Use Cases:** Translation, Summarization, Paraphrasing
- **Typical Size:** Millions of parameters

Although Large Language Models come in various forms, LLMs are typically decoder-based models with billions of parameters.

A “token” is the unit of information an LLM works with. You can think of a “token” as if it was a “word”, but for efficiency reasons LLMs don’t use whole words.

LLMs are said to be **autoregressive**, meaning that **the output from one pass becomes the input for the next one**. This loop continues until the model predicts the next token to be the EOS token, at which point the model can stop.

LLMs are trained on large datasets of text, where they learn to predict the next word in a sequence through a self-supervised or masked language modeling objective.

From this unsupervised learning, the model learns the structure of the language and **underlying patterns in text, allowing the model to generalize to unseen data**.

After this initial *pre-training*, LLMs can be fine-tuned on a supervised learning objective to perform specific tasks. For example, some models are trained for conversational structures or tool usage, while others focus on classification or code generation

LLM is **the brain of the Agent**.

What are AI Tools?

AI Tool is An executable process or external API that allows agents to perform specific tasks and interact with external environments

A Tool is a function given to the LLM. This function should fulfill a clear objective.

Tool	Description		
Web Search	Allows the agent to fetch up-to-date information from the internet.		
Image Generation	Creates images based on text descriptions.		

Retrieval	Retrieves information from an external source.		
API Interface	Interacts with an external API (GitHub, YouTube, Spotify, etc.).		

Those are only examples, as you can in fact create a tool for any use case!

- A Tool should contain:
 - A **textual description of what the function does.**
 - A *Callable* (something to perform an action).
 - *Arguments* with typings.
 - (Optional) Outputs with typings.

AI agents use tools as a form of “acting” in an environment , where Agents can invoke tools and use reasoning to plan and re-plan based on the information gained.

How do tools work?

LLMs, as we saw, can only receive text inputs and generate text outputs. They have no way to call tools on their own. When we talk about providing tools to an Agent, we mean teaching the LLM about the existence of these tools and instructing it to generate text-based invocations when needed.

For example, if we provide a tool to check the weather at a location from the internet and then ask the LLM about the weather in Paris, the LLM will recognize that this is an opportunity to use the “weather” tool. Instead of retrieving the weather data itself, the LLM will generate text that represents a tool call, such as `call weather_tool('Paris')`.

we have to be very precise and accurate about:

1. **What the tool does**
2. **What exact inputs it expects**

This is the reason why tool descriptions are usually provided using expressive but precise structures, such as computer languages or JSON.

Large Language Model (LLM) is A deep learning model trained on large amounts of text to understand and generate human-like language

Model Context Protocol (MCP): a unified tool interface

Model Context Protocol (MCP) is an **open protocol** that standardizes how applications **provide tools to LLMs**. MCP provides:

- A growing list of pre-built integrations that your LLM can directly plug into
- The flexibility to switch between LLM providers and vendors
- Best practices for securing your data within your infrastructure

MCP is a new open source standard to connect your agents to data sources such as databases or APIs. MCP consists of multiple components. The most important ones are the host, the client, and the server.

AI Agent Workflow, a cycle we defined as Thought-Action-Observation.

ReAct approach, a prompting technique that encourages the model to think "step by step" before acting.

Agents work in a continuous cycle of: **thinking (Thought) → acting (Act) and observing (Observe)**.

Let's break down these actions together:

1. **Thought:** The LLM part of the Agent decides what the next step should be.
2. **Action:** The agent takes an action, by calling the tools with the associated arguments.
3. **Observation:** The model reflects on the response from the tool.

The ReAct Approach

A key method is the **ReAct approach**, which is the concatenation of "Reasoning" (Think) with "Acting" (Act).

ReAct is a simple prompting technique that appends "Let's think step by step" before letting the LLM decode the next tokens. prompting the model to think "step by step" encourages the decoding process toward next tokens **that generate a plan**, rather than a final solution, since the model is encouraged to **decompose** the problem into *sub-tasks*.

We have recently seen a lot of interest for reasoning strategies. This is what's behind models like Deepseek R1 or OpenAI's o1, which have been fine-tuned to "think before answering".

These models have been trained to always include specific *thinking* sections (enclosed between `<think>` and `</think>` special tokens). This is not just a

prompting technique like ReAct, but a training method where the model learns to generate these sections after analyzing thousands of examples that show what we expect it to do.

Actions are the concrete steps an **AI agent takes to interact with its environment**.

Whether it's browsing the web for information or controlling a physical device, each action is a deliberate operation executed by the agent.

For example, an agent assisting with customer service might retrieve customer data, offer support articles, or transfer issues to a human representative.

Types of Agent Actions

There are multiple types of Agents that take actions differently:

Type of Agent	Description
JSON Agent	The Action to take is specified in JSON format.
Code Agent	The Agent writes a code block that is interpreted externally.
Function-calling Agent	It is a subcategory of the JSON Agent which has been fine-tuned to generate a new message for each action.

Actions themselves can serve many purposes:

Type of Action	Description
Information Gathering	Performing web searches, querying databases, or retrieving documents.
Tool Usage	Making API calls, running calculations, and executing code.
Environment Interaction	Manipulating digital interfaces or controlling physical devices.
Communication	Engaging with users via chat or collaborating with other agents.

One crucial part of an agent is the **ability to STOP generating new tokens when an action is complete**, and that is true for all formats of Agent: JSON, code, or function-calling. This prevents unintended output and ensures that the agent's response is clear and precise.

The LLM only handles text and uses it to describe the action it wants to take and the parameters to supply to the tool.

Instruction: Determine the most cost-effective country to purchase the smartphone model "CodeAct 1". The countries to consider are the USA, Japan, Germany, and India.

Available APIs

(1) lookup_rates(country: str) -> (float, float)

(2) convert_and_tax(price: float, exchange_rate: float, tax_rate: float) -> float

(3) estimate_final_price(converted_price: float, shipping_cost: float) -> float

(4) lookup_phone_price(model: str, country: str) -> float

(5) estimate_shipping_cost(destination_country: str) -> float

LLM Agent using [Text/JSON] as Action

Think

I should calculate the phone price in USD for each country, then find the most cost-effective country.

Action

Text: lookup_rates, Germany
JSON: {"tool": "lookup_rates", "country": "Germany"}

Environment

1.1, 0.19

Action

Text: lookup_phone_price, CodeAct 1, Germany
JSON: {"tool": "lookup_phone_price", "model": "CodeAct 1", "country": "Germany"}

Environment

700

Action

Text: convert_and_tax, 700, 1.1, 0.19
JSON: {"tool": "convert_and_tax", "price": 700, "exchange_rate": 1.1, "tax_rate": 0.19}

Environment

916.3

[... interactions omitted (look up shipping cost and calculate final price) ...]

Action

Text: lookup_rates, Japan
JSON: {"tool": "lookup_rates", "country": "Japan"}

[... interactions omitted (calculate final price for all other countries) ...]

Response

The most cost-effective country to purchase the smartphone model is Japan with price 904.00 in USD.

Fewer Actions Required!

CodeAct: LLM Agent using [Code] as Action

Think

I should calculate the phone price in USD for each country, then find the most cost-effective country.

Action

```

countries = ['USA', 'Japan', 'Germany', 'India']
final_prices = {}

for country in countries:
    exchange_rate, tax_rate = lookup_rates(country)
    local_price = lookup_phone_price("xAct 1", country)
    converted_price = convert_and_tax(
        local_price, exchange_rate, tax_rate
    )
    shipping_cost = estimate_shipping_cost(country)
    final_price = estimate_final_price(converted_price, shipping_cost)
    final_prices[country] = final_price

most_cost_effective_country = min(final_prices, key=final_prices.get)
most_cost_effective_price = final_prices[most_cost_effective_country]
print(most_cost_effective_country, most_cost_effective_price)

```

Environment

1.1, 0.19

Response

The most cost-effective country to purchase the smartphone model is Japan with price 904.00 in USD.

Control & Data Flow of Code Simplifies Complex Operations

Re-use 'min' Function from Existing Software Infrastructures (Python library)

Observations are
how an Agent perceives the consequences of its actions.

Type of Observation	Example
System Feedback	Error messages, success notifications, status codes
Data Changes	Database updates, file system modifications, state changes
Environmental Data	Sensor readings, system metrics, resource usage
Response Analysis	API responses, query results, computation outputs
Time-based Events	Deadlines reached, scheduled tasks completed