Welcome to the ■ AI Agents Course - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit0/introduction

Agents Course

Welcome to the ■ AI Agents Course

Welcome to the most exciting topic in AI today: Agents !
This free course will take you on a journey, from beginner to expert , in understanding, using and building AI agents.
This first unit will help you onboard:

bullet Discover the course's syllabus .

bullet Choose the path you're going to take (either self-audit or certification process).

bullet Get more information about the certification process and the deadlines .

bullet Get to know the team behind the course.

bullet Create your Hugging Face account .

bullet Sign-up to our Discord server , and meet your classmates and us.

Let's get started!

What to expect from this course?

In this course, you will:

bullet ■ Study AI Agents in theory, design, and practice.

bullet ■■■ Learn to use established AI Agent libraries such as smolagents , LlamaIndex , and LangGraph .

bullet ■ Share your agents on the Hugging Face Hub and explore agents created by the community.

bullet ■ Participate in challenges where you will evaluate your agents against other students'.

bullet ■ Earn a certificate of completion by completing assignments.

And more!

At the end of this course you’ll understand how Agents work and how to build your own Agents using the latest libraries and tools .
Don’t forget to sign up to the course!
(We are respectful of your privacy. We collect your email address to be able to send you the links when each Unit is published and give you information about the challenges and updates).

What does the course look like?

The course is composed of:

bullet Foundational Units : where you learn Agents concepts in theory .

bullet Hands-on : where you’ll learn to use established AI Agent libraries to train your agents in unique environments. These hands-on sections will be Hugging Face Spaces with a pre-configured environment.

bullet Use case assignments : where you’ll apply the concepts you’ve learned to solve a real-world problem that you’ll choose.

bullet The Challenge : you’ll get to put your agent to compete against other agents in a challenge. There will also be a leaderboard (not available yet) for you to compare the agents’ performance.

This course is a living project, evolving with your feedback and contributions! Feel free to open issues and PRs in GitHub , and engage in discussions in our Discord server.
After you have gone through the course, you can also send your feedback ■ using this form

What’s the syllabus?

Here is the general syllabus for the course . A more detailed list of topics will be released with each unit.
We are also planning to release some bonus units, stay tuned!

What are the prerequisites?

To be able to follow this course you should have a:

bullet Basic knowledge of Python

bullet Basic knowledge of LLMs (we have a section in Unit 1 to recap what they are)

What tools do I need?

You only need 2 things:

bullet A computer with an internet connection.

bullet A Hugging Face Account : to push and load models, agents, and create Spaces. If you don’t have an account yet, you can create one here (it’s free).

The Certification Process

You can choose to follow this course in audit mode , or do the activities and get one of the two certificates we’ll issue .
If you audit the course, you can participate in all the challenges and do assignments if you want, and you don’t need to notify us .
The certification process is completely free :

bullet To get a certification for fundamentals : you need to complete Unit 1 of the course. This is intended for students that want to get up to date with the latest trends in Agents.

bullet To get a certificate of completion : you need to complete Unit 1, one of the use case assignments we’ll propose during the course, and the final challenge.

There’s a deadline for the certification process: all the assignments must be finished before July 1st 2025 .

What is the recommended pace?

Each chapter in this course is designed to be completed in 1 week, with approximately 3-4 hours of work per week .
Since there’s a deadline, we provide you a recommended pace:

How to get the most out of the course?

To get the most out of the course, we have some advice:

bullet Join study groups in Discord : studying in groups is always easier. To do that, you need to join our discord server and verify your Hugging Face account.

bullet Do the quizzes and assignments : the best way to learn is through hands-on practice and self-assessment.

bullet Define a schedule to stay in sync : you can use our recommended pace schedule below or create yours.

Who are we

About the authors:

Acknowledgments

We would like to extend our gratitude to the following individuals for their invaluable contributions to this course:

bullet Pedro Cuenca – For his guidance and expertise in reviewing the materials.

bullet Aymeric Roucher – For his amazing demo spaces (decoding and final agent) as well as his help on the smolagents parts.

bullet Joshua Lochner – For his amazing demo space on tokenization.

bullet Quentin Gallouédec – For his help on the course content.

bullet David Berenstein – For his help on the course content and moderation.

bullet XiaXiao (ShawnSiao) – Chinese translator for the course.

bullet Jiaming Huang – Chinese translator for the course.

I found a bug, or I want to improve the course

Contributions are welcome ■

bullet If you found a bug ■ in a notebook , please open an issue and describe the problem .

bullet If you want to improve the course , you can open a Pull Request.

bullet If you want to add a full section or a new unit , the best is to open an issue and describe what content you want to add before starting to write it so that we can guide you .

I still have questions

Please ask your question in our discord server #ai-agents-discussions.
Now that you have all the information, let’s get on board ■

Joffrey Thomas

Joffrey is a machine learning engineer at Hugging Face and has built and deployed AI Agents in production. Joffrey will be your main instructor for this course.

bullet Follow Joffrey on Hugging Face

bullet Follow Joffrey on X

bullet Follow Joffrey on Linkedin

Ben Burtenshaw

Ben is a machine learning engineer at Hugging Face and has delivered multiple courses across various platforms. Ben’s goal is to make the course accessible to everyone.

bullet Follow Ben on Hugging Face

bullet Follow Ben on X

bullet Follow Ben on Linkedin

Thomas Simonini

Thomas is a machine learning engineer at Hugging Face and delivered the successful Deep RL and ML for games courses. Thomas is a big fan of Agents and is excited to see what the community will build.

bullet Follow Thomas on Hugging Face

bullet Follow Thomas on X

bullet Follow Thomas on Linkedin

Sergio Paniego

Sergio is a machine learning engineer at Hugging Face. He contributed to several sections of Units 2, 3, 4, and the bonus units.

bullet Follow Sergio on Hugging Face

bullet Follow Sergio on X

bullet Follow Sergio on Linkedin

(Optional) Discord 101 - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit0/discord101

Agents Course

(Optional) Discord 101

This guide is designed to help you get started with Discord, a free chat platform popular in the gaming and ML communities.
Join the Hugging Face Community Discord server, which has over 100,000 members , by clicking here . It's a great place to connect with others!

The Agents course on Hugging Face's Discord Community

Starting on Discord can be a bit overwhelming, so here's a quick guide to help you navigate.
The HF Community Server hosts a vibrant community with interests in various areas, offering opportunities for learning through paper discussions, events, and more.
After signing up , introduce yourself in the #introduce-yourself channel.
We created 4 channels for the Agents Course:

bullet agents-course-announcements : for the latest course informations .

bullet ■-agents-course-general : for general discussions and chitchat .

bullet agents-course-questions : to ask questions and help your classmates .

bullet agents-course-showcase : to show your best agents .

In addition you can check:

bullet smolagents : for discussion and support with the library .

Tips for using Discord effectively

How to join a server

If you are less familiar with Discord, you might want to check out this guide on how to join a server. Here’s a quick summary of the steps:

bullet Click on the Invite Link .

bullet Sign in with your Discord account, or create an account if you don’t have one.

bullet Validate that you are not an AI agent!

bullet Setup your nickname and avatar.

bullet Click “Join Server”.

How to use Discord effectively

Here are a few tips for using Discord effectively:

bullet Voice channels are available, though text chat is more commonly used.

bullet You can format text using markdown style , which is especially useful for writing code. Note that markdown doesn’t work as well for links.

bullet Consider opening threads for long conversations to keep discussions organized.

We hope you find this guide helpful! If you have any questions, feel free to ask us on Discord ■.

Onboarding: Your First Steps ■ - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit0/onboarding

Agents Course

Onboarding: Your First Steps ■

Now that you have all the details, let’s get started! We’re going to do four things:

- bullet Create your Hugging Face Account if it’s not already done
- bullet Sign up to Discord and introduce yourself (don’t be shy ■)
- bullet Follow the Hugging Face Agents Course on the Hub
- bullet Spread the word about the course

Step 1: Create Your Hugging Face Account

(If you haven’t already) create a Hugging Face account here .

Step 2: Join Our Discord Community

■■ Join our discord server here.
When you join, remember to introduce yourself in #introduce-yourself .
We have multiple AI Agent-related channels:

- bullet agents-course-announcements : for the latest course information .
- bullet ■-agents-course-general : for general discussions and chitchat .
- bullet agents-course-questions : to ask questions and help your classmates .
- bullet agents-course-showcase : to show your best agents .

In addition you can check:

- bullet smolagents : for discussion and support with the library .

If this is your first time using Discord, we wrote a Discord 101 to get the best practices. Check the next section .

Step 3: Follow the Hugging Face Agent Course Organization

Stay up to date with the latest course materials, updates, and announcements by following the Hugging Face Agents Course Organization .
■ Go here and click on follow .

Step 4: Spread the word about the course

Help us make this course more visible! There are two way you can help us:

bullet Show your support by ■ the course’s repository .

bullet Share Your Learning Journey: Let others know you’re taking this course ! We’ve prepared an illustration you can use in your social media posts

You can download the image by clicking ■ here

Step 5: Running Models Locally with Ollama (In case you run into Credit limits)

bullet Install Ollama Follow the official Instructions here.

bullet Pull a model Locally

bullet Start Ollama in the background (In one terminal)

bullet Use LiteLLMModel Instead of HfApiModel

bullet Why this works?

bullet Ollama serves models locally using an OpenAI-compatible API at http://localhost:11434 .

bullet LiteLLMModel is built to communicate with any model that supports the OpenAI chat/completion API format.

bullet This means you can simply swap out HfApiModel for LiteLLMModel no other code changes required. It’s a seamless, plug-and-play solution.

Congratulations! ■ You’ve completed the onboarding process ! You’re now ready to start learning about AI Agents. Have fun!
Keep Learning, stay awesome ■

Welcome to the ■ AI Agents Course - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit0/introduction

Agents Course

Welcome to the ■ AI Agents Course

Welcome to the most exciting topic in AI today: Agents !
This free course will take you on a journey, from beginner to expert , in understanding, using and building AI agents.
This first unit will help you onboard:

bullet Discover the course's syllabus .

bullet Choose the path you're going to take (either self-audit or certification process).

bullet Get more information about the certification process and the deadlines .

bullet Get to know the team behind the course.

bullet Create your Hugging Face account .

bullet Sign-up to our Discord server , and meet your classmates and us.

Let's get started!

What to expect from this course?

In this course, you will:

bullet ■ Study AI Agents in theory, design, and practice.

bullet ■■■ Learn to use established AI Agent libraries such as smolagents , LlamaIndex , and LangGraph .

bullet ■ Share your agents on the Hugging Face Hub and explore agents created by the community.

bullet ■ Participate in challenges where you will evaluate your agents against other students'.

bullet ■ Earn a certificate of completion by completing assignments.

And more!

At the end of this course you’ll understand how Agents work and how to build your own Agents using the latest libraries and tools .
Don’t forget to sign up to the course!
(We are respectful of your privacy. We collect your email address to be able to send you the links when each Unit is published and give you information about the challenges and updates).

What does the course look like?

The course is composed of:

bullet Foundational Units : where you learn Agents concepts in theory .

bullet Hands-on : where you’ll learn to use established AI Agent libraries to train your agents in unique environments. These hands-on sections will be Hugging Face Spaces with a pre-configured environment.

bullet Use case assignments : where you’ll apply the concepts you’ve learned to solve a real-world problem that you’ll choose.

bullet The Challenge : you’ll get to put your agent to compete against other agents in a challenge. There will also be a leaderboard (not available yet) for you to compare the agents’ performance.

This course is a living project, evolving with your feedback and contributions! Feel free to open issues and PRs in GitHub , and engage in discussions in our Discord server.
After you have gone through the course, you can also send your feedback ■ using this form

What’s the syllabus?

Here is the general syllabus for the course . A more detailed list of topics will be released with each unit.
We are also planning to release some bonus units, stay tuned!

What are the prerequisites?

To be able to follow this course you should have a:

bullet Basic knowledge of Python

bullet Basic knowledge of LLMs (we have a section in Unit 1 to recap what they are)

What tools do I need?

You only need 2 things:

bullet A computer with an internet connection.

bullet A Hugging Face Account : to push and load models, agents, and create Spaces. If you don’t have an account yet, you can create one here (it’s free).

The Certification Process

You can choose to follow this course in audit mode , or do the activities and get one of the two certificates we’ll issue .
If you audit the course, you can participate in all the challenges and do assignments if you want, and you don’t need to notify us .
The certification process is completely free :

bullet To get a certification for fundamentals : you need to complete Unit 1 of the course. This is intended for students that want to get up to date with the latest trends in Agents.

bullet To get a certificate of completion : you need to complete Unit 1, one of the use case assignments we’ll propose during the course, and the final challenge.

There’s a deadline for the certification process: all the assignments must be finished before July 1st 2025 .

What is the recommended pace?

Each chapter in this course is designed to be completed in 1 week, with approximately 3-4 hours of work per week .
Since there’s a deadline, we provide you a recommended pace:

How to get the most out of the course?

To get the most out of the course, we have some advice:

bullet Join study groups in Discord : studying in groups is always easier. To do that, you need to join our discord server and verify your Hugging Face account.

bullet Do the quizzes and assignments : the best way to learn is through hands-on practice and self-assessment.

bullet Define a schedule to stay in sync : you can use our recommended pace schedule below or create yours.

Who are we

About the authors:

Acknowledgments

We would like to extend our gratitude to the following individuals for their invaluable contributions to this course:

bullet Pedro Cuenca – For his guidance and expertise in reviewing the materials.

bullet Aymeric Roucher – For his amazing demo spaces (decoding and final agent) as well as his help on the smolagents parts.

bullet Joshua Lochner – For his amazing demo space on tokenization.

bullet Quentin Gallouédec – For his help on the course content.

bullet David Berenstein – For his help on the course content and moderation.

bullet XiaXiao (ShawnSiao) – Chinese translator for the course.

bullet Jiaming Huang – Chinese translator for the course.

I found a bug, or I want to improve the course

Contributions are welcome ■

bullet If you found a bug ■ in a notebook , please open an issue and describe the problem .

bullet If you want to improve the course , you can open a Pull Request.

bullet If you want to add a full section or a new unit , the best is to open an issue and describe what content you want to add before starting to write it so that we can guide you .

I still have questions

Please ask your question in our discord server #ai-agents-discussions.
Now that you have all the information, let’s get on board ■

Joffrey Thomas

Joffrey is a machine learning engineer at Hugging Face and has built and deployed AI Agents in production. Joffrey will be your main instructor for this course.

bullet Follow Joffrey on Hugging Face

bullet Follow Joffrey on X

bullet Follow Joffrey on Linkedin

Ben Burtenshaw

Ben is a machine learning engineer at Hugging Face and has delivered multiple courses across various platforms. Ben’s goal is to make the course accessible to everyone.

bullet Follow Ben on Hugging Face

bullet Follow Ben on X

bullet Follow Ben on Linkedin

Thomas Simonini

Thomas is a machine learning engineer at Hugging Face and delivered the successful Deep RL and ML for games courses. Thomas is a big fan of Agents and is excited to see what the community will build.

bullet Follow Thomas on Hugging Face

bullet Follow Thomas on X

bullet Follow Thomas on Linkedin

Sergio Paniego

Sergio is a machine learning engineer at Hugging Face. He contributed to several sections of Units 2, 3, 4, and the bonus units.

bullet Follow Sergio on Hugging Face

bullet Follow Sergio on X

bullet Follow Sergio on Linkedin

Live 1: How the Course Works and First Q&A; - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/communication/live1

Agents Course

Live 1: How the Course Works and First Q&A;

In this first live stream of the Agents Course, we explained how the course works (scope, units, challenges, and more) and answered your questions.
To know when the next live session is scheduled, check our Discord server . We will also send you an email. If you can’t participate, don’t worry, we record all live sessions .

Introduction to Agents - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/introduction

Agents Course

Introduction to Agents

Welcome to this first unit, where you'll build a solid foundation in the fundamentals of AI Agents including:

bullet Understanding Agents What is an Agent, and how does it work? How do Agents make decisions using reasoning and planning?

bullet What is an Agent, and how does it work?

bullet How do Agents make decisions using reasoning and planning?

bullet The Role of LLMs (Large Language Models) in Agents How LLMs serve as the "brain" behind an Agent. How LLMs structure conversations via the Messages system.

bullet How LLMs serve as the "brain" behind an Agent.

bullet How LLMs structure conversations via the Messages system.

bullet Tools and Actions How Agents use external tools to interact with the environment. How to build and integrate tools for your Agent.

bullet How Agents use external tools to interact with the environment.

bullet How to build and integrate tools for your Agent.

bullet The Agent Workflow: Think → Act → Observe .

bullet Think → Act → Observe .

After exploring these topics, you'll build your first Agent using smolagents !
Your Agent, named Alfred, will handle a simple task and demonstrate how to apply these concepts in practice.
You'll even learn how to publish your Agent on Hugging Face Spaces , so you can share it with friends and colleagues.
Finally, at the end of this Unit, you'll take a quiz. Pass it, and you'll earn your first course certification : the ■ Certificate of Fundamentals of Agents.
This Unit is your essential starting point , laying the groundwork for understanding Agents before you move on to more advanced topics.
It's a big unit, so take your time and don't hesitate to come back to these sections from time to time.

Ready? Let’s dive in! ■

Conclusion - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/conclusion

Agents Course

Conclusion

Congratulations on finishing this first Unit ■
You’ve just mastered the fundamentals of Agents and you’ve created your first AI Agent!
It’s normal if you still feel confused by some of these elements . Agents are a complex topic and it’s common to take a while to grasp everything.
Take time to really grasp the material before continuing. It’s important to master these elements and have a solid foundation before entering the fun part.
And if you pass the Quiz test, don’t forget to get your certificate ■ ■ here
In the next (bonus) unit, you’re going to learn to fine-tune a Agent to do function calling (aka to be able to call tools based on user prompt) .
Finally, we would love to hear what you think of the course and how we can improve it . If you have some feedback then, please ■ fill this form

Keep Learning, stay awesome ■

Unit 1 Quiz - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/final-quiz

Agents Course

Unit 1 Quiz

Well done on working through the first unit! Let’s test your understanding of the key concepts covered so far.
When you pass the quiz, proceed to the next section to claim your certificate.
Good luck!

Quiz

Here is the interactive quiz. The quiz is hosted on the Hugging Face Hub in a space. It will take you through a set of multiple choice questions to test your understanding of the key concepts covered in this unit. Once you’ve completed the quiz, you’ll be able to see your score and a breakdown of the correct answers.
One important thing: don’t forget to click on Submit after you passed, otherwise your exam score will not be saved!
You can also access the quiz ■ here

Certificate

Now that you have successfully passed the quiz, you can get your certificate ■
When you complete the quiz, it will grant you access to a certificate of completion for this unit. You can download and share this certificate to showcase your progress in the course.
Once you receive your certificate, you can add it to your LinkedIn ■■■ or share it on X, Bluesky, etc.
We would be super proud and would love to congratulate you if you tag @huggingface ! ■

Let’s Create Our First Agent Using smolagents - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/tutorial

Agents Course

Let’s Create Our First Agent Using smolagents

In the last section, we learned how we can create Agents from scratch using Python code, and we saw just how tedious that process can be . Fortunately, many Agent libraries simplify this work by handling much of the heavy lifting for you .
In this tutorial, you’ll create your very first Agent capable of performing actions such as image generation, web search, time zone checking and much more!
You will also publish your agent on a Hugging Face Space so you can share it with friends and colleagues .
Let’s get started!

What is smolagents?

To make this Agent, we’re going to use smolagents , a library that provides a framework for developing your agents with ease .
This lightweight library is designed for simplicity, but it abstracts away much of the complexity of building an Agent, allowing you to focus on designing your agent’s behavior.
We’re going to get deeper into smolagents in the next Unit. Meanwhile, you can also check this blog post or the library’s repo in GitHub .
In short, smolagents is a library that focuses on codeAgent , a kind of agent that performs “Actions” through code blocks, and then “Observes” results by executing the code.
Here is an example of what we’ll build!
We provided our agent with an Image generation tool and asked it to generate an image of a cat.
The agent inside smolagents is going to have the same behaviors as the custom one we built previously : it’s going to think, act and observe in cycle until it reaches a final answer:
Exciting, right?

Let’s build our Agent!

To start, duplicate this Space: https://huggingface.co/spaces/agents-course/First_agent_template
Duplicating this space means creating a local copy on your own profile :
After duplicating the Space, you’ll need to add your Hugging Face API token so your agent can access the model API:

bullet First, get your Hugging Face token from https://hf.co/settings/tokens with permission for inference, if you don’t already have one

bullet Go to your duplicated Space and click on the Settings tab

bullet Scroll down to the Variables and Secrets section and click New Secret

bullet Create a secret with the name HF_TOKEN and paste your token as the value

bullet Click Save to store your token securely

Throughout this lesson, the only file you will need to modify is the (currently incomplete) “app.py” . You can see here the original one in the template . To find yours, go to your copy of the space, then click the Files tab and then on app.py in the directory listing.
Let’s break down the code together:

bullet The file begins with some simple but necessary library imports

As outlined earlier, we will directly use the CodeAgent class from smolagents .

The Tools

Now let’s get into the tools! If you want a refresher about tools, don’t hesitate to go back to the Tools section of the course.
The Tools are what we are encouraging you to build in this section! We give you two examples:

bullet A non-working dummy Tool that you can modify to make something useful.

bullet An actually working Tool that gets the current time somewhere in the world.

To define your tool it is important to:

bullet Provide input and output types for your function, like in get_current_time_in_timezone(timezone: str) -> str:

bullet A well formatted docstring . smolagents is expecting all the arguments to have a textual description in the docstring .

The Agent

It uses Qwen/Qwen2.5-Coder-32B-Instruct as the LLM engine. This is a very capable model that we’ll access via the serverless API.
This Agent still uses the InferenceClient we saw in an earlier section behind the HfApiModel class!
We will give more in-depth examples when we present the framework in Unit 2. For now, you need to focus on adding new tools to the list of tools using the tools parameter of your Agent.
For example, you could use the DuckDuckGoSearchTool that was imported in the first line of the code, or you can examine the image_generation_tool that is loaded from the Hub later in the code.

Adding tools will give your agent new capabilities , try to be creative here!

The System Prompt

The agent’s system prompt is stored in a seperate prompts.yaml file. This file contains predefined instructions that guide the agent’s behavior.

Storing prompts in a YAML file allows for easy customization and reuse across different agents or use cases.

You can check the Space’s file structure to see where the prompts.yaml file is located and how it’s organized within the project.

The complete “app.py”:

Your Goal is to get familiar with the Space and the Agent.

Currently, the agent in the template does not use any tools, so try to provide it with some of the pre-made ones or even make some new tools yourself!

We are eagerly waiting for your amazing agents output in the discord channel #agents-course-showcase !

Congratulations, you’ve built your first Agent! Don’t hesitate to share it with your friends and colleagues.

Since this is your first try, it’s perfectly normal if it’s a little buggy or slow. In future units, we’ll learn how to build even better Agents.

The best way to learn is to try, so don’t hesitate to update it, add more tools, try with another model, etc.

In the next section, you’re going to fill the final Quiz and get your certificate!

Dummy Agent Library - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/dummy-agent-library

Agents Course

Dummy Agent Library

This course is framework-agnostic because we want to focus on the concepts of AI agents and avoid getting bogged down in the specifics of a particular framework .
Also, we want students to be able to use the concepts they learn in this course in their own projects, using any framework they like.
Therefore, for this Unit 1, we will use a dummy agent library and a simple serverless API to access our LLM engine.
You probably wouldn't use these in production, but they will serve as a good starting point for understanding how agents work .
After this section, you'll be ready to create a simple Agent using smolagents
And in the following Units we will also use other AI Agent libraries like LangGraph , LangChain , and LlamaIndex .
To keep things simple we will use a simple Python function as a Tool and Agent.
We will use built-in Python packages like datetime and os so that you can try it out in any environment.
You can follow the process in this notebook and run the code yourself .

Serverless API

In the Hugging Face ecosystem, there is a convenient feature called Serverless API that allows you to easily run inference on many models. There's no installation or deployment required.
output:
As seen in the LLM section, if we just do decoding, the model will only stop when it predicts an EOS token , and this does not happen here because this is a conversational (chat) model and we didn't apply the chat template it expects .
If we now add the special tokens related to the Llama-3.2-3B-Instruct model that we're using, the behavior changes and it now produces the expected EOS.
output:
Using the "chat" method is a much more convenient and reliable way to apply chat templates:
output:

The chat method is the RECOMMENDED method to use in order to ensure a smooth transition between models, but since this notebook is only educational, we will keep using the “text_generation” method to understand the details.

Dummy Agent

In the previous sections, we saw that the core of an agent library is to append information in the system prompt.
This system prompt is a bit more complex than the one we saw earlier, but it already contains:

bullet Information about the tools

bullet Cycle instructions (Thought → Action → Observation)

Since we are running the “text_generation” method, we need to apply the prompt manually:
We can also do it like this, which is what happens inside the chat method :
The prompt now is :
Let’s decode!
output:
Do you see the issue?
output:
Much Better! Let’s now create a dummy get weather function. In a real situation, you would likely call an API.
output:
Let’s concatenate the base prompt, the completion until function execution and the result of the function as an Observation and resume generation.
Here is the new prompt:
Output:
We learned how we can create Agents from scratch using Python code, and we saw just how tedious that process can be . Fortunately, many Agent libraries simplify this work by handling much of the heavy lifting for you.
Now, we’re ready to create our first real Agent using the smolagents library.

Observe: Integrating Feedback to Reflect and Adapt - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/observations

Agents Course

Observe: Integrating Feedback to Reflect and Adapt

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

They provide crucial information that fuels the Agent’s thought process and guides future actions.

They are signals from the environment —whether it’s data from an API, error messages, or system logs—that guide the next cycle of thought.

In the observation phase, the agent:

bullet Collects Feedback: Receives data or confirmation that its action was successful (or not).

bullet Appends Results: Integrates the new information into its existing context, effectively updating its memory.

bullet Adapts its Strategy: Uses this updated context to refine subsequent thoughts and actions.

For example, if a weather API returns the data “partly cloudy, 15°C, 60% humidity” , this observation is appended to the agent’s memory (at the end of the prompt).

The Agent then uses it to decide whether additional information is needed or if it’s ready to provide a final answer.

This iterative incorporation of feedback ensures the agent remains dynamically aligned with its goals , constantly learning and adjusting based on real-world outcomes.

These observations can take many forms , from reading webpage text to monitoring a robot arm’s position. This can be seen like Tool “logs” that provide textual feedback of the Action execution.

How Are the Results Appended?

After performing an action, the framework follows these steps in order:

bullet Parse the action to identify the function(s) to call and the argument(s) to use.

bullet Execute the action.

bullet Append the result as an Observation .

We’ve now learned the Agent’s Thought-Action-Observation Cycle.

If some aspects still seem a bit blurry, don’t worry—we’ll revisit and deepen these concepts in future Units.

Now, it’s time to put your knowledge into practice by coding your very first Agent!

Actions: Enabling the Agent to Engage with Its Environment - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/actions

Agents Course

Actions: Enabling the Agent to Engage with Its Environment

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:
Actions themselves can serve many purposes:
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.

The Stop and Parse Approach

One key method for implementing actions is the stop and parse approach . This method ensures that the agent’s output is structured and predictable:

bullet Generation in a Structured Format :

The agent outputs its intended action in a clear, predetermined format (JSON or code).

bullet Halting Further Generation :

Once the action is complete, the agent stops generating additional tokens . This prevents extra or erroneous output.

bullet Parsing the Output :

An external parser reads the formatted action, determines which Tool to call, and extracts the required parameters.
For example, an agent needing to check the weather might output:
The framework can then easily parse the name of the function to call and the arguments to apply.
This clear, machine-readable format minimizes errors and enables external tools to accurately process the agent's command.
Note: Function-calling agents operate similarly by structuring each action so that a designated function is invoked with the correct arguments. We'll dive deeper into those types of Agents in a future Unit.

Code Agents

An alternative approach is using Code Agents . The idea is: instead of outputting a simple JSON object , a Code Agent generates an executable code block—typically in a high-level language like Python .
This approach offers several advantages:

bullet Expressiveness: Code can naturally represent complex logic, including loops, conditionals, and nested functions, providing greater flexibility than JSON.

bullet Modularity and Reusability: Generated code can include functions and modules that are reusable across different actions or tasks.

bullet Enhanced Debuggability: With a well-defined programming syntax, code errors are often easier to detect and correct.

bullet Direct Integration: Code Agents can integrate directly with external libraries and APIs, enabling more complex operations such as data processing or real-time decision making.

For example, a Code Agent tasked with fetching the weather might generate the following Python snippet:
In this example, the Code Agent:

bullet Retrieves weather data via an API call ,

bullet Processes the response,

bullet And uses the print() function to output a final answer.

This method also follows the stop and parse approach by clearly delimiting the code block and signaling when execution is complete (here, by printing the final_answer).
We learned that Actions bridge an agent's internal reasoning and its real-world interactions by executing clear, structured tasks—whether through JSON, code, or function calls.
This deliberate execution ensures that each action is precise and ready for external processing via the stop and parse approach. In the next section, we will explore Observations to see how agents capture and integrate feedback from their environment.
After this, we will finally be ready to build our first Agent!

Thought: Internal Reasoning and the ReAct Approach - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/thoughts

Agents Course

Thought: Internal Reasoning and the ReAct Approach

Thoughts represent the Agent's internal reasoning and planning processes to solve the task.
This utilises the agent's Large Language Model (LLM) capacity to analyze information when presented in its prompt .
Think of it as the agent's internal dialogue, where it considers the task at hand and strategizes its approach.
The Agent's thoughts are responsible for accessing current observations and decide what the next action(s) should be.
Through this process, the agent can break down complex problems into smaller, more manageable steps , reflect on past experiences, and continuously adjust its plans based on new information.
Here are some examples of common thoughts:

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.
Indeed, 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 .
This allows the model to consider sub-steps in more detail, which in general leads to less errors than trying to generate the final solution directly.
Now that we better understand the Thought process, let's go deeper on the second part of the process: Act.

Understanding AI Agents through the Thought-Action-Observation Cycle - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/agent-steps-and-structure

Agents Course

Understanding AI Agents through the Thought-Action-Observation Cycle

In the previous sections, we learned:

- bullet How tools are made available to the agent in the system prompt .
- bullet How AI agents are systems that can ‘reason’, plan, and interact with their environment .

In this section, we’ll explore the complete AI Agent Workflow , a cycle we defined as Thought-Action-Observation.
And then, we’ll dive deeper on each of these steps.

The Core Components

Agents work in a continuous cycle of: thinking (Thought) → acting (Act) and observing (Observe) .
Let’s break down these actions together:

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

The Thought-Action-Observation Cycle

The three components work together in a continuous loop. To use an analogy from programming, the agent uses a while loop : the loop continues until the objective of the agent has been fulfilled.
Visually, it looks like this:

In many Agent frameworks, the rules and guidelines are embedded directly into the system prompt , ensuring that every cycle adheres to a defined logic.
In a simplified version, our system prompt may look like this:
We see here that in the System Message we defined :

bullet The Agent’s behavior .

bullet The Tools our Agent has access to , as we described in the previous section.

bullet The Thought-Action-Observation Cycle , that we bake into the LLM instructions.

Let’s take a small example to understand the process before going deeper into each step of the process.

Alfred, the weather Agent

We created Alfred, the Weather Agent.
A user asks Alfred: “What’s the current weather in New York?”
Alfred’s job is to answer this query using a weather API tool.
Here’s how the cycle unfolds:

Thought

Internal Reasoning:
Upon receiving the query, Alfred’s internal dialogue might be:
“The user needs current weather information for New York. I have access to a tool that fetches weather data. First, I need to call the weather API to get up-to-date details.”
This step shows the agent breaking the problem into steps: first, gathering the necessary data.

Action

Tool Usage:
Based on its reasoning and the fact that Alfred knows about a get_weather tool, Alfred prepares a JSON-formatted command that calls the weather API tool. For example, its first action could be:
Thought: I need to check the current weather for New York.
Here, the action clearly specifies which tool to call (e.g., get_weather) and what parameter to pass (the “location”: “New York”).

Observation

Feedback from the Environment:
After the tool call, Alfred receives an observation. This might be the raw weather data from the API such as:
“Current weather in New York: partly cloudy, 15°C, 60% humidity.”
This observation is then added to the prompt as additional context. It functions as real-world feedback, confirming whether the action succeeded and providing the needed details.

Updated thought

Reflecting:
With the observation in hand, Alfred updates its internal reasoning:
“Now that I have the weather data for New York, I can compile an answer for the user.”

Final Action

Alfred then generates a final response formatted as we told it to:
Thought: I have the weather data now. The current weather in New York is partly cloudy with a temperature of 15°C and 60% humidity.”
Final answer : The current weather in New York is partly cloudy with a temperature of 15°C and 60% humidity.
This final action sends the answer back to the user, closing the loop.
What we see in this example:

bullet Agents iterate through a loop until the objective is fulfilled:

Alfred’s process is cyclical . It starts with a thought, then acts by calling a tool, and finally observes the outcome. If the observation had indicated an error or incomplete data, Alfred could have re-entered the cycle to correct its approach.

bullet Tool Integration:

The ability to call a tool (like a weather API) enables Alfred to go beyond static knowledge and retrieve real-time data , an essential aspect of many AI Agents.

bullet Dynamic Adaptation:

Each cycle allows the agent to incorporate fresh information (observations) into its reasoning (thought), ensuring that the final answer is well-informed and accurate.
This example showcases the core concept behind the ReAct cycle (a concept we’re going to develop in the next section): the interplay of Thought, Action, and Observation empowers AI agents to solve complex tasks iteratively .
By understanding and applying these principles, you can design agents that not only reason about their tasks but also effectively utilize external tools to complete them , all while continuously refining their output based on environmental feedback.
Let’s now dive deeper into the Thought, Action, Observation as the individual steps of the process.

Quick Self-Check (ungraded) - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/quiz2

Agents Course

Quick Self-Check (ungraded)

What?! Another Quiz? We know, we know, … ■ But this short, ungraded quiz is here to help you reinforce key concepts you’ve just learned .
This quiz covers Large Language Models (LLMs), message systems, and tools; essential components for understanding and building AI agents.

Q1: Which of the following best describes an AI tool?

Q2: How do AI agents use tools as a form of “acting” in an environment?

Q3: What is a Large Language Model (LLM)?

Q4: Which of the following best describes the role of special tokens in LLMs?

Q5: How do AI chat models process user messages internally?

Got it? Great! Now let’s dive into the complete Agent flow and start building your first AI Agent!

What are Tools? - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/tools

Agents Course

What are Tools?

One crucial aspect of AI Agents is their ability to take actions . As we saw, this happens through the use of Tools .
In this section, we'll learn what Tools are, how to design them effectively, and how to integrate them into your Agent via the System Message.
By giving your Agent the right Tools—and clearly describing how those Tools work—you can dramatically increase what your AI can accomplish. Let's dive in!

What are AI Tools?

A Tool is a function given to the LLM . This function should fulfill a clear objective .
Here are some commonly used tools in AI agents:
Those are only examples, as you can in fact create a tool for any use case!
A good tool should be something that complements the power of an LLM .
For instance, if you need to perform arithmetic, giving a calculator tool to your LLM will provide better results than relying on the native capabilities of the model.
Furthermore, LLMs predict the completion of a prompt based on their training data , which means that their internal knowledge only includes events prior to their training. Therefore, if your agent needs up-to-date data you must provide it through some tool.
For instance, if you ask an LLM directly (without a search tool) for today's weather, the LLM will potentially hallucinate random weather.

bullet 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.

bullet A textual description of what the function does .

bullet A Callable (something to perform an action).

bullet Arguments with typings.

bullet (Optional) Outputs with typings.

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').

The Agent then reads this response, identifies that a tool call is required, executes the tool on the LLM's behalf, and retrieves the actual weather data.

The Tool-calling steps are typically not shown to the user: the Agent appends them as a new message before passing the updated conversation to the LLM again. The LLM then processes this additional context and generates a natural-sounding response for the user. From the user's perspective, it appears as if the LLM directly interacted with the tool, but in reality, it was the Agent that handled the entire execution process in the background.

We'll talk a lot more about this process in future sessions.

How do we give tools to an LLM?

The complete answer may seem overwhelming, but we essentially use the system prompt to provide textual descriptions of available tools to the model:

For this to work, we have to be very precise and accurate about:

bullet What the tool does

bullet 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. It's not necessary to do it like that, any precise and coherent format would work.

If this seems too theoretical, let's understand it through a concrete example.

We will implement a simplified calculator tool that will just multiply two integers. This could be our Python implementation:

So our tool is called calculator , it multiplies two integers , and it requires the following inputs:

bullet a (int): An integer.

bullet b (int): An integer.

The output of the tool is another integer number that we can describe like this:

bullet (int): The product of a and b .

All of these details are important. Let's put them together in a text string that describes our tool for the LLM to understand.

When we pass the previous string as part of the input to the LLM, the model will recognize it as a tool, and will know what it needs to pass as inputs and what to expect from the output.

If we want to provide additional tools, we must be consistent and always use the same format. This process can be fragile, and we might accidentally overlook some details.

Is there a better way?

Auto-formatting Tool sections

Our tool was written in Python, and the implementation already provides everything we need:

bullet A descriptive name of what it does: calculator

bullet A longer description, provided by the function’s docstring comment: Multiply two integers.

bullet The inputs and their type: the function clearly expects two int s.

bullet The type of the output.

There’s a reason people use programming languages: they are expressive, concise, and precise.

We could provide the Python source code as the specification of the tool for the LLM, but the way the tool is implemented does not matter. All that matters is its name, what it does, the inputs it expects and the output it provides.

We will leverage Python’s introspection features to leverage the source code and build a tool description automatically for us. All we need is that the tool implementation uses type hints, docstrings, and sensible function names. We will write some code to extract the relevant portions from the source code.

After we are done, we’ll only need to use a Python decorator to indicate that the calculator function is a tool:

Note the @tool decorator before the function definition.

With the implementation we’ll see next, we will be able to retrieve the following text automatically from the source code via the to_string() function provided by the decorator:

As you can see, it’s the same thing we wrote manually before!

Generic Tool implementation

We create a generic Tool class that we can reuse whenever we need to use a tool.

It may seem complicated, but if we go slowly through it we can see what it does. We define a Tool class that includes:

bullet name (str): The name of the tool.

bullet description (str): A brief description of what the tool does.

bullet function (callable): The function the tool executes.

bullet arguments (list): The expected input parameters.

bullet outputs (str or list): The expected outputs of the tool.

bullet __call__() : Calls the function when the tool instance is invoked.

bullet to_string() : Converts the tool’s attributes into a textual representation.

We could create a Tool with this class using code like the following:

But we can also use Python’s inspect module to retrieve all the information for us! This is what the @tool decorator does.

Just to reiterate, with this decorator in place we can implement our tool like this:

And we can use the Tool ’s to_string method to automatically retrieve a text suitable to be used as a tool description for an LLM:

The description is injected in the system prompt. Taking the example with which we started this section, here is how it would look like after replacing the tools_description :

In the Actions section, we will learn more about how an Agent can Call this tool we just created.

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:

bullet A growing list of pre-built integrations that your LLM can directly plug into

bullet The flexibility to switch between LLM providers and vendors

bullet Best practices for securing your data within your infrastructure

This means that any framework implementing MCP can leverage tools defined within the protocol , eliminating the need to reimplement the same tool interface for each framework.
Tools play a crucial role in enhancing the capabilities of AI agents.
To summarize, we learned:

bullet What Tools Are : Functions that give LLMs extra capabilities, such as performing calculations or accessing external data.

bullet How to Define a Tool : By providing a clear textual description, inputs, outputs, and a callable function.

bullet Why Tools Are Essential : They enable Agents to overcome the limitations of static model training, handle real-time tasks, and perform specialized actions.

Now, we can move on to the Agent Workflow where you’ll see how an Agent observes, thinks, and acts. This brings together everything we’ve covered so far and sets the stage for creating your own fully functional AI Agent.
But first, it’s time for another short quiz!

Messages and Special Tokens - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/messages-and-special-tokens

Agents Course

Messages and Special Tokens

Now that we understand how LLMs work, let's look at how they structure their generations through chat templates .
Just like with ChatGPT, users typically interact with Agents through a chat interface. Therefore, we aim to understand how LLMs manage chats.
Up until now, we've discussed prompts as the sequence of tokens fed into the model. But when you chat with systems like ChatGPT or HuggingChat, you're actually exchanging messages . Behind the scenes, these messages are concatenated and formatted into a prompt that the model can understand .
This is where chat templates come in. They act as the bridge between conversational messages (user and assistant turns) and the specific formatting requirements of your chosen LLM. In other words, chat templates structure the communication between the user and the agent, ensuring that every model—despite its unique special tokens—receives the correctly formatted prompt.
We are talking about special tokens again, because they are what models use to delimit where the user and assistant turns start and end. Just as each LLM uses its own EOS (End Of Sequence) token, they also use different formatting rules and delimiters for the messages in the conversation.

Messages: The Underlying System of LLMs

Chat-Templates

As mentioned, chat templates are essential for structuring conversations between language models and users . They guide how message exchanges are formatted into a single prompt.

System Messages

System messages (also called System Prompts) define how the model should behave . They serve as persistent instructions , guiding every subsequent interaction.
For example:
With this System Message, Alfred becomes polite and helpful:
But if we change it to:
Alfred will act as a rebel Agent ■:
When using Agents, the System Message also gives information about the available tools, provides instructions to the model on how to format the actions to take, and includes guidelines on how the thought process should be segmented.

Conversations: User and Assistant Messages

A conversation consists of alternating messages between a Human (user) and an LLM (assistant).
Chat templates help maintain context by preserving conversation history, storing previous exchanges between the user and the assistant. This leads to more coherent multi-turn conversations.
For example:
In this example, the user initially wrote that they needed help with their order. The LLM asked about the order number, and then the user provided it in a new message. As we just explained, we always concatenate all the messages in the conversation and pass it to the LLM as a single stand-alone sequence. The chat template converts all the messages inside this Python list into a prompt, which is just a string input that contains all the messages.
For example, this is how the SmolLM2 chat template would format the previous exchange into a prompt:
However, the same conversation would be translated into the following prompt when using Llama 3.2:
Templates can handle complex multi-turn conversations while maintaining context:

Base Models vs. Instruct Models

Another point we need to understand is the difference between a Base Model vs. an Instruct Model:

bullet A Base Model is trained on raw text data to predict the next token.

bullet An Instruct Model is fine-tuned specifically to follow instructions and engage in conversations. For example, SmolLM2-135M is a base model, while SmolLM2-135M-Instruct is its instruction-tuned variant.

To make a Base Model behave like an instruct model, we need to format our prompts in a consistent way that the model can understand . This is where chat templates come in.
ChatML is one such template format that structures conversations with clear role indicators (system, user, assistant). If you have interacted with some AI API lately, you know that’s the standard practice.
It’s important to note that a base model could be fine-tuned on different chat templates, so when we’re using an instruct model we need to make sure we’re using the correct chat template.

Understanding Chat Templates

Because each instruct model uses different conversation formats and special tokens, chat templates are implemented to ensure that we correctly format the prompt the way each model expects.

In transformers , chat templates include Jinja2 code that describes how to transform the ChatML list of JSON messages, as presented in the above examples, into a textual representation of the system-level instructions, user messages and assistant responses that the model can understand.
This structure helps maintain consistency across interactions and ensures the model responds appropriately to different types of inputs .
Below is a simplified version of the SmolLM2-135M-Instruct chat template:
As you can see, a chat_template describes how the list of messages will be formatted.
Given these messages:
The previous chat template will produce the following string:
The transformers library will take care of chat templates for you as part of the tokenization process. Read more about how transformers uses chat templates here . All we have to do is structure our messages in the correct way and the tokenizer will take care of the rest.
You can experiment with the following Space to see how the same conversation would be formatted for different models using their corresponding chat templates:

Messages to prompt

The easiest way to ensure your LLM receives a conversation correctly formatted is to use the chat_template from the model's tokenizer.
To convert the previous conversation into a prompt, we load the tokenizer and call apply_chat_template :
The rendered_prompt returned by this function is now ready to use as the input for the model you chose!
Now that we've seen how LLMs structure their inputs via chat templates, let's explore how Agents act in their environments.
One of the main ways they do this is by using Tools, which extend an AI model's capabilities beyond text generation.
We'll discuss messages again in upcoming units, but if you want a deeper dive now, check out:

bullet Hugging Face Chat Templating Guide

bullet Transformers Documentation

What are LLMs? - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/what-are-llms

Agents Course

What are LLMs?

In the previous section we learned that each Agent needs an AI Model at its core , and that LLMs are the most common type of AI models for this purpose.
Now we will learn what LLMs are and how they power Agents.
This section offers a concise technical explanation of the use of LLMs. If you want to dive deeper, you can check our free Natural Language Processing Course .

What is a Large Language Model?

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:

bullet 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

bullet Example : BERT from Google

bullet Use Cases : Text classification, semantic search, Named Entity Recognition

bullet Typical Size : Millions of parameters

bullet 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

bullet Example : Llama from Meta

bullet Use Cases : Text generation, chatbots, code generation

bullet Typical Size : Billions (in the US sense, i.e., 10^9) of parameters

bullet 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

bullet Example : T5, BART

bullet Use Cases : Translation, Summarization, Paraphrasing

bullet Typical Size : Millions of parameters

Although Large Language Models come in various forms, LLMs are typically decoder-based models with billions of parameters. Here are some of the most well-known LLMs:

The underlying principle of an LLM is simple yet highly effective: its objective is to predict the next token, given a sequence of previous tokens . 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. For example, while English has an estimated 600,000 words, an LLM might have a vocabulary of around 32,000 tokens (as is the case with Llama 2). Tokenization often works on sub-word units that can be combined.

For instance, consider how the tokens “interest” and “ing” can be combined to form “interesting”, or “ed” can be appended to form “interested.”

You can experiment with different tokenizers in the interactive playground below:

Each LLM has some special tokens specific to the model. The LLM uses these tokens to open and close the structured components of its generation. For example, to indicate the start or end of a sequence, message, or response. Moreover, the input prompts that we pass to the model are also structured with special tokens. The most important of those is the End of sequence token (EOS).

The forms of special tokens are highly diverse across model providers.

The table below illustrates the diversity of special tokens.

Understanding next token prediction.

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.

In other words, an LLM will decode text until it reaches the EOS. But what happens during a single decoding loop?

While the full process can be quite technical for the purpose of learning agents, here’s a brief overview:

bullet Once the input text is tokenized , the model computes a representation of the sequence that captures information about the meaning and the position of each token in the input sequence.

bullet This representation goes into the model, which outputs scores that rank the likelihood of each token in its vocabulary as being the next one in the sequence.

Based on these scores, we have multiple strategies to select the tokens to complete the sentence.

bullet The easiest decoding strategy would be to always take the token with the maximum score.

You can interact with the decoding process yourself with SmolLM2 in this Space (remember, it decodes until reaching an EOS token which is <|im_end|> for this model):

bullet But there are more advanced decoding strategies. For example, beam search explores multiple candidate sequences to find the one with the maximum total score–even if some individual tokens have lower scores.

If you want to know more about decoding, you can take a look at the NLP course .

Attention is all you need

A key aspect of the Transformer architecture is Attention . When predicting the next word, not every word in a sentence is equally important; words like “France” and “capital” in the sentence “The capital of France is …” carry the most meaning.
Although the basic principle of LLMs—predicting the next token—has remained consistent since GPT-2, there have been significant advancements in scaling neural networks and making the attention mechanism work for longer and longer sequences.
If you’ve interacted with LLMs, you’re probably familiar with the term context length , which refers to the maximum number of tokens the LLM can process, and the maximum attention span it has.

Prompting the LLM is important

Considering that the only job of an LLM is to predict the next token by looking at every input token, and to choose which tokens are “important”, the wording of your input sequence is very important.
The input sequence you provide an LLM is called a prompt . Careful design of the prompt makes it easier to guide the generation of the LLM toward the desired output .

How are LLMs trained?

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.

How can I use LLMs?

You have two main options:

bullet Run Locally (if you have sufficient hardware).

bullet Use a Cloud/API (e.g., via the Hugging Face Serverless Inference API).

Throughout this course, we will primarily use models via APIs on the Hugging Face Hub. Later on, we will explore how to run these models locally on your hardware.

How are LLMs used in AI Agents?

LLMs are a key component of AI Agents, providing the foundation for understanding and generating human language .
They can interpret user instructions, maintain context in conversations, define a plan and decide which tools to use.
We will explore these steps in more detail in this Unit, but for now, what you need to understand is that the LLM is the brain of the Agent .
That was a lot of information! We’ve covered the basics of what LLMs are, how they function, and their role in powering AI agents.
If you’d like to dive even deeper into the fascinating world of language models and natural language processing, don’t hesitate to check out our free NLP course .
Now that we understand how LLMs work, it’s time to see how LLMs structure their generations in a conversational context .
To run this notebook , you need a Hugging Face token that you can get from https://hf.co/settings/tokens .
For more information on how to run Jupyter Notebooks, checkout Jupyter Notebooks on the Hugging Face Hub .
You also need to request access to the Meta Llama models .

Q1: What is an Agent? - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/quiz1

Agents Course

Q1: What is an Agent?

Which of the following best describes an AI Agent?

Q2: What is the Role of Planning in an Agent?

Why does an Agent need to plan before taking an action?

Q3: How Do Tools Enhance an Agent’s Capabilities?

Why are tools essential for an Agent?

Q4: How Do Actions Differ from Tools?

What is the key difference between Actions and Tools?

Q5: What Role Do Large Language Models (LLMs) Play in Agents?

How do LLMs contribute to an Agent’s functionality?

Q6: Which of the Following Best Demonstrates an AI Agent?

Which real-world example best illustrates an AI Agent at work?
Congrats on finishing this Quiz ■! If you need to review any elements, take the time to revisit the chapter to reinforce your knowledge before diving deeper into the “Agent’s brain”: LLMs.

What is an Agent? - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/unit1/what-are-agents

Agents Course

What is an Agent?

By the end of this section, you’ll feel comfortable with the concept of agents and their various applications in AI.
To explain what an Agent is, let’s start with an analogy.

The Big Picture: Alfred The Agent

Meet Alfred. Alfred is an Agent .
Imagine Alfred receives a command , such as: “Alfred, I would like a coffee please.”
Because Alfred understands natural language , he quickly grasps our request.
Before fulfilling the order, Alfred engages in reasoning and planning , figuring out the steps and tools he needs to:

bullet Go to the kitchen

bullet Use the coffee machine

bullet Brew the coffee

bullet Bring the coffee back

Once he has a plan, he must act . To execute his plan, he can use tools from the list of tools he knows about .
In this case, to make a coffee, he uses a coffee machine. He activates the coffee machine to brew the coffee.
Finally, Alfred brings the freshly brewed coffee to us.
And this is what an 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.

Let’s go more formal

Now that you have the big picture, here’s a more precise definition:
Think of the Agent as having two main parts:

bullet The Brain (AI Model)

This is where all the thinking happens. The AI model handles reasoning and planning . It decides which Actions to take based on the situation .

bullet The Body (Capabilities and Tools)

This part represents everything the Agent is equipped to do .
The scope of possible actions depends on what the agent has been equipped with . For example, because humans lack wings, they can’t perform the “fly” Action , but they can execute Actions like “walk”, “run” ,“jump”, “grab”, and so on.

What type of AI Models do we use for Agents?

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. We will learn more about LLMs in the next section .

How does an AI take action on its environment?

LLMs are amazing models, but they can only generate text .
However, if you ask a well-known chat application like HuggingChat or ChatGPT to generate an image, they can! How is that possible?
The answer is that the developers of HuggingChat, ChatGPT and similar apps implemented additional functionality (called Tools), that the LLM can use to create images.
We will learn more about tools in the Tools section.

What type of tasks can an Agent do?

An Agent can perform any task we implement via Tools to complete Actions .
For example, if I write an Agent to act as my personal assistant (like Siri) on my computer, and I ask it to “send an email to my Manager asking to delay today’s meeting”, I can give it some code to send emails. This will be a new Tool the Agent can use whenever it needs to send an email. We can write it in Python:
The LLM, as we’ll see, will generate code to run the tool when it needs to, and thus fulfill the desired task.
The design of the Tools is very important and has a great impact on the quality of your Agent . Some tasks will require very specific Tools to be crafted, while others may be solved with general purpose tools like “web_search”.
Allowing an agent to interact with its environment allows real-life usage for companies and individuals .

The spectrum of “Agency”

Following this definition, Agents exist on a continuous spectrum of increasing agency:
Table from smolagents conceptual guide .

Example 1: Personal Virtual Assistants

Virtual assistants like Siri, Alexa, or Google Assistant, work as agents when they interact on behalf of users using their digital environments.
They take user queries, analyze context, retrieve information from databases, and provide responses or initiate actions (like setting reminders, sending messages, or controlling smart devices).

Example 2: Customer Service Chatbots

Many companies deploy chatbots as agents that interact with customers in natural language.
These agents can answer questions, guide users through troubleshooting steps, open issues in internal databases, or even complete transactions.
Their predefined objectives might include improving user satisfaction, reducing wait times, or increasing sales conversion rates. By interacting directly with customers, learning from the dialogues, and adapting their responses over time, they demonstrate the core principles of an agent in action.

Example 3: AI Non-Playable Character in a video game

AI agents powered by LLMs can make Non-Playable Characters (NPCs) more dynamic and unpredictable.
Instead of following rigid behavior trees, they can respond contextually, adapt to player interactions , and generate more nuanced dialogue. This flexibility helps create more lifelike, engaging characters that evolve alongside the player’s actions.
To summarize, an Agent is a system that uses an AI Model (typically an LLM) as its core reasoning engine, to:

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

bullet Reason and plan: Analyze information, make decisions, and devise strategies to solve problems.

bullet Interact with its environment: Gather information, take actions, and observe the results of those actions.

Now that you have a solid grasp of what Agents are, let’s reinforce your understanding with a short, ungraded quiz. After that, we’ll dive into the “Agent’s brain”: the LLMs .