

Hugging Face Agents Course

Table of Contents

Agents Course

Welcome to the ■ AI Agents Course

What to expect from this course?

What does the course look like?

What's the syllabus?

What are the prerequisites?

What tools do I need?

The Certification Process

What is the recommended pace?

How to get the most out of the course?

Who are we

Acknowledgments

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

I still have questions

Joffrey Thomas

Ben Burtenshaw

Thomas Simonini

Sergio Paniego

Agents Course

Welcome to the ■ AI Agents Course

What to expect from this course?

What does the course look like?

What's the syllabus?

What are the prerequisites?

What tools do I need?

The Certification Process

What is the recommended pace?

How to get the most out of the course?

Who are we

Acknowledgments

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

I still have questions

Joffrey Thomas

Ben Burtenshaw

Thomas Simonini

Sergio Paniego

Agents Course

Onboarding: Your First Steps ■

Step 1: Create Your Hugging Face Account

Step 2: Join Our Discord Community

Step 3: Follow the Hugging Face Agent Course

Organization

Step 4: Spread the word about the course

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

Agents Course

Onboarding: Your First Steps ■

Step 1: Create Your Hugging Face Account

Step 2: Join Our Discord Community

Step 3: Follow the Hugging Face Agent Course

Organization

Step 4: Spread the word about the course

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

Agents Course

(Optional) Discord 101

The Agents course on Hugging Face's Discord Community

Tips for using Discord effectively

How to join a server

How to use Discord effectively

Agents Course

(Optional) Discord 101

The Agents course on Hugging Face's Discord Community

Tips for using Discord effectively

How to join a server

How to use Discord effectively

Agents Course

Introduction to Agents

Agents Course

What is an Agent?

The Big Picture: Alfred The Agent

Let's go more formal

What type of AI Models do we use for Agents?

How does an AI take action on its environment?

What type of tasks can an Agent do?

The spectrum of "Agency"

Example 1: Personal Virtual Assistants

Example 2: Customer Service Chatbots

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

Agents Course

Introduction

What You'll Learn

Agents Course

Introduction

What You'll Learn

Agents Course

What is an Agent?

The Big Picture: Alfred The Agent

Let's go more formal

What type of AI Models do we use for Agents?

How does an AI take action on its environment?

What type of tasks can an Agent do?

The spectrum of "Agency"

Example 1: Personal Virtual Assistants

Example 2: Customer Service Chatbots

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

Agents Course

Introduction to Agents

Agents Course

What are LLMs?

What is a Large Language Model?

Understanding next token prediction.

Attention is all you need

Prompting the LLM is important

How are LLMs trained?

How can I use LLMs?

How are LLMs used in AI Agents?

Agents Course

What are LLMs?

What is a Large Language Model?

Understanding next token prediction.

Attention is all you need

Prompting the LLM is important

How are LLMs trained?

How can I use LLMs?

How are LLMs used in AI Agents?

Agents Course

Messages and Special Tokens

Messages: The Underlying System of LLMs

Chat-Templates

System Messages

Conversations: User and Assistant Messages

Base Models vs. Instruct Models

Understanding Chat Templates

Messages to prompt

Agents Course

Messages and Special Tokens

Messages: The Underlying System of LLMs

Chat-Templates

System Messages

Conversations: User and Assistant Messages

Base Models vs. Instruct Models

Understanding Chat Templates

Messages to prompt

Agents Course

What are Tools?

What are AI Tools?

How do tools work?

How do we give tools to an LLM?

Auto-formatting Tool sections

Generic Tool implementation

Model Context Protocol (MCP): a unified tool interface

Agents Course

What are Tools?

What are AI Tools?

How do tools work?

How do we give tools to an LLM?

- Auto-formatting Tool sections

- Generic Tool implementation

- Model Context Protocol (MCP): a unified tool interface

Agents Course

Understanding AI Agents through the Thought-Action-Observation Cycle

- The Core Components

- The Thought-Action-Observation Cycle

- Alfred, the weather Agent

 - Thought

 - Action

 - Observation

 - Updated thought

 - Final Action

Agents Course

Understanding AI Agents through the Thought-Action-Observation Cycle

- The Core Components

- The Thought-Action-Observation Cycle

- Alfred, the weather Agent

 - Thought

 - Action

 - Observation

 - Updated thought

 - Final Action

Agents Course

Thought: Internal Reasoning and the ReAct Approach

- The ReAct Approach

Agents Course

Thought: Internal Reasoning and the ReAct Approach

The ReAct Approach

Agents Course

Actions: Enabling the Agent to Engage with Its Environment

Types of Agent Actions

The Stop and Parse Approach

Code Agents

Agents Course

Actions: Enabling the Agent to Engage with Its Environment

Types of Agent Actions

The Stop and Parse Approach

Code Agents

Agents Course

Observe: Integrating Feedback to Reflect and Adapt

How Are the Results Appended?

Agents Course

Observe: Integrating Feedback to Reflect and Adapt

How Are the Results Appended?

Agents Course

Dummy Agent Library

Serverless API

Dummy Agent

Agents Course

Dummy Agent Library

Serverless API

Dummy Agent

Agents Course

Let's Create Our First Agent Using smolagents

What is smolagents?

Let's build our Agent!

The Tools

The Agent

The System Prompt

Agents Course

Let's Create Our First Agent Using smolagents

What is smolagents?

Let's build our Agent!

The Tools

The Agent

The System Prompt

Agents Course

Q1: What is an Agent?

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

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

Q4: How Do Actions Differ from Tools?

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

Agents?

Q6: Which of the Following Best Demonstrates an AI

Agent?

Agents Course

Q1: What is an Agent?

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

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

Q4: How Do Actions Differ from Tools?

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

Agents?

Q6: Which of the Following Best Demonstrates an AI

Agent?

Agents Course

Quick Self-Check (ungraded)

- 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?

Agents Course

Quick Self-Check (ungraded)

- 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?

Agents Course

Unit 1 Quiz

Quiz

Certificate

Agents Course

Unit 1 Quiz

Quiz

Certificate

Agents Course

Conclusion

Keep Learning, stay awesome ■

Agents Course

Conclusion

Keep Learning, Stay Awesome ■

Agents Course

Conclusion

Keep Learning, Stay Awesome ■

Agents Course

Conclusion

Keep Learning, stay awesome ■

Agents Course

Let's Fine-Tune Your Model for Function-Calling

How do we train our model for function-calling?

LoRA (Low-Rank Adaptation of Large Language Models)

Fine-Tuning a Model for Function-Calling

Agents Course

What is Function Calling?

How does the model "learn" to take an action?

Agents Course

Let's Fine-Tune Your Model for Function-Calling

How do we train our model for function-calling?

LoRA (Low-Rank Adaptation of Large Language Models)

Fine-Tuning a Model for Function-Calling

Agents Course

What is Function Calling?

How does the model "learn" to take an action?

Agents Course

Introduction to Agentic Frameworks

When to Use an Agentic Framework

Agentic Frameworks Units

Agents Course

AI Agent Observability & Evaluation

- When Should I Do This Bonus Unit?
- What You'll Learn
- Ready to Get Started?

Agents Course

AI Agent Observability & Evaluation

- When Should I Do This Bonus Unit?
- What You'll Learn
- Ready to Get Started?

Agents Course

Introduction to Agentic Frameworks

When to Use an Agentic Framework

Agentic Frameworks Units

Agents Course

Quiz: Evaluating AI Agents

Q1: What does observability in AI agents primarily refer to?

Q2: Which of the following is NOT a common metric monitored in agent observability?

Q3: What best describes offline evaluation of an AI agent?

Q4: Which advantage does online evaluation of agents offer?

Q5: What role does OpenTelemetry play in AI agent observability and evaluation?

Agents Course

Bonus Unit 2: Observability and Evaluation of Agents

Exercise Prerequisites ■■

Step 0: Install the Required Libraries

Step 1: Instrument Your Agent

Step 2: Test Your Instrumentation

Step 3: Observe and Evaluate a More Complex Agent

Online Evaluation

Offline Evaluation

Final Thoughts

Trace Structure

Common Metrics to Track in Production

Dataset Evaluation

1. Costs

2. Latency

3. Additional Attributes

4. User Feedback

5. LLM-as-a-Judge

6. Observability Metrics Overview

Running the Agent on the Dataset

Agents Course

AI Agent Observability and Evaluation

■ What is Observability?

■ Why Agent Observability Matters

■ Observability Tools

■ Traces and Spans

■ Key Metrics to Monitor

■ Evaluating AI Agents

■■■ Lets see how this works in practice

■ Offline Evaluation

■ Online Evaluation

■ Combining the two

Agents Course

Bonus Unit 2: Observability and Evaluation of Agents

Exercise Prerequisites ■■■

Step 0: Install the Required Libraries

Step 1: Instrument Your Agent

Step 2: Test Your Instrumentation

Step 3: Observe and Evaluate a More Complex Agent

Online Evaluation

Offline Evaluation

Final Thoughts

Trace Structure

Common Metrics to Track in Production

Dataset Evaluation

1. Costs
 2. Latency
 3. Additional Attributes
 4. User Feedback
 5. LLM-as-a-Judge
 6. Observability Metrics Overview
- Running the Agent on the Dataset

Agents Course

Quiz: Evaluating AI Agents

Q1: What does observability in AI agents primarily refer to?

Q2: Which of the following is NOT a common metric monitored in agent observability?

Q3: What best describes offline evaluation of an AI agent?

Q4: Which advantage does online evaluation of agents offer?

Q5: What role does OpenTelemetry play in AI agent observability and evaluation?

Agents Course

AI Agent Observability and Evaluation

- What is Observability?
- Why Agent Observability Matters
- Observability Tools
- Traces and Spans
- Key Metrics to Monitor
- Evaluating AI Agents
- Lets see how this works in practice
 - Offline Evaluation
 - Online Evaluation
 - Combining the two

Agents Course

Introduction

Want to go further?

Agents Course

Introduction

Want to go further?

Agents Course

Conclusion

Agents Course

Conclusion

Agents Course

Launching Your Pokémon Battle Agent

Battle the Stream Agent!

Pokémon Battle Agent Challenger

How to Launch Your Agent

Agents Course

Build Your Own Pokémon Battle Agent

■ Poke-env

■■ Pokémon Showdown

- LLMAgentBase
- TemplateAgent
 - Core Logic
 - Key Internal Methods

Agents Course

From LLMs to AI Agents

The big limitation of Agents: it's slow (for now)

Agents Course

The State of the Art in Using LLMs in Games

- Covert Protocol by NVIDIA and Inworld AI
- NEO NPCs by Ubisoft
- Mecha BREAK Featuring NVIDIA's ACE
- Suck Up! by Proxima Enterprises

Wait... Where Are the Agents?

Agents Course

Launching Your Pokémon Battle Agent

Battle the Stream Agent!

Pokémon Battle Agent Challenger

How to Launch Your Agent

Agents Course

Build Your Own Pokémon Battle Agent

- Poke-env
- Pokémon Showdown
- LLMAgentBase
- TemplateAgent
 - Core Logic
 - Key Internal Methods

Agents Course

From LLMs to AI Agents

The big limitation of Agents: it's slow (for now)

Agents Course

The State of the Art in Using LLMs in Games

■■■■■ Covert Protocol by NVIDIA and Inworld AI

■ NEO NPCs by Ubisoft

■■ Mecha BREAK Featuring NVIDIA's ACE

■■■■ Suck Up! by Proxima Enterprises

Wait... Where Are the Agents?

Agents Course

Welcome to the final Unit

What's the challenge?

Agents Course

Welcome to the final Unit

What's the challenge?

Agents Course

Conclusion

Agents Course

Conclusion

Agents Course

And now? What topics I should learn?

■ Model Context Protocol (MCP)

■ Agent-to-Agent (A2A) Protocol

Agents Course

And now? What topics I should learn?

■ Model Context Protocol (MCP)

■ Agent-to-Agent (A2A) Protocol

Agents Course

Claim Your Certificate ■

Agents Course

Hands-On

The Dataset

The process

Agents Course

What is GAIA?

■ GAIA's Core Principles

Difficulty Levels

Example of a Hard GAIA Question

Live Evaluation

Agents Course

Claim Your Certificate ■

Agents Course

Hands-On

The Dataset

The process

Agents Course

What is GAIA?

■ GAIA's Core Principles

Difficulty Levels

Example of a Hard GAIA Question

Live Evaluation

Unit 0

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

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

Source: <https://huggingface.co/learn/agents-course/en/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

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 ■

Onboarding: Your First Steps ■ - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/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 ■

(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 ■.

(Optional) Discord 101 - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/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 ■.

Unit 1

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! ■

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 .

Introduction - Hugging Face Agents Course

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

Agents Course

Introduction

Welcome to this first Bonus Unit , where you'll learn to fine-tune a Large Language Model (LLM) for function calling .

In terms of LLMs, function calling is quickly becoming a must-know technique.

The idea is, rather than relying only on prompt-based approaches like we did in Unit 1, function calling trains your model to take actions and interpret observations during the training phase , making your AI more robust.

The best way for you to be able to follow this Bonus Unit is:

- bullet Know how to Fine-Tune an LLM with Transformers, if it's not the case check this .
- bullet Know how to use SFTTrainer to fine-tune our model, to learn more about it check this documentation .

What You'll Learn

- bullet Function Calling How modern LLMs structure their conversations effectively letting them trigger Tools .
- bullet LoRA (Low-Rank Adaptation) A lightweight and efficient fine-tuning method that cuts down on computational and storage overhead. LoRA makes training large models faster, cheaper, and easier to deploy.
- bullet The Thought → Act → Observe Cycle in Function Calling models A simple but powerful approach for structuring how your model decides when (and how) to call functions, track intermediate steps, and interpret the results from external Tools or APIs.
- bullet New Special Tokens We'll introduce special markers that help the model distinguish between: Internal "chain-of-thought" reasoning Outgoing function calls Responses coming back from external tools
- bullet Internal "chain-of-thought" reasoning

- bullet Outgoing function calls
- bullet Responses coming back from external tools

By the end of this bonus unit, you'll be able to:

- bullet Understand the inner working of APIs when it comes to Tools.
- bullet Fine-tune a model using the LoRA technique.
- bullet Implement and modify the Thought → Act → Observe cycle to create robust and maintainable Function-calling workflows.
- bullet Design and utilize special tokens to seamlessly separate the model's internal reasoning from its external actions.

And you'll have fine-tuned your own model to do function calling. ■

Let's dive into function calling !

Introduction - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/bonus-unit1/introduction>

Agents Course

Introduction

Welcome to this first Bonus Unit , where you'll learn to fine-tune a Large Language Model (LLM) for function calling .

In terms of LLMs, function calling is quickly becoming a must-know technique.

The idea is, rather than relying only on prompt-based approaches like we did in Unit 1, function calling trains your model to take actions and interpret observations during the training phase , making your AI more robust.

The best way for you to be able to follow this Bonus Unit is:

- bullet Know how to Fine-Tune an LLM with Transformers, if it's not the case check this .
- bullet Know how to use SFTTrainer to fine-tune our model, to learn more about it check this documentation .

What You'll Learn

- bullet Function Calling How modern LLMs structure their conversations effectively letting them trigger Tools .
- bullet LoRA (Low-Rank Adaptation) A lightweight and efficient fine-tuning method that cuts down on computational and storage overhead. LoRA makes training large models faster, cheaper, and easier to deploy.
- bullet The Thought → Act → Observe Cycle in Function Calling models A simple but powerful approach for structuring how your model decides when (and how) to call functions, track intermediate steps, and interpret the results from external Tools or APIs.
- bullet New Special Tokens We'll introduce special markers that help the model distinguish between: Internal "chain-of-thought" reasoning Outgoing function calls Responses coming back from external tools
- bullet Internal "chain-of-thought" reasoning

- bullet Outgoing function calls
- bullet Responses coming back from external tools

By the end of this bonus unit, you'll be able to:

- bullet Understand the inner working of APIs when it comes to Tools.
- bullet Fine-tune a model using the LoRA technique.
- bullet Implement and modify the Thought → Act → Observe cycle to create robust and maintainable Function-calling workflows.
- bullet Design and utilize special tokens to seamlessly separate the model's internal reasoning from its external actions.

And you'll have fine-tuned your own model to do function calling. ■

Let's dive into function calling !

What is an Agent? - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/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 .

Introduction to Agents - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/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! ■

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 .

What are LLMs? - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/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 .

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](#)

Messages and Special Tokens - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/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 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!

What are Tools? - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/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!

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:

- 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.

- 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.

- 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.

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

Source: <https://huggingface.co/learn/agents-course/en/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:

- 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.

- 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.

- 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.

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.

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

Source: <https://huggingface.co/learn/agents-course/en/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.

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!

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

Source: <https://huggingface.co/learn/agents-course/en/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!

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!

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

Source: <https://huggingface.co/learn/agents-course/en/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!

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.

Dummy Agent Library - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/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.

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 separate 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!

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

Source: <https://huggingface.co/learn/agents-course/en/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 separate 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!

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.

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

Source: <https://huggingface.co/learn/agents-course/en/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.

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!

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

Source: <https://huggingface.co/learn/agents-course/en/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!

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 [here](#)

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 [here](#) or share it on X, Bluesky, etc.

We would be super proud and would love to congratulate you if you tag [@huggingface](#) ! [here](#)

Unit 1 Quiz - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/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 [here](#)

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 [here](#) or share it on X, Bluesky, etc.

We would be super proud and would love to congratulate you if you tag [@huggingface](#) ! [here](#)

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 ■

Conclusion - Hugging Face Agents Course

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

Agents Course

Conclusion

Congratulations on finishing this first Bonus Unit ■

You've just mastered understanding function-calling and how to fine-tune your model to do function-calling !

If we have one piece of advice now, it's to try to fine-tune different models . The best way to learn is by trying.

In the next Unit, you're going to learn how to use state-of-the-art frameworks such as smolagents , LlamaIndex and LangGraph .

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 ■

Conclusion - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/bonus-unit1/conclusion>

Agents Course

Conclusion

Congratulations on finishing this first Bonus Unit ■

You've just mastered understanding function-calling and how to fine-tune your model to do function-calling !

If we have one piece of advice now, it's to try to fine-tune different models . The best way to learn is by trying.

In the next Unit, you're going to learn how to use state-of-the-art frameworks such as smolagents , LlamaIndex and LangGraph .

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 ■

Conclusion - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/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 ■

Let's Fine-Tune Your Model for Function-Calling - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/bonus-unit1/fine-tuning>

Agents Course

Let's Fine-Tune Your Model for Function-Calling

We're now ready to fine-tune our first model for function-calling ■.

How do we train our model for function-calling?

A model training process can be divided into 3 steps:

- bullet The model is pre-trained on a large quantity of data . The output of that step is a pre-trained model . For instance, google/gemma-2-2b . It's a base model and only knows how to predict the next token without strong instruction following capabilities .
- bullet To be useful in a chat context, the model then needs to be fine-tuned to follow instructions. In this step, it can be trained by model creators, the open-source community, you, or anyone. For instance, google/gemma-2-2b-it is an instruction-tuned model by the Google Team behind the Gemma project.
- bullet The model can then be aligned to the creator's preferences. For instance, a customer service chat model that must never be impolite to customers.

Usually a complete product like Gemini or Mistral will go through all 3 steps , whereas the models you can find on Hugging Face have completed one or more steps of this training.

In this tutorial, we will build a function-calling model based on google/gemma-2-2b-it . We choose the fine-tuned model google/gemma-2-2b-it instead of the base model google/gemma-2-2b because the fine-tuned model has been improved for our use-case.

Starting from the pre-trained model would require more training in order to learn instruction following, chat AND function-calling .

By starting from the instruction-tuned model, we minimize the amount of information that our model needs to learn .

LoRA (Low-Rank Adaptation of Large Language Models)

LoRA is a popular and lightweight training technique that significantly reduces the number of trainable parameters .

It works by inserting a smaller number of new weights as an adapter into the model to train . This makes training with LoRA much faster, memory-efficient, and produces smaller model weights (a few hundred MBs), which are easier to store and share.

LoRA works by adding pairs of rank decomposition matrices to Transformer layers, typically focusing on linear layers. During training, we will “freeze” the rest of the model and will only update the weights of those newly added adapters.

By doing so, the number of parameters that we need to train drops considerably as we only need to update the adapter’s weights.

During inference, the input is passed into the adapter and the base model, or these adapter weights can be merged with the base model, resulting in no additional latency overhead.

LoRA is particularly useful for adapting large language models to specific tasks or domains while keeping resource requirements manageable. This helps reduce the memory required to train a model. If you want to learn more about how LoRA works, you should check out this tutorial .

Fine-Tuning a Model for Function-Calling

You can access the tutorial notebook [here](#) .

Then, click on to be able to run it in a Colab Notebook.

What is Function Calling? - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/bonus-unit1/what-is-function-calling>

Agents Course

What is Function Calling?

Function-calling is a way for an LLM to take actions on its environment . It was first introduced in GPT-4 , and was later reproduced in other models.

Just like the tools of an Agent, function-calling gives the model the capacity to take an action on its environment . However, the function calling capacity is learned by the model , and relies less on prompting than other agents techniques .

During Unit 1, the Agent didn't learn to use the Tools , we just provided the list, and we relied on the fact that the model was able to generalize on defining a plan using these Tools .

While here, with function-calling, the Agent is fine-tuned (trained) to use Tools .

How does the model “learn” to take an action?

In Unit 1, we explored the general workflow of an agent. Once the user has given some tools to the agent and prompted it with a query, the model will cycle through:

- bullet Think : What action(s) do I need to take in order to fulfill the objective.
- bullet Act : Format the action with the correct parameter and stop the generation.
- bullet Observe : Get back the result from the execution.

In a “typical” conversation with a model through an API, the conversation will alternate between user and assistant messages like this:

Function-calling brings new roles to the conversation !

- bullet One new role for an Action
- bullet One new role for an Observation

If we take the Mistral API as an example, it would look like this:

Yes and no , in this case and in a lot of other APIs, the model formats the action to take as an “assistant” message. The chat template will then represent this as special tokens for function-calling.

- bullet [AVAILABLE_TOOLS] – Start the list of available tools

- bullet [/AVAILABLE_TOOLS] – End the list of available tools
- bullet [TOOL_CALLS] – Make a call to a tool (i.e., take an “Action”)
- bullet [TOOL_RESULTS] – “Observe” the result of the action
- bullet [/TOOL_RESULTS] – End of the observation (i.e., the model can decode again)

We'll talk again about function-calling in this course, but if you want to dive deeper you can check this excellent documentation section .

Now that we learned what function-calling is and how it works, let's add some function-calling capabilities to a model that does not have those capacities yet : google/gemma-2-2b-it , by appending some new special tokens to the model.

To be able to do that, we need first to understand fine-tuning and LoRA .

Let's Fine-Tune Your Model for Function-Calling - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/bonus-unit1/fine-tuning>

Agents Course

Let's Fine-Tune Your Model for Function-Calling

We're now ready to fine-tune our first model for function-calling ■.

How do we train our model for function-calling?

A model training process can be divided into 3 steps:

- bullet The model is pre-trained on a large quantity of data . The output of that step is a pre-trained model . For instance, google/gemma-2-2b . It's a base model and only knows how to predict the next token without strong instruction following capabilities .
- bullet To be useful in a chat context, the model then needs to be fine-tuned to follow instructions. In this step, it can be trained by model creators, the open-source community, you, or anyone. For instance, google/gemma-2-2b-it is an instruction-tuned model by the Google Team behind the Gemma project.
- bullet The model can then be aligned to the creator's preferences. For instance, a customer service chat model that must never be impolite to customers.

Usually a complete product like Gemini or Mistral will go through all 3 steps , whereas the models you can find on Hugging Face have completed one or more steps of this training.

In this tutorial, we will build a function-calling model based on google/gemma-2-2b-it . We choose the fine-tuned model google/gemma-2-2b-it instead of the base model google/gemma-2-2b because the fine-tuned model has been improved for our use-case.

Starting from the pre-trained model would require more training in order to learn instruction following, chat AND function-calling .

By starting from the instruction-tuned model, we minimize the amount of information that our model needs to learn .

LoRA (Low-Rank Adaptation of Large Language Models)

LoRA is a popular and lightweight training technique that significantly reduces the number of trainable parameters .

It works by inserting a smaller number of new weights as an adapter into the model to train . This makes training with LoRA much faster, memory-efficient, and produces smaller model weights (a few hundred MBs), which are easier to store and share.

LoRA works by adding pairs of rank decomposition matrices to Transformer layers, typically focusing on linear layers. During training, we will “freeze” the rest of the model and will only update the weights of those newly added adapters.

By doing so, the number of parameters that we need to train drops considerably as we only need to update the adapter’s weights.

During inference, the input is passed into the adapter and the base model, or these adapter weights can be merged with the base model, resulting in no additional latency overhead.

LoRA is particularly useful for adapting large language models to specific tasks or domains while keeping resource requirements manageable. This helps reduce the memory required to train a model. If you want to learn more about how LoRA works, you should check out this tutorial .

Fine-Tuning a Model for Function-Calling

You can access the tutorial notebook [here](#) .

Then, click on to be able to run it in a Colab Notebook.

What is Function Calling? - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/bonus-unit1/what-is-function-calling>

Agents Course

What is Function Calling?

Function-calling is a way for an LLM to take actions on its environment . It was first introduced in GPT-4 , and was later reproduced in other models.

Just like the tools of an Agent, function-calling gives the model the capacity to take an action on its environment . However, the function calling capacity is learned by the model , and relies less on prompting than other agents techniques .

During Unit 1, the Agent didn't learn to use the Tools , we just provided the list, and we relied on the fact that the model was able to generalize on defining a plan using these Tools .

While here, with function-calling, the Agent is fine-tuned (trained) to use Tools .

How does the model “learn” to take an action?

In Unit 1, we explored the general workflow of an agent. Once the user has given some tools to the agent and prompted it with a query, the model will cycle through:

- bullet Think : What action(s) do I need to take in order to fulfill the objective.
- bullet Act : Format the action with the correct parameter and stop the generation.
- bullet Observe : Get back the result from the execution.

In a “typical” conversation with a model through an API, the conversation will alternate between user and assistant messages like this:

Function-calling brings new roles to the conversation !

- bullet One new role for an Action
- bullet One new role for an Observation

If we take the Mistral API as an example, it would look like this:

Yes and no , in this case and in a lot of other APIs, the model formats the action to take as an “assistant” message. The chat template will then represent this as special tokens for function-calling.

- bullet [AVAILABLE_TOOLS] – Start the list of available tools

- bullet `[/AVAILABLE_TOOLS]` – End the list of available tools
- bullet `[TOOL_CALLS]` – Make a call to a tool (i.e., take an “Action”)
- bullet `[TOOL_RESULTS]` – “Observe” the result of the action
- bullet `[/TOOL_RESULTS]` – End of the observation (i.e., the model can decode again)

We'll talk again about function-calling in this course, but if you want to dive deeper you can check this excellent documentation section .

Now that we learned what function-calling is and how it works, let's add some function-calling capabilities to a model that does not have those capacities yet : google/gemma-2-2b-it , by appending some new special tokens to the model.

To be able to do that, we need first to understand fine-tuning and LoRA .

Unit 2

Introduction to Agentic Frameworks - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/unit2/introduction>

Agents Course

Introduction to Agentic Frameworks

Welcome to this second unit, where we'll explore different agentic frameworks that can be used to build powerful agentic applications.

We will study:

- bullet In Unit 2.1: smolagents
- bullet In Unit 2.2: LlamaIndex
- bullet In Unit 2.3: LangGraph

Let's dive in! ■

When to Use an Agentic Framework

An agentic framework is not always needed when building an application around LLMs . They provide flexibility in the workflow to efficiently solve a specific task, but they're not always necessary.

Sometimes, predefined workflows are sufficient to fulfill user requests, and there is no real need for an agentic framework. If the approach to build an agent is simple, like a chain of prompts, using plain code may be enough. The advantage is that the developer will have full control and understanding of their system without abstractions .

However, when the workflow becomes more complex, such as letting an LLM call functions or using multiple agents, these abstractions start to become helpful.

Considering these ideas, we can already identify the need for some features:

- bullet An LLM engine that powers the system.
- bullet A list of tools the agent can access.
- bullet A parser for extracting tool calls from the LLM output.
- bullet A system prompt synced with the parser.
- bullet A memory system .

- bullet Error logging and retry mechanisms to control LLM mistakes. We'll explore how these topics are resolved in various frameworks, including smolagents , LlamaIndex , and LangGraph .

Agentic Frameworks Units

AI Agent Observability & Evaluation - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/bonus-unit2/introduction>

Agents Course

AI Agent Observability & Evaluation

Welcome to Bonus Unit 2 ! In this chapter, you'll explore advanced strategies for observing, evaluating, and ultimately improving the performance of your agents.

■ When Should I Do This Bonus Unit?

This bonus unit is perfect if you:

- bullet Develop and Deploy AI Agents: You want to ensure that your agents are performing reliably in production.
- bullet Need Detailed Insights: You're looking to diagnose issues, optimize performance, or understand the inner workings of your agent.
- bullet Aim to Reduce Operational Overhead: By monitoring agent costs, latency, and execution details, you can efficiently manage resources.
- bullet Seek Continuous Improvement: You're interested in integrating both real-time user feedback and automated evaluation into your AI applications.

In short, for everyone who wants to bring their agents in front of users!

■ What You'll Learn

In this unit, you'll learn:

- bullet Instrument Your Agent: Learn how to integrate observability tools via OpenTelemetry with the smolagents framework.
- bullet Monitor Metrics: Track performance indicators such as token usage (costs), latency, and error traces.

- bullet Evaluate in Real-Time: Understand techniques for live evaluation, including gathering user feedback and leveraging an LLM-as-a-judge.
- bullet Offline Analysis: Use benchmark datasets (e.g., GSM8K) to test and compare agent performance.

■ Ready to Get Started?

In the next section, you'll learn the basics of Agent Observability and Evaluation. After that, its time to see it in action!

AI Agent Observability & Evaluation - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/bonus-unit2/introduction>

Agents Course

AI Agent Observability & Evaluation

Welcome to Bonus Unit 2 ! In this chapter, you'll explore advanced strategies for observing, evaluating, and ultimately improving the performance of your agents.

■ When Should I Do This Bonus Unit?

This bonus unit is perfect if you:

- bullet Develop and Deploy AI Agents: You want to ensure that your agents are performing reliably in production.
- bullet Need Detailed Insights: You're looking to diagnose issues, optimize performance, or understand the inner workings of your agent.
- bullet Aim to Reduce Operational Overhead: By monitoring agent costs, latency, and execution details, you can efficiently manage resources.
- bullet Seek Continuous Improvement: You're interested in integrating both real-time user feedback and automated evaluation into your AI applications.

In short, for everyone who wants to bring their agents in front of users!

■ What You'll Learn

In this unit, you'll learn:

- bullet Instrument Your Agent: Learn how to integrate observability tools via OpenTelemetry with the smolagents framework.
- bullet Monitor Metrics: Track performance indicators such as token usage (costs), latency, and error traces.

- bullet Evaluate in Real-Time: Understand techniques for live evaluation, including gathering user feedback and leveraging an LLM-as-a-judge.
- bullet Offline Analysis: Use benchmark datasets (e.g., GSM8K) to test and compare agent performance.

■ Ready to Get Started?

In the next section, you'll learn the basics of Agent Observability and Evaluation. After that, its time to see it in action!

Introduction to Agentic Frameworks - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/unit2/introduction>

Agents Course

Introduction to Agentic Frameworks

Welcome to this second unit, where we'll explore different agentic frameworks that can be used to build powerful agentic applications.

We will study:

- bullet In Unit 2.1: smolagents
- bullet In Unit 2.2: LlamaIndex
- bullet In Unit 2.3: LangGraph

Let's dive in! ■

When to Use an Agentic Framework

An agentic framework is not always needed when building an application around LLMs . They provide flexibility in the workflow to efficiently solve a specific task, but they're not always necessary.

Sometimes, predefined workflows are sufficient to fulfill user requests, and there is no real need for an agentic framework. If the approach to build an agent is simple, like a chain of prompts, using plain code may be enough. The advantage is that the developer will have full control and understanding of their system without abstractions .

However, when the workflow becomes more complex, such as letting an LLM call functions or using multiple agents, these abstractions start to become helpful.

Considering these ideas, we can already identify the need for some features:

- bullet An LLM engine that powers the system.
- bullet A list of tools the agent can access.
- bullet A parser for extracting tool calls from the LLM output.
- bullet A system prompt synced with the parser.
- bullet A memory system .

- bullet Error logging and retry mechanisms to control LLM mistakes. We'll explore how these topics are resolved in various frameworks, including smolagents , LlamaIndex , and LangGraph .

Agentic Frameworks Units

Quiz: Evaluating AI Agents - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/bonus-unit2/quiz>

Agents Course

Quiz: Evaluating AI Agents

Let's assess your understanding of the agent tracing and evaluation concepts covered in this bonus unit.

This quiz is optional and ungraded.

Q1: What does observability in AI agents primarily refer to?

Which statement accurately describes the purpose of observability for AI agents?

Q2: Which of the following is NOT a common metric monitored in agent observability?

Select the metric that does not typically fall under the observability umbrella.

Q3: What best describes offline evaluation of an AI agent?

Determine the statement that correctly captures the essence of offline evaluation.

Q4: Which advantage does online evaluation of agents offer?

Pick the statement that best reflects the benefit of online evaluation.

Q5: What role does OpenTelemetry play in AI agent observability and evaluation?

Which statement best describes the role of OpenTelemetry in monitoring AI agents?

Congratulations on completing this quiz! ■ If you missed any questions, consider reviewing the content of this bonus unit for a deeper understanding. If you did well, you're ready to explore more advanced topics in agent observability and evaluation!

Bonus Unit 2: Observability and Evaluation of Agents - Hugging Face Agents Course

Source:

<https://huggingface.co/learn/agents-course/bonus-unit2/monitoring-and-evaluating-agents-notebook>

Agents Course

Bonus Unit 2: Observability and Evaluation of Agents

In this notebook, we will learn how to monitor the internal steps (traces) of our AI agent and evaluate its performance using open-source observability tools.

The ability to observe and evaluate an agent's behavior is essential for:

- bullet Debugging issues when tasks fail or produce suboptimal results
- bullet Monitoring costs and performance in real-time
- bullet Improving reliability and safety through continuous feedback

Exercise Prerequisites ■■

Before running this notebook, please be sure you have:

- Studied Introduction to Agents
- Studied The smolagents framework

Step 0: Install the Required Libraries

We will need a few libraries that allow us to run, monitor, and evaluate our agents:

Step 1: Instrument Your Agent

In this notebook, we will use Langfuse as our observability tool, but you can use any other OpenTelemetry-compatible service . The code below shows how to set environment variables for Langfuse (or any OTEL endpoint) and how to instrument your smolagent.

Note: If you are using LlamaIndex or LangGraph, you can find documentation on instrumenting them [here](#) and [here](#) .

First, let's configure the right environment variable for setting up the connection to the Langfuse OpenTelemetry endpoint.

We also need to configure our Hugging Face token for inference calls.

Next, we can set up the a tracer-provider for our configured OpenTelemetry.

Step 2: Test Your Instrumentation

Here is a simple CodeAgent from smolagents that calculates $1+1$. We run it to confirm that the instrumentation is working correctly. If everything is set up correctly, you will see logs/spans in your observability dashboard.

Check your Langfuse Traces Dashboard (or your chosen observability tool) to confirm that the spans and logs have been recorded.

Example screenshot from Langfuse:

[Link to the trace](#)

Step 3: Observe and Evaluate a More Complex Agent

Now that you have confirmed your instrumentation works, let's try a more complex query so we can see how advanced metrics (token usage, latency, costs, etc.) are tracked.

Online Evaluation

In the previous section, we learned about the difference between online and offline evaluation. Now, we will see how to monitor your agent in production and evaluate it live.

Offline Evaluation

Online evaluation is essential for live feedback, but you also need offline evaluation —systematic checks before or during development. This helps maintain quality and reliability before rolling changes into production.

Final Thoughts

In this notebook, we covered how to:

- bullet Set up Observability using smolagents + OpenTelemetry exporters
- bullet Check Instrumentation by running a simple agent
- bullet Capture Detailed Metrics (cost, latency, etc.) through an observability tools
- bullet Collect User Feedback via a Gradio interface
- bullet Use LLM-as-a-Judge to automatically evaluate outputs
- bullet Perform Offline Evaluation with a benchmark dataset

■ Happy coding!

Trace Structure

Most observability tools record a trace that contains spans , which represent each step of your agent's logic. Here, the trace contains the overall agent run and sub-spans for:

- bullet The tool calls (DuckDuckGoSearchTool)
- bullet The LLM calls (HfApiModel)

You can inspect these to see precisely where time is spent, how many tokens are used, and so on:
[Link to the trace](#)

Common Metrics to Track in Production

- bullet Costs — The smolagents instrumentation captures token usage, which you can transform into approximate costs by assigning a price per token.
- bullet Latency — Observe the time it takes to complete each step, or the entire run.
- bullet User Feedback — Users can provide direct feedback (thumbs up/down) to help refine or correct the agent.
- bullet LLM-as-a-Judge — Use a separate LLM to evaluate your agent's output in near real-time (e.g., checking for toxicity or correctness).

Below, we show examples of these metrics.

Dataset Evaluation

In offline evaluation, you typically:

- bullet Have a benchmark dataset (with prompt and expected output pairs)
- bullet Run your agent on that dataset
- bullet Compare outputs to the expected results or use an additional scoring mechanism

Below, we demonstrate this approach with the GSM8K dataset , which contains math questions and solutions.

Next, we create a dataset entity in Langfuse to track the runs. Then, we add each item from the dataset to the system. (If you're not using Langfuse, you might simply store these in your own database or local file for analysis.)

1. Costs

Below is a screenshot showing usage for Qwen2.5-Coder-32B-Instruct calls. This is useful to see costly steps and optimize your agent.

[Link to the trace](#)

2. Latency

We can also see how long it took to complete each step. In the example below, the entire conversation took 32 seconds, which you can break down by step. This helps you identify bottlenecks and optimize your agent.

[Link to the trace](#)

3. Additional Attributes

You may also pass additional attributes—such as user IDs, session IDs, or tags—by setting them on the spans. For example, smolagents instrumentation uses OpenTelemetry to attach attributes like `langfuse.user.id` or custom tags.

4. User Feedback

If your agent is embedded into a user interface, you can record direct user feedback (like a thumbs-up/down in a chat UI). Below is an example using Gradio to embed a chat with a simple feedback mechanism.

In the code snippet below, when a user sends a chat message, we capture the OpenTelemetry trace ID. If the user likes/dislikes the last answer, we attach a score to the trace.

User feedback is then captured in your observability tool:

5. LLM-as-a-Judge

LLM-as-a-Judge is another way to automatically evaluate your agent's output. You can set up a separate LLM call to gauge the output's correctness, toxicity, style, or any other criteria you care about. Workflow :

- bullet You define an Evaluation Template , e.g., "Check if the text is toxic."
- bullet Each time your agent generates output, you pass that output to your "judge" LLM with the template.
- bullet The judge LLM responds with a rating or label that you log to your observability tool.

Example from Langfuse:

You can see that the answer of this example is judged as “not toxic”.

6. Observability Metrics Overview

All of these metrics can be visualized together in dashboards. This enables you to quickly see how your agent performs across many sessions and helps you to track quality metrics over time.

Running the Agent on the Dataset

We define a helper function `run_smolagent()` that:

- bullet Starts an OpenTelemetry span
- bullet Runs our agent on the prompt
- bullet Records the trace ID in Langfuse

Then, we loop over each dataset item, run the agent, and link the trace to the dataset item. We can also attach a quick evaluation score if desired.

You can repeat this process with different:

- bullet Models (OpenAI GPT, local LLM, etc.)
- bullet Tools (search vs. no search)
- bullet Prompts (different system messages)

Then compare them side-by-side in your observability tool:

AI Agent Observability and Evaluation - Hugging Face Agents Course

Source:

<https://huggingface.co/learn/agents-course/bonus-unit2/what-is-agent-observability-and-evaluation>

Agents Course

AI Agent Observability and Evaluation

■ What is Observability?

Observability is about understanding what's happening inside your AI agent by looking at external signals like logs, metrics, and traces. For AI agents, this means tracking actions, tool usage, model calls, and responses to debug and improve agent performance.

■ Why Agent Observability Matters

Without observability, AI agents are “black boxes.” Observability tools make agents transparent, enabling you to:

- bullet Understand costs and accuracy trade-offs
- bullet Measure latency
- bullet Detect harmful language & prompt injection
- bullet Monitor user feedback

In other words, it makes your demo agent ready for production!

■ Observability Tools

Common observability tools for AI agents include platforms like Langfuse and Arize . These tools help collect detailed traces and offer dashboards to monitor metrics in real-time, making it easy to detect

problems and optimize performance.

Observability tools vary widely in their features and capabilities. Some tools are open source, benefiting from large communities that shape their roadmaps and extensive integrations. Additionally, certain tools specialize in specific aspects of LLMOps—such as observability, evaluations, or prompt management—while others are designed to cover the entire LLMOps workflow. We encourage you to explore the documentation of different options to pick a solution that works well for you.

Many agent frameworks such as smolagents use the OpenTelemetry standard to expose metadata to the observability tools. In addition to this, observability tools build custom instrumentations to allow for more flexibility in the fast moving world of LLMs. You should check the documentation of the tool you are using to see what is supported.

■ Traces and Spans

Observability tools usually represent agent runs as traces and spans.

- bullet Traces represent a complete agent task from start to finish (like handling a user query).
- bullet Spans are individual steps within the trace (like calling a language model or retrieving data).

■ Key Metrics to Monitor

Here are some of the most common metrics that observability tools monitor:

Latency: How quickly does the agent respond? Long waiting times negatively impact user experience. You should measure latency for tasks and individual steps by tracing agent runs. For example, an agent that takes 20 seconds for all model calls could be accelerated by using a faster model or by running model calls in parallel.

Costs: What's the expense per agent run? AI agents rely on LLM calls billed per token or external APIs. Frequent tool usage or multiple prompts can rapidly increase costs. For instance, if an agent calls an LLM five times for marginal quality improvement, you must assess if the cost is justified or if you could reduce the number of calls or use a cheaper model. Real-time monitoring can also help identify unexpected spikes (e.g., bugs causing excessive API loops).

Request Errors: How many requests did the agent fail? This can include API errors or failed tool calls. To make your agent more robust against these in production, you can then set up fallbacks or retries. E.g. if LLM provider A is down, you switch to LLM provider B as backup.

User Feedback: Implementing direct user evaluations provide valuable insights. This can include explicit ratings (■thumbs-up/■down, ■1-5 stars) or textual comments. Consistent negative feedback should alert you as this is a sign that the agent is not working as expected.

Implicit User Feedback: User behaviors provide indirect feedback even without explicit ratings. This can include immediate question rephrasing, repeated queries or clicking a retry button. E.g. if you see that users repeatedly ask the same question, this is a sign that the agent is not working as expected.

Accuracy: How frequently does the agent produce correct or desirable outputs? Accuracy definitions vary (e.g., problem-solving correctness, information retrieval accuracy, user satisfaction). The first step is to define what success looks like for your agent. You can track accuracy via automated checks, evaluation scores, or task completion labels. For example, marking traces as “succeeded” or “failed”.

Automated Evaluation Metrics: You can also set up automated evals. For instance, you can use an LLM to score the output of the agent e.g. if it is helpful, accurate, or not. There are also several open source libraries that help you to score different aspects of the agent. E.g. RAGAS for RAG agents or LLM Guard to detect harmful language or prompt injection.

In practice, a combination of these metrics gives the best coverage of an AI agent's health. In this chapters example notebook , we'll show you how these metrics looks in real examples but first, we'll learn how a typical evaluation workflow looks like.

■ Evaluating AI Agents

Observability gives us metrics, but evaluation is the process of analyzing that data (and performing tests) to determine how well an AI agent is performing and how it can be improved. In other words, once you have those traces and metrics, how do you use them to judge the agent and make decisions? Regular evaluation is important because AI agents are often non-deterministic and can evolve (through updates or drifting model behavior) – without evaluation, you wouldn't know if your “smart agent” is actually doing its job well or if it's regressed.

There are two categories of evaluations for AI agents: online evaluation and offline evaluation . Both are valuable, and they complement each other. We usually begin with offline evaluation, as this is the minimum necessary step before deploying any agent.

■■■ Lets see how this works in practice

In the next section, we'll see examples of how we can use observability tools to monitor and evaluate our agent.

■ Offline Evaluation

This involves evaluating the agent in a controlled setting, typically using test datasets, not live user queries. You use curated datasets where you know what the expected output or correct behavior is, and then run your agent on those.

For instance, if you built a math word-problem agent, you might have a test dataset of 100 problems with known answers. Offline evaluation is often done during development (and can be part of CI/CD pipelines) to check improvements or guard against regressions. The benefit is that it's repeatable and you can get clear accuracy metrics since you have ground truth . You might also simulate user queries and measure the agent's responses against ideal answers or use automated metrics as described above.

The key challenge with offline eval is ensuring your test dataset is comprehensive and stays relevant – the agent might perform well on a fixed test set but encounter very different queries in production. Therefore, you should keep test sets updated with new edge cases and examples that reflect real-world scenarios■. A mix of small “smoke test” cases and larger evaluation sets is useful: small sets for quick checks and larger ones for broader performance metrics■.

■ Online Evaluation

This refers to evaluating the agent in a live, real-world environment, i.e. during actual usage in production. Online evaluation involves monitoring the agent's performance on real user interactions and analyzing outcomes continuously.

For example, you might track success rates, user satisfaction scores, or other metrics on live traffic. The advantage of online evaluation is that it captures things you might not anticipate in a lab setting – you can observe model drift over time (if the agent's effectiveness degrades as input patterns shift) and catch unexpected queries or situations that weren't in your test data. It provides a true picture of how the agent behaves in the wild.

Online evaluation often involves collecting implicit and explicit user feedback, as discussed, and possibly running shadow tests or A/B tests (where a new version of the agent runs in parallel to compare against the old). The challenge is that it can be tricky to get reliable labels or scores for live interactions – you might rely on user feedback or downstream metrics (like did the user click the result).

■ Combining the two

In practice, successful AI agent evaluation blends online and offline methods. You might run regular offline benchmarks to quantitatively score your agent on defined tasks and continuously monitor live usage to catch things the benchmarks miss. For example, offline tests can catch if a code-generation agent's success rate on a known set of problems is improving, while online monitoring might alert you that users have started asking a new category of question that the agent struggles with. Combining both gives a more robust picture.

In fact, many teams adopt a loop: offline evaluation → deploy new agent version → monitor online metrics and collect new failure examples → add those examples to offline test set → iterate. This way, evaluation is continuous and ever-improving.

Bonus Unit 2: Observability and Evaluation of Agents - Hugging Face Agents Course

Source:

<https://huggingface.co/learn/agents-course/en/bonus-unit2/monitoring-and-evaluating-agents-notebook>

Agents Course

Bonus Unit 2: Observability and Evaluation of Agents

In this notebook, we will learn how to monitor the internal steps (traces) of our AI agent and evaluate its performance using open-source observability tools.

The ability to observe and evaluate an agent's behavior is essential for:

- bullet Debugging issues when tasks fail or produce suboptimal results
- bullet Monitoring costs and performance in real-time
- bullet Improving reliability and safety through continuous feedback

Exercise Prerequisites ■■

Before running this notebook, please be sure you have:

- Studied Introduction to Agents
- Studied The smolagents framework

Step 0: Install the Required Libraries

We will need a few libraries that allow us to run, monitor, and evaluate our agents:

Step 1: Instrument Your Agent

In this notebook, we will use Langfuse as our observability tool, but you can use any other OpenTelemetry-compatible service . The code below shows how to set environment variables for Langfuse (or any OTEL endpoint) and how to instrument your smolagent.

Note: If you are using LlamaIndex or LangGraph, you can find documentation on instrumenting them [here](#) and [here](#) .

First, let's configure the right environment variable for setting up the connection to the Langfuse OpenTelemetry endpoint.

We also need to configure our Hugging Face token for inference calls.

Next, we can set up the a tracer-provider for our configured OpenTelemetry.

Step 2: Test Your Instrumentation

Here is a simple CodeAgent from smolagents that calculates $1+1$. We run it to confirm that the instrumentation is working correctly. If everything is set up correctly, you will see logs/spans in your observability dashboard.

Check your Langfuse Traces Dashboard (or your chosen observability tool) to confirm that the spans and logs have been recorded.

Example screenshot from Langfuse:

[Link to the trace](#)

Step 3: Observe and Evaluate a More Complex Agent

Now that you have confirmed your instrumentation works, let's try a more complex query so we can see how advanced metrics (token usage, latency, costs, etc.) are tracked.

Online Evaluation

In the previous section, we learned about the difference between online and offline evaluation. Now, we will see how to monitor your agent in production and evaluate it live.

Offline Evaluation

Online evaluation is essential for live feedback, but you also need offline evaluation —systematic checks before or during development. This helps maintain quality and reliability before rolling changes into production.

Final Thoughts

In this notebook, we covered how to:

- bullet Set up Observability using smolagents + OpenTelemetry exporters
- bullet Check Instrumentation by running a simple agent
- bullet Capture Detailed Metrics (cost, latency, etc.) through an observability tools
- bullet Collect User Feedback via a Gradio interface
- bullet Use LLM-as-a-Judge to automatically evaluate outputs
- bullet Perform Offline Evaluation with a benchmark dataset

■ Happy coding!

Trace Structure

Most observability tools record a trace that contains spans , which represent each step of your agent's logic. Here, the trace contains the overall agent run and sub-spans for:

- bullet The tool calls (DuckDuckGoSearchTool)
- bullet The LLM calls (HfApiModel)

You can inspect these to see precisely where time is spent, how many tokens are used, and so on:
[Link to the trace](#)

Common Metrics to Track in Production

- bullet Costs — The smolagents instrumentation captures token usage, which you can transform into approximate costs by assigning a price per token.
- bullet Latency — Observe the time it takes to complete each step, or the entire run.
- bullet User Feedback — Users can provide direct feedback (thumbs up/down) to help refine or correct the agent.
- bullet LLM-as-a-Judge — Use a separate LLM to evaluate your agent's output in near real-time (e.g., checking for toxicity or correctness).

Below, we show examples of these metrics.

Dataset Evaluation

In offline evaluation, you typically:

- bullet Have a benchmark dataset (with prompt and expected output pairs)
- bullet Run your agent on that dataset
- bullet Compare outputs to the expected results or use an additional scoring mechanism

Below, we demonstrate this approach with the GSM8K dataset , which contains math questions and solutions.

Next, we create a dataset entity in Langfuse to track the runs. Then, we add each item from the dataset to the system. (If you're not using Langfuse, you might simply store these in your own database or local file for analysis.)

1. Costs

Below is a screenshot showing usage for Qwen2.5-Coder-32B-Instruct calls. This is useful to see costly steps and optimize your agent.

[Link to the trace](#)

2. Latency

We can also see how long it took to complete each step. In the example below, the entire conversation took 32 seconds, which you can break down by step. This helps you identify bottlenecks and optimize your agent.

[Link to the trace](#)

3. Additional Attributes

You may also pass additional attributes—such as user IDs, session IDs, or tags—by setting them on the spans. For example, smolagents instrumentation uses OpenTelemetry to attach attributes like `langfuse.user.id` or custom tags.

4. User Feedback

If your agent is embedded into a user interface, you can record direct user feedback (like a thumbs-up/down in a chat UI). Below is an example using Gradio to embed a chat with a simple feedback mechanism.

In the code snippet below, when a user sends a chat message, we capture the OpenTelemetry trace ID. If the user likes/dislikes the last answer, we attach a score to the trace.

User feedback is then captured in your observability tool:

5. LLM-as-a-Judge

LLM-as-a-Judge is another way to automatically evaluate your agent's output. You can set up a separate LLM call to gauge the output's correctness, toxicity, style, or any other criteria you care about.

Workflow :

- bullet You define an Evaluation Template , e.g., "Check if the text is toxic."
- bullet Each time your agent generates output, you pass that output to your "judge" LLM with the template.
- bullet The judge LLM responds with a rating or label that you log to your observability tool.

Example from Langfuse:

You can see that the answer of this example is judged as “not toxic”.

6. Observability Metrics Overview

All of these metrics can be visualized together in dashboards. This enables you to quickly see how your agent performs across many sessions and helps you to track quality metrics over time.

Running the Agent on the Dataset

We define a helper function `run_smolagent()` that:

- bullet Starts an OpenTelemetry span
- bullet Runs our agent on the prompt
- bullet Records the trace ID in Langfuse

Then, we loop over each dataset item, run the agent, and link the trace to the dataset item. We can also attach a quick evaluation score if desired.

You can repeat this process with different:

- bullet Models (OpenAI GPT, local LLM, etc.)
- bullet Tools (search vs. no search)
- bullet Prompts (different system messages)

Then compare them side-by-side in your observability tool:

Quiz: Evaluating AI Agents - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/bonus-unit2/quiz>

Agents Course

Quiz: Evaluating AI Agents

Let's assess your understanding of the agent tracing and evaluation concepts covered in this bonus unit.

This quiz is optional and ungraded.

Q1: What does observability in AI agents primarily refer to?

Which statement accurately describes the purpose of observability for AI agents?

Q2: Which of the following is NOT a common metric monitored in agent observability?

Select the metric that does not typically fall under the observability umbrella.

Q3: What best describes offline evaluation of an AI agent?

Determine the statement that correctly captures the essence of offline evaluation.

Q4: Which advantage does online evaluation of agents offer?

Pick the statement that best reflects the benefit of online evaluation.

Q5: What role does OpenTelemetry play in AI agent observability and evaluation?

Which statement best describes the role of OpenTelemetry in monitoring AI agents?

Congratulations on completing this quiz! ■ If you missed any questions, consider reviewing the content of this bonus unit for a deeper understanding. If you did well, you're ready to explore more advanced topics in agent observability and evaluation!

AI Agent Observability and Evaluation - Hugging Face Agents Course

Source:

<https://huggingface.co/learn/agents-course/en/bonus-unit2/what-is-agent-observability-and-evaluation>

Agents Course

AI Agent Observability and Evaluation

■ What is Observability?

Observability is about understanding what's happening inside your AI agent by looking at external signals like logs, metrics, and traces. For AI agents, this means tracking actions, tool usage, model calls, and responses to debug and improve agent performance.

■ Why Agent Observability Matters

Without observability, AI agents are “black boxes.” Observability tools make agents transparent, enabling you to:

- bullet Understand costs and accuracy trade-offs
- bullet Measure latency
- bullet Detect harmful language & prompt injection
- bullet Monitor user feedback

In other words, it makes your demo agent ready for production!

■ Observability Tools

Common observability tools for AI agents include platforms like Langfuse and Arize . These tools help collect detailed traces and offer dashboards to monitor metrics in real-time, making it easy to detect

problems and optimize performance.

Observability tools vary widely in their features and capabilities. Some tools are open source, benefiting from large communities that shape their roadmaps and extensive integrations. Additionally, certain tools specialize in specific aspects of LLMOps—such as observability, evaluations, or prompt management—while others are designed to cover the entire LLMOps workflow. We encourage you to explore the documentation of different options to pick a solution that works well for you.

Many agent frameworks such as smolagents use the OpenTelemetry standard to expose metadata to the observability tools. In addition to this, observability tools build custom instrumentations to allow for more flexibility in the fast moving world of LLMs. You should check the documentation of the tool you are using to see what is supported.

■ Traces and Spans

Observability tools usually represent agent runs as traces and spans.

- bullet Traces represent a complete agent task from start to finish (like handling a user query).
- bullet Spans are individual steps within the trace (like calling a language model or retrieving data).

■ Key Metrics to Monitor

Here are some of the most common metrics that observability tools monitor:

Latency: How quickly does the agent respond? Long waiting times negatively impact user experience. You should measure latency for tasks and individual steps by tracing agent runs. For example, an agent that takes 20 seconds for all model calls could be accelerated by using a faster model or by running model calls in parallel.

Costs: What's the expense per agent run? AI agents rely on LLM calls billed per token or external APIs. Frequent tool usage or multiple prompts can rapidly increase costs. For instance, if an agent calls an LLM five times for marginal quality improvement, you must assess if the cost is justified or if you could reduce the number of calls or use a cheaper model. Real-time monitoring can also help identify unexpected spikes (e.g., bugs causing excessive API loops).

Request Errors: How many requests did the agent fail? This can include API errors or failed tool calls. To make your agent more robust against these in production, you can then set up fallbacks or retries. E.g. if LLM provider A is down, you switch to LLM provider B as backup.

User Feedback: Implementing direct user evaluations provide valuable insights. This can include explicit ratings (■thumbs-up/■down, ■1-5 stars) or textual comments. Consistent negative feedback should alert you as this is a sign that the agent is not working as expected.

Implicit User Feedback: User behaviors provide indirect feedback even without explicit ratings. This can include immediate question rephrasing, repeated queries or clicking a retry button. E.g. if you see that users repeatedly ask the same question, this is a sign that the agent is not working as expected.

Accuracy: How frequently does the agent produce correct or desirable outputs? Accuracy definitions vary (e.g., problem-solving correctness, information retrieval accuracy, user satisfaction). The first step is to define what success looks like for your agent. You can track accuracy via automated checks, evaluation scores, or task completion labels. For example, marking traces as “succeeded” or “failed”.

Automated Evaluation Metrics: You can also set up automated evals. For instance, you can use an LLM to score the output of the agent e.g. if it is helpful, accurate, or not. There are also several open source libraries that help you to score different aspects of the agent. E.g. RAGAS for RAG agents or LLM Guard to detect harmful language or prompt injection.

In practice, a combination of these metrics gives the best coverage of an AI agent's health. In this chapter's example notebook, we'll show you how these metrics look in real examples but first, we'll learn how a typical evaluation workflow looks like.

■ Evaluating AI Agents

Observability gives us metrics, but evaluation is the process of analyzing that data (and performing tests) to determine how well an AI agent is performing and how it can be improved. In other words, once you have those traces and metrics, how do you use them to judge the agent and make decisions? Regular evaluation is important because AI agents are often non-deterministic and can evolve (through updates or drifting model behavior) – without evaluation, you wouldn't know if your “smart agent” is actually doing its job well or if it's regressed.

There are two categories of evaluations for AI agents: online evaluation and offline evaluation. Both are valuable, and they complement each other. We usually begin with offline evaluation, as this is the minimum necessary step before deploying any agent.

■■■ Lets see how this works in practice

In the next section, we'll see examples of how we can use observability tools to monitor and evaluate our agent.

■ Offline Evaluation

This involves evaluating the agent in a controlled setting, typically using test datasets, not live user queries. You use curated datasets where you know what the expected output or correct behavior is, and then run your agent on those.

For instance, if you built a math word-problem agent, you might have a test dataset of 100 problems with known answers. Offline evaluation is often done during development (and can be part of CI/CD pipelines) to check improvements or guard against regressions. The benefit is that it's repeatable and you can get clear accuracy metrics since you have ground truth. You might also simulate user queries and measure the agent's responses against ideal answers or use automated metrics as described above.

The key challenge with offline eval is ensuring your test dataset is comprehensive and stays relevant – the agent might perform well on a fixed test set but encounter very different queries in production. Therefore, you should keep test sets updated with new edge cases and examples that reflect real-world scenarios. A mix of small “smoke test” cases and larger evaluation sets is useful: small sets for quick checks and larger ones for broader performance metrics.

■ Online Evaluation

This refers to evaluating the agent in a live, real-world environment, i.e. during actual usage in production. Online evaluation involves monitoring the agent's performance on real user interactions and analyzing outcomes continuously.

For example, you might track success rates, user satisfaction scores, or other metrics on live traffic. The advantage of online evaluation is that it captures things you might not anticipate in a lab setting – you can observe model drift over time (if the agent’s effectiveness degrades as input patterns shift) and catch unexpected queries or situations that weren’t in your test data. It provides a true picture of how the agent behaves in the wild.

Online evaluation often involves collecting implicit and explicit user feedback, as discussed, and possibly running shadow tests or A/B tests (where a new version of the agent runs in parallel to compare against the old). The challenge is that it can be tricky to get reliable labels or scores for live interactions – you might rely on user feedback or downstream metrics (like did the user click the result).

■ Combining the two

In practice, successful AI agent evaluation blends online and offline methods. You might run regular offline benchmarks to quantitatively score your agent on defined tasks and continuously monitor live usage to catch things the benchmarks miss. For example, offline tests can catch if a code-generation agent’s success rate on a known set of problems is improving, while online monitoring might alert you that users have started asking a new category of question that the agent struggles with. Combining both gives a more robust picture.

In fact, many teams adopt a loop: offline evaluation → deploy new agent version → monitor online metrics and collect new failure examples → add those examples to offline test set → iterate. This way, evaluation is continuous and ever-improving.

Unit 3

Introduction - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/bonus-unit3/introduction>

Agents Course

Introduction

■ I want to be the very best ... ■

Welcome to this bonus unit , where you'll explore the exciting intersection of AI Agents and games ! ■■
Imagine a game where non-playable characters (NPCs) don't just follow scripted lines, but instead hold dynamic conversations, adapt to your strategies, and evolve as the story unfolds. This is the power of combining LLMs and agentic behavior in games : it opens the door to emergent storytelling and gameplay like never before .

In this bonus unit, you'll:

- bullet Learn how to build an AI Agent that can engage in Pokémon-style turn-based battles
- bullet Play against it, or even challenge other agents online

We've already seen some examples from the AI community for playing Pokémon using LLMs, and in this unit you'll learn how you can replicate that using your own Agent with the ideas that you've learnt through the course.

Want to go further?

- bullet ■ Master LLMs in Games : Dive deeper into game development with our full course Machine Learning for Games Course .
- bullet ■ Get the AI Playbook : Discover insights, ideas, and practical tips in the AI Playbook for Game Developers , where the future of intelligent game design is explored.

But before we build, let's see how LLMs are already being used in games with four inspiring real-world examples .

Introduction - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/bonus-unit3/introduction>

Agents Course

Introduction

■ I want to be the very best ... ■

Welcome to this bonus unit , where you'll explore the exciting intersection of AI Agents and games ! ■■
Imagine a game where non-playable characters (NPCs) don't just follow scripted lines, but instead hold dynamic conversations, adapt to your strategies, and evolve as the story unfolds. This is the power of combining LLMs and agentic behavior in games : it opens the door to emergent storytelling and gameplay like never before .

In this bonus unit, you'll:

- bullet Learn how to build an AI Agent that can engage in Pokémon-style turn-based battles
- bullet Play against it, or even challenge other agents online

We've already seen some examples from the AI community for playing Pokémon using LLMs, and in this unit you'll learn how you can replicate that using your own Agent with the ideas that you've learnt through the course.

Want to go further?

- bullet ■ Master LLMs in Games : Dive deeper into game development with our full course Machine Learning for Games Course .
- bullet ■ Get the AI Playbook : Discover insights, ideas, and practical tips in the AI Playbook for Game Developers , where the future of intelligent game design is explored.

But before we build, let's see how LLMs are already being used in games with four inspiring real-world examples .

Conclusion - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/bonus-unit3/conclusion>

Agents Course

Conclusion

If you've made it this far, congratulations! ■ You've successfully built your very own Pokémon battle agent! ■■■

You've conquered the fundamentals of Agentic workflows , connected an LLM to a game environment, and deployed an intelligent Agent ready to face the challenges of battle.

But the journey doesn't end here! Now that you have your first Agent up and running, think about how you can evolve it further:

- bullet Can you improve its strategic thinking?
- bullet How would a memory mechanism or feedback loop change its performance?
- bullet What experiments could help make it more competitive in battle?

We'd love to hear your thoughts on the course and how we can make it even better for future learners.

Got feedback? ■ Fill out this form

Thanks for learning with us, and remember:

Keep learning, Keep training, keep battling, and stay awesome! ■

Conclusion - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/bonus-unit3/conclusion>

Agents Course

Conclusion

If you've made it this far, congratulations! ■ You've successfully built your very own Pokémon battle agent! ■■■

You've conquered the fundamentals of Agentic workflows , connected an LLM to a game environment, and deployed an intelligent Agent ready to face the challenges of battle.

But the journey doesn't end here! Now that you have your first Agent up and running, think about how you can evolve it further:

- bullet Can you improve its strategic thinking?
- bullet How would a memory mechanism or feedback loop change its performance?
- bullet What experiments could help make it more competitive in battle?

We'd love to hear your thoughts on the course and how we can make it even better for future learners.

Got feedback? ■ Fill out this form

Thanks for learning with us, and remember:

Keep learning, Keep training, keep battling, and stay awesome! ■

Launching Your Pokémon Battle Agent - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/bonus-unit3/launching_agent_battle

Agents Course

Launching Your Pokémon Battle Agent

It's now time to battle! ■■

Battle the Stream Agent!

If you don't feel like building your own agent, and you're just curious about the battle potential of agents in pokémon. We are hosting an automated livestream on twitch

To battle the agent in stream you can:

Instructions:

- bullet Go to the Pokémon Showdown Space : [Link Here](#)
- bullet Choose Your Name (Top-right corner).
- bullet Find the Current Agent's Username . Check: The Stream Display : [Link Here](#)
- bullet The Stream Display : [Link Here](#)
- bullet Search for that username on the Showdown Space and Send a Battle Invitation .

Heads Up: Only one agent is online at once! Make sure you've got the right name.

Pokémon Battle Agent Challenger

If you've created your own Pokémon Battle Agent from the last section, you're probably wondering: how can I test it against others? Let's find out!

We've built a dedicated Hugging Face Space for this purpose:

This Space is connected to our own Pokémon Showdown server , where your Agent can take on others in epic AI-powered battles.

How to Launch Your Agent

Follow these steps to bring your Agent to life in the arena:

- bullet Duplicate the Space Click the three dots in the top-right menu of the Space and select “Duplicate this Space”.
- bullet Add Your Agent Code to agent.py Open the file and paste your Agent implementation. You can follow this example or check out the project structure for guidance.
- bullet Register Your Agent in app.py Add your Agent’s name and logic to the dropdown menu. Refer to this snippet for inspiration.
- bullet Select Your Agent Once added, your Agent will show up in the “Select Agent” dropdown menu. Pick it from the list! ■
- bullet Enter Your Pokémon Showdown Username Make sure the username matches the one shown in the iframe’s “Choose name” input. You can also connect with your official account.
- bullet Click “Send Battle Invitation” Your Agent will send an invite to the selected opponent. It should appear on-screen!
- bullet Accept the Battle & Enjoy the Fight! Let the battle begin! May the smartest Agent win

Ready to see your creation in action? Let the AI showdown commence! ■

Build Your Own Pokémon Battle Agent - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/bonus-unit3/building_your_pokemon_agent

Agents Course

Build Your Own Pokémon Battle Agent

Now that you've explored the potential and limitations of Agentic AI in games, it's time to get hands-on. In this section, you'll build your very own AI Agent to battle in Pokémon-style turn-based combat, using everything you've learned throughout the course.

We'll break the system into four key building blocks:

- bullet Poke-env: A Python library designed to train rule-based or reinforcement learning Pokémon bots.
- bullet Pokémon Showdown: An online battle simulator where your agent will fight.
- bullet LLMAgentBase: A custom Python class we've built to connect your LLM with the Poke-env battle environment.
- bullet TemplateAgent: A starter template you'll complete to create your own unique battle agent.

Let's explore each of these components in more detail.

■ Poke-env

Poke-env is a Python interface originally built for training reinforcement learning bots by Haris Sahovic, but we've repurposed it for Agentic AI. It allows your agent to interact with Pokémon Showdown through a simple API.

It provides a Player class from which your Agent will inherit, covering everything needed to communicate with the graphical interface.

Documentation : poke-env.readthedocs.io Repository : github.com/hsahovic/poke-env

■■ Pokémon Showdown

Pokémon Showdown is an open-source battle simulator where your agent will play live Pokémon battles. It provides a full interface to simulate and display battles in real time. In our challenge, your bot will act just like a human player, choosing moves turn by turn.

We've deployed a server that all participants will use to battle. Let's see who builds the best AI battle Agent!

Repository : github.com/smogon/Pokemon-Showdown Website : pokemonshowdown.com

■ LLMAgentBase

LLMAgentBase is a Python class that extends the Player class from Poke-env . It serves as the bridge between your LLM and the Pokémon battle simulator , handling input/output formatting and maintaining battle context.

This base agent provides a set of tools (defined in STANDARD_TOOL_SCHEMA) to interact with the environment, including:

- bullet choose_move : for selecting an attack during battle
- bullet choose_switch : for switching Pokémon

The LLM should use these tools to make decisions during a match.

■ TemplateAgent

Now comes the fun part! With LLMAgentBase as your foundation, it's time to implement your own agent, with your own strategy to climb the leaderboard.

You'll start from this template and build your own logic. We've also provided three complete examples using OpenAI , Mistral , and Gemini models to guide you.

Here's a simplified version of the template:

This code won't run out of the box, it's a blueprint for your custom logic.

With all the pieces ready, it's your turn to build a competitive agent. In the next section, we'll show how to deploy your agent to our server and battle others in real-time.

Let the battle begin! ■

■ Core Logic

- bullet choose_move(battle: Battle) : This is the main method invoked each turn. It takes a Battle object and returns an action string based on the LLM's output.

■ Key Internal Methods

- bullet _format_battle_state(battle) : Converts the current battle state into a string, making it suitable for sending to the LLM.
- bullet _find_move_by_name(battle, move_name) : Finds a move by name, used in LLM responses that call choose_move .

- bullet `_find_pokemon_by_name(battle, pokemon_name)` : Locates a specific Pokémon to switch into, based on the LLM's switch command.
- bullet `_get_llm_decision(battle_state)` : This method is abstract in the base class. You'll need to implement it in your own agent (see next section), where you define how to query the LLM and parse its response.

Here's an excerpt showing how that decision-making works:

Full source code : [agents.py](#)

From LLMs to AI Agents - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/bonus-unit3/from-llm-to-agents>

Agents Course

From LLMs to AI Agents

We learned in the first unit of the course that AI Agents are able to plan and make decisions. And while LLMs have enabled more natural interactions with NPCs, Agentic AI takes it a step further by allowing characters to make decisions, plan actions, and adapt to changing environments.

To illustrate the difference, think of a classic RPG NPC:

- bullet With an LLM: the NPC might respond to your questions in a more natural, varied way. It's great for dialogue, but the NPC remains static, it won't act unless you do something first.
- bullet With Agentic AI: the NPC can decide to go look for help, set a trap, or avoid you completely, even if you're not interacting with it directly.

This small shift changes everything. We're moving from scripted responders to autonomous actors within the game world.

This shift means NPCs can now directly interact with their environment through goal-directed behaviors, ultimately leading to more dynamic and unpredictable gameplay.

Agentic AI empowers NPCs with:

- bullet Autonomy : Making independent decisions based on the game state.
- bullet Adaptability : Adjusting strategies in response to player actions.
- bullet Persistence : Remembering past interactions to inform future behavior.

This transforms NPCs from reactive entities (reacting to your inputs) into proactive participants in the game world, opening the door for innovative gameplay.

The big limitation of Agents: it's slow (for now)

However, let's not be too optimistic just yet. Despite its potential, Agentic AI currently faces challenges in real-time applications.

The reasoning and planning processes can introduce latency, making it less suitable for fast-paced games like Doom or Super Mario Bros.

Take the example of Claude Plays Pokémon . If you consider the number of tokens needed to think , plus the tokens needed to act , it becomes clear that we'd need entirely different decoding strategies to make real-time play feasible.

Most games need to run at around 30 FPS, which means a real-time AI agent would need to act 30 times per second, not currently feasible with today's agentic LLMs.

However, turn-based games like Pokémon are ideal candidates, as they allow the AI enough time to deliberate and make strategic decisions.

That's why in the next section, you'll build your very own AI Agent to battle in Pokémon-style turn-based combat, and even challenge it yourself. Let's get into it!

The State of the Art in Using LLMs in Games - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/bonus-unit3/state-of-art>

Agents Course

The State of the Art in Using LLMs in Games

To give you a sense on how much progress has been made in this field, let's examine three tech demos and one published game that showcase the integration of LLMs in gaming.

■■■■■ Covert Protocol by NVIDIA and Inworld AI

Unveiled at GDC 2024, Covert Protocol is a tech demo that places you in the shoes of a private detective.

What's interesting in this demo is the use of AI-powered NPCs that respond to your inquiries in real-time, influencing the narrative based on your interactions.

The demo is built on Unreal Engine 5, it leverages NVIDIA's Avatar Cloud Engine (ACE) and Inworld's AI to create lifelike character interactions.

Learn more here ■ Inworld AI Blog

■ NEO NPCs by Ubisoft

Also at GDC 2024, Ubisoft introduced NEO NPCs, a prototype showcasing NPCs powered by generative AI.

These characters can perceive their environment, remember past interactions, and engage in meaningful conversations with players.

The idea here is to create more immersive and responsive game worlds where the player can have true interaction with NPCs.

Learn more here ■ Inworld AI Blog

■■ Mecha BREAK Featuring NVIDIA's ACE

Mecha BREAK , an upcoming multiplayer mech battle game, integrates NVIDIA's ACE technology to bring AI-powered NPCs to life.

Players can interact with these characters using natural language, and the NPCs can recognize players and objects via webcam, thanks to GPT-4o integration. This innovation promises a more immersive and interactive gaming experience.

Learn more here ■ [NVIDIA Blog](#)

■■■■ Suck Up! by Proxima Enterprises

Finally, Suck Up! is a published game where you play as a vampire attempting to gain entry into homes by convincing AI-powered NPCs to invite you in.

Each character is driven by generative AI, allowing for dynamic and unpredictable interactions.

Learn more here ■ [Suck Up! Official Website](#)

Wait... Where Are the Agents?

After exploring these demos, you might be wondering: "These examples showcase the use of LLMs in games but they don't seem to involve Agents. So, what's the distinction, and what additional capabilities do Agents bring to the table?"

Don't worry, it's what we're going to study in the next section.

Launching Your Pokémon Battle Agent - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/en/bonus-unit3/launching_agent_battle

Agents Course

Launching Your Pokémon Battle Agent

It's now time to battle! ■■

Battle the Stream Agent!

If you don't feel like building your own agent, and you're just curious about the battle potential of agents in pokémon. We are hosting an automated livestream on twitch

To battle the agent in stream you can:

Instructions:

- bullet Go to the Pokémon Showdown Space : [Link Here](#)
- bullet Choose Your Name (Top-right corner).
- bullet Find the Current Agent's Username . Check: The Stream Display : [Link Here](#)
- bullet The Stream Display : [Link Here](#)
- bullet Search for that username on the Showdown Space and Send a Battle Invitation .

Heads Up: Only one agent is online at once! Make sure you've got the right name.

Pokémon Battle Agent Challenger

If you've created your own Pokémon Battle Agent from the last section, you're probably wondering: how can I test it against others? Let's find out!

We've built a dedicated Hugging Face Space for this purpose:

This Space is connected to our own Pokémon Showdown server , where your Agent can take on others in epic AI-powered battles.

How to Launch Your Agent

Follow these steps to bring your Agent to life in the arena:

- bullet Duplicate the Space Click the three dots in the top-right menu of the Space and select “Duplicate this Space”.
- bullet Add Your Agent Code to agent.py Open the file and paste your Agent implementation. You can follow this example or check out the project structure for guidance.
- bullet Register Your Agent in app.py Add your Agent’s name and logic to the dropdown menu. Refer to this snippet for inspiration.
- bullet Select Your Agent Once added, your Agent will show up in the “Select Agent” dropdown menu. Pick it from the list! ■
- bullet Enter Your Pokémon Showdown Username Make sure the username matches the one shown in the iframe’s “Choose name” input. You can also connect with your official account.
- bullet Click “Send Battle Invitation” Your Agent will send an invite to the selected opponent. It should appear on-screen!
- bullet Accept the Battle & Enjoy the Fight! Let the battle begin! May the smartest Agent win

Ready to see your creation in action? Let the AI showdown commence! ■

Build Your Own Pokémon Battle Agent - Hugging Face Agents Course

Source: https://huggingface.co/learn/agents-course/en/bonus-unit3/building_your_pokemon_agent

Agents Course

Build Your Own Pokémon Battle Agent

Now that you've explored the potential and limitations of Agentic AI in games, it's time to get hands-on. In this section, you'll build your very own AI Agent to battle in Pokémon-style turn-based combat, using everything you've learned throughout the course.

We'll break the system into four key building blocks:

- bullet Poke-env: A Python library designed to train rule-based or reinforcement learning Pokémon bots.
- bullet Pokémon Showdown: An online battle simulator where your agent will fight.
- bullet LLMAgentBase: A custom Python class we've built to connect your LLM with the Poke-env battle environment.
- bullet TemplateAgent: A starter template you'll complete to create your own unique battle agent.

Let's explore each of these components in more detail.

■ Poke-env

Poke-env is a Python interface originally built for training reinforcement learning bots by Haris Sahovic, but we've repurposed it for Agentic AI. It allows your agent to interact with Pokémon Showdown through a simple API.

It provides a Player class from which your Agent will inherit, covering everything needed to communicate with the graphical interface.

Documentation : poke-env.readthedocs.io Repository : github.com/hsahovic/poke-env

■■ Pokémon Showdown

Pokémon Showdown is an open-source battle simulator where your agent will play live Pokémon battles. It provides a full interface to simulate and display battles in real time. In our challenge, your bot will act just like a human player, choosing moves turn by turn.

We've deployed a server that all participants will use to battle. Let's see who builds the best AI battle Agent!

Repository : github.com/smogon/Pokemon-Showdown Website : pokemonshowdown.com

■ LLMAgentBase

LLMAgentBase is a Python class that extends the Player class from Poke-env . It serves as the bridge between your LLM and the Pokémon battle simulator , handling input/output formatting and maintaining battle context.

This base agent provides a set of tools (defined in STANDARD_TOOL_SCHEMA) to interact with the environment, including:

- bullet choose_move : for selecting an attack during battle
- bullet choose_switch : for switching Pokémon

The LLM should use these tools to make decisions during a match.

■ TemplateAgent

Now comes the fun part! With LLMAgentBase as your foundation, it's time to implement your own agent, with your own strategy to climb the leaderboard.

You'll start from this template and build your own logic. We've also provided three complete examples using OpenAI , Mistral , and Gemini models to guide you.

Here's a simplified version of the template:

This code won't run out of the box, it's a blueprint for your custom logic.

With all the pieces ready, it's your turn to build a competitive agent. In the next section, we'll show how to deploy your agent to our server and battle others in real-time.

Let the battle begin! ■

■ Core Logic

- bullet choose_move(battle: Battle) : This is the main method invoked each turn. It takes a Battle object and returns an action string based on the LLM's output.

■ Key Internal Methods

- bullet _format_battle_state(battle) : Converts the current battle state into a string, making it suitable for sending to the LLM.
- bullet _find_move_by_name(battle, move_name) : Finds a move by name, used in LLM responses that call choose_move .

- bullet `_find_pokemon_by_name(battle, pokemon_name)` : Locates a specific Pokémon to switch into, based on the LLM's switch command.
- bullet `_get_llm_decision(battle_state)` : This method is abstract in the base class. You'll need to implement it in your own agent (see next section), where you define how to query the LLM and parse its response.

Here's an excerpt showing how that decision-making works:

Full source code : [agents.py](#)

From LLMs to AI Agents - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/bonus-unit3/from-llm-to-agents>

Agents Course

From LLMs to AI Agents

We learned in the first unit of the course that AI Agents are able to plan and make decisions. And while LLMs have enabled more natural interactions with NPCs, Agentic AI takes it a step further by allowing characters to make decisions, plan actions, and adapt to changing environments.

To illustrate the difference, think of a classic RPG NPC:

- bullet With an LLM: the NPC might respond to your questions in a more natural, varied way. It's great for dialogue, but the NPC remains static, it won't act unless you do something first.
- bullet With Agentic AI: the NPC can decide to go look for help, set a trap, or avoid you completely, even if you're not interacting with it directly.

This small shift changes everything. We're moving from scripted responders to autonomous actors within the game world.

This shift means NPCs can now directly interact with their environment through goal-directed behaviors, ultimately leading to more dynamic and unpredictable gameplay.

Agentic AI empowers NPCs with:

- bullet Autonomy : Making independent decisions based on the game state.
- bullet Adaptability : Adjusting strategies in response to player actions.
- bullet Persistence : Remembering past interactions to inform future behavior.

This transforms NPCs from reactive entities (reacting to your inputs) into proactive participants in the game world, opening the door for innovative gameplay.

The big limitation of Agents: it's slow (for now)

However, let's not be too optimistic just yet. Despite its potential, Agentic AI currently faces challenges in real-time applications.

The reasoning and planning processes can introduce latency, making it less suitable for fast-paced games like Doom or Super Mario Bros.

Take the example of Claude Plays Pokémon . If you consider the number of tokens needed to think , plus the tokens needed to act , it becomes clear that we'd need entirely different decoding strategies to make real-time play feasible.

Most games need to run at around 30 FPS, which means a real-time AI agent would need to act 30 times per second, not currently feasible with today's agentic LLMs.

However, turn-based games like Pokémon are ideal candidates, as they allow the AI enough time to deliberate and make strategic decisions.

That's why in the next section, you'll build your very own AI Agent to battle in Pokémon-style turn-based combat, and even challenge it yourself. Let's get into it!

The State of the Art in Using LLMs in Games - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/bonus-unit3/state-of-art>

Agents Course

The State of the Art in Using LLMs in Games

To give you a sense on how much progress has been made in this field, let's examine three tech demos and one published game that showcase the integration of LLMs in gaming.

■■■■■ Covert Protocol by NVIDIA and Inworld AI

Unveiled at GDC 2024, Covert Protocol is a tech demo that places you in the shoes of a private detective.

What's interesting in this demo is the use of AI-powered NPCs that respond to your inquiries in real-time, influencing the narrative based on your interactions.

The demo is built on Unreal Engine 5, it leverages NVIDIA's Avatar Cloud Engine (ACE) and Inworld's AI to create lifelike character interactions.

Learn more here ■ Inworld AI Blog

■ NEO NPCs by Ubisoft

Also at GDC 2024, Ubisoft introduced NEO NPCs, a prototype showcasing NPCs powered by generative AI.

These characters can perceive their environment, remember past interactions, and engage in meaningful conversations with players.

The idea here is to create more immersive and responsive game worlds where the player can have true interaction with NPCs.

Learn more here ■ Inworld AI Blog

■■ Mecha BREAK Featuring NVIDIA's ACE

Mecha BREAK , an upcoming multiplayer mech battle game, integrates NVIDIA's ACE technology to bring AI-powered NPCs to life.

Players can interact with these characters using natural language, and the NPCs can recognize players and objects via webcam, thanks to GPT-4o integration. This innovation promises a more immersive and interactive gaming experience.

Learn more here ■ [NVIDIA Blog](#)

■■■■ Suck Up! by Proxima Enterprises

Finally, Suck Up! is a published game where you play as a vampire attempting to gain entry into homes by convincing AI-powered NPCs to invite you in.

Each character is driven by generative AI, allowing for dynamic and unpredictable interactions.

Learn more here ■ [Suck Up! Official Website](#)

Wait... Where Are the Agents?

After exploring these demos, you might be wondering: "These examples showcase the use of LLMs in games but they don't seem to involve Agents. So, what's the distinction, and what additional capabilities do Agents bring to the table?"

Don't worry, it's what we're going to study in the next section.

Unit 4

Welcome to the final Unit - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/unit4/introduction>

Agents Course

Welcome to the final Unit

Welcome to the final unit of the course! ■

So far, you've built a strong foundation in AI Agents , from understanding their components to creating your own. With this knowledge, you're now ready to build powerful agents and stay up-to-date with the latest advancements in this fast-evolving field.

This unit is all about applying what you've learned. It's your final hands-on project , and completing it is your ticket to earning the course certificate .

What's the challenge?

You'll create your own agent and evaluate its performance using a subset of the GAIA benchmark .

To successfully complete the course, your agent needs to score 30% or higher on the benchmark.

Achieve that, and you'll earn your Certificate of Completion , officially recognizing your expertise. ■

Additionally, see how you stack up against your peers! A dedicated Student Leaderboard is available for you to submit your scores and see the community's progress.

Sounds exciting? Let's get started! ■

Welcome to the final Unit - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/unit4/introduction>

Agents Course

Welcome to the final Unit

Welcome to the final unit of the course! ■

So far, you've built a strong foundation in AI Agents , from understanding their components to creating your own. With this knowledge, you're now ready to build powerful agents and stay up-to-date with the latest advancements in this fast-evolving field.

This unit is all about applying what you've learned. It's your final hands-on project , and completing it is your ticket to earning the course certificate .

What's the challenge?

You'll create your own agent and evaluate its performance using a subset of the GAIA benchmark .

To successfully complete the course, your agent needs to score 30% or higher on the benchmark.

Achieve that, and you'll earn your Certificate of Completion , officially recognizing your expertise. ■

Additionally, see how you stack up against your peers! A dedicated Student Leaderboard is available for you to submit your scores and see the community's progress.

Sounds exciting? Let's get started! ■

Conclusion - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/unit4/conclusion>

Agents Course

Conclusion

Congratulations on finishing the Agents Course!

Through perseverance and dedication, you've built a solid foundation in the world of AI Agents.

But finishing this course is not the end of your journey . It's just the beginning: don't hesitate to explore the next section where we share curated resources to help you continue learning, including advanced topics like MCPs and beyond.

Thank you for being part of this course. We hope you liked this course as much as we loved writing it . And don't forget: Keep Learning, Stay Awesome ■

Conclusion - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/unit4/conclusion>

Agents Course

Conclusion

Congratulations on finishing the Agents Course!

Through perseverance and dedication, you've built a solid foundation in the world of AI Agents.

But finishing this course is not the end of your journey . It's just the beginning: don't hesitate to explore the next section where we share curated resources to help you continue learning, including advanced topics like MCPs and beyond.

Thank you for being part of this course. We hope you liked this course as much as we loved writing it . And don't forget: Keep Learning, Stay Awesome ■

And now? What topics I should learn? - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/unit4/additional-readings>

Agents Course

And now? What topics I should learn?

Agentic AI is a rapidly evolving field, and understanding foundational protocols is essential for building intelligent, autonomous systems.

Two important standards you should get familiar with are:

- bullet The Model Context Protocol (MCP)
- bullet The Agent-to-Agent Protocol (A2A)

■ Model Context Protocol (MCP)

The Model Context Protocol (MCP) by Anthropic is an open standard that enables AI models to securely and seamlessly connect with external tools, data sources, and applications, making agents more capable and autonomous.

Think of MCP as a universal adapter, like a USB-C port, that allows AI models to plug into various digital environments without needing custom integration for each one.

MCP is quickly gaining traction across the industry, with major companies like OpenAI and Google beginning to adopt it.

■ Learn more:

- bullet Anthropic's official announcement and documentation
- bullet MCP on Wikipedia
- bullet Blog on MCP

■ Agent-to-Agent (A2A) Protocol

Google has developed the Agent-to-Agent (A2A) protocol as a complementary counterpart to Anthropic's Model Context Protocol (MCP).

While MCP connects agents to external tools, A2A connects agents to each other , paving the way for cooperative, multi-agent systems that can work together to solve complex problems.

■ Dive deeper into A2A:

- bullet Google's A2A announcement

And now? What topics I should learn? - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/unit4/additional-readings>

Agents Course

And now? What topics I should learn?

Agentic AI is a rapidly evolving field, and understanding foundational protocols is essential for building intelligent, autonomous systems.

Two important standards you should get familiar with are:

- bullet The Model Context Protocol (MCP)
- bullet The Agent-to-Agent Protocol (A2A)

■ Model Context Protocol (MCP)

The Model Context Protocol (MCP) by Anthropic is an open standard that enables AI models to securely and seamlessly connect with external tools, data sources, and applications, making agents more capable and autonomous.

Think of MCP as a universal adapter, like a USB-C port, that allows AI models to plug into various digital environments without needing custom integration for each one.

MCP is quickly gaining traction across the industry, with major companies like OpenAI and Google beginning to adopt it.

■ Learn more:

- bullet Anthropic's official announcement and documentation
- bullet MCP on Wikipedia
- bullet Blog on MCP

■ Agent-to-Agent (A2A) Protocol

Google has developed the Agent-to-Agent (A2A) protocol as a complementary counterpart to Anthropic's Model Context Protocol (MCP).

While MCP connects agents to external tools, A2A connects agents to each other , paving the way for cooperative, multi-agent systems that can work together to solve complex problems.

■ Dive deeper into A2A:

- bullet Google's A2A announcement

Claim Your Certificate ■ - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/unit4/get-your-certificate>

Agents Course

Claim Your Certificate ■

If you scored above 30%, congratulations! ■ You're now eligible to claim your official certificate. Follow the steps below to receive it:

- bullet Visit the certificate page .
- bullet Sign in with your Hugging Face account using the button provided.
- bullet Enter your full name . This is the name that will appear on your certificate.
- bullet Click "Get My Certificate" to verify your score and download your certificate.

Once you've got your certificate, feel free to:

- bullet Add it to your LinkedIn profile ■■■
- bullet Share it on X , Bluesky , etc. ■

Don't forget to tag @huggingface . We'd be super proud and we'd love to cheer you on! ■

Hands-On - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/unit4/hands-on>

Agents Course

Hands-On

Now that you're ready to dive deeper into the creation of your final agent, let's see how you can submit it for review.

The Dataset

The Dataset used in this leaderboard consist of 20 questions extracted from the level 1 questions of the validation set from GAIA. The chosen question were filtered based on the number of tools and steps needed to answer a question.

Based on the current look of the GAIA benchmark, we think that getting you to try to aim for 30% on level 1 question is a fair test.

The process

Now the big question in your mind is probably : "How do I start submitting ?"

For this Unit, we created an API that will allow you to get the questions, and send your answers for scoring. Here is a summary of the routes (see the live documentation for interactive details):

- bullet GET /questions : Retrieve the full list of filtered evaluation questions.
- bullet GET /random-question : Fetch a single random question from the list.
- bullet GET /files/{task_id} : Download a specific file associated with a given task ID.
- bullet POST /submit : Submit agent answers, calculate the score, and update the leaderboard.

The submit function will compare the answer to the ground truth in an EXACT MATCH manner, hence prompt it well ! The GAIA team shared a prompting example for your agent here

■ Make the Template Your Own!

To demonstrate the process of interacting with the API, we've included a basic template as a starting point. Please feel free—and actively encouraged—to change, add to, or completely restructure it! Modify it in any way that best suits your approach and creativity.

In order to submit this templates compute 3 things needed by the API :

- bullet Username: Your Hugging Face username (here obtained via Gradio login), which is used to identify your submission.
- bullet Code Link (`agent_code`): the URL linking to your Hugging Face Space code (`.../tree/main`) for verification purposes, so please keep you space public.
- bullet Answers (`answers`): The list of responses (`{"task_id": ..., "submitted_answer": ...}`) generated by your Agent for scoring.

Hence we encourage you to start by duplicating this template on your own huggingface profile.

■ Check out the leaderboard here

A friendly note: This leaderboard is meant for fun! We know it's possible to submit scores without full verification. If we see too many high scores posted without a public link to back them up, we might need to review, adjust, or remove some entries to keep the leaderboard useful. The leaderboard will show the link to your space code-base, since this leaderboard is for students only, please keep your space public if you get a score you're proud of.

What is GAIA? - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/unit4/what-is-gaia>

Agents Course

What is GAIA?

GAIA is a benchmark designed to evaluate AI assistants on real-world tasks that require a combination of core capabilities—such as reasoning, multimodal understanding, web browsing, and proficient tool use.

It was introduced in the paper "GAIA: A Benchmark for General AI Assistants".

The benchmark features 466 carefully curated questions that are conceptually simple for humans, yet remarkably challenging for current AI systems.

To illustrate the gap:

- bullet Humans : ~92% success rate
- bullet GPT-4 with plugins : ~15%
- bullet Deep Research (OpenAI) : 67.36% on the validation set

GAIA highlights the current limitations of AI models and provides a rigorous benchmark to evaluate progress toward truly general-purpose AI assistants.

■ GAIA's Core Principles

GAIA is carefully designed around the following pillars:

- bullet ■ Real-world difficulty : Tasks require multi-step reasoning, multimodal understanding, and tool interaction.
- bullet ■ Human interpretability : Despite their difficulty for AI, tasks remain conceptually simple and easy to follow for humans.
- bullet ■■ Non-gameability : Correct answers demand full task execution, making brute-forcing ineffective.
- bullet ■ Simplicity of evaluation : Answers are concise, factual, and unambiguous—ideal for benchmarking.

Difficulty Levels

GAIA tasks are organized into three levels of increasing complexity , each testing specific skills:

- bullet Level 1 : Requires less than 5 steps and minimal tool usage.
- bullet Level 2 : Involves more complex reasoning and coordination between multiple tools and 5-10 steps.
- bullet Level 3 : Demands long-term planning and advanced integration of various tools.

Example of a Hard GAIA Question

As you can see, this question challenges AI systems in several ways:

- bullet Requires a structured response format
- bullet Involves multimodal reasoning (e.g., analyzing images)
- bullet Demands multi-hop retrieval of interdependent facts: Identifying the fruits in the painting
Discovering which ocean liner was used in The Last Voyage Looking up the breakfast menu from October 1949 for that ship
- bullet Identifying the fruits in the painting
- bullet Discovering which ocean liner was used in The Last Voyage
- bullet Looking up the breakfast menu from October 1949 for that ship
- bullet Needs correct sequencing and high-level planning to solve in the right order

This kind of task highlights where standalone LLMs often fall short, making GAIA an ideal benchmark for agent-based systems that can reason, retrieve, and execute over multiple steps and modalities.

Live Evaluation

To encourage continuous benchmarking, GAIA provides a public leaderboard hosted on Hugging Face , where you can test your models against 300 testing questions .

■ Check out the leaderboard here

Want to dive deeper into GAIA?

- bullet ■ Read the full paper
- bullet ■ Deep Research release post by OpenAI
- bullet ■ Open-source DeepResearch – Freeing our search agents

Claim Your Certificate ■ - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/unit4/get-your-certificate>

Agents Course

Claim Your Certificate ■

If you scored above 30%, congratulations! ■ You're now eligible to claim your official certificate. Follow the steps below to receive it:

- bullet Visit the certificate page .
- bullet Sign in with your Hugging Face account using the button provided.
- bullet Enter your full name . This is the name that will appear on your certificate.
- bullet Click "Get My Certificate" to verify your score and download your certificate.

Once you've got your certificate, feel free to:

- bullet Add it to your LinkedIn profile ■■■
- bullet Share it on X , Bluesky , etc. ■

Don't forget to tag @huggingface . We'd be super proud and we'd love to cheer you on! ■

Hands-On - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/unit4/hands-on>

Agents Course

Hands-On

Now that you're ready to dive deeper into the creation of your final agent, let's see how you can submit it for review.

The Dataset

The Dataset used in this leaderboard consist of 20 questions extracted from the level 1 questions of the validation set from GAIA. The chosen question were filtered based on the number of tools and steps needed to answer a question.

Based on the current look of the GAIA benchmark, we think that getting you to try to aim for 30% on level 1 question is a fair test.

The process

Now the big question in your mind is probably : "How do I start submitting ?"

For this Unit, we created an API that will allow you to get the questions, and send your answers for scoring. Here is a summary of the routes (see the live documentation for interactive details):

- bullet GET /questions : Retrieve the full list of filtered evaluation questions.
- bullet GET /random-question : Fetch a single random question from the list.
- bullet GET /files/{task_id} : Download a specific file associated with a given task ID.
- bullet POST /submit : Submit agent answers, calculate the score, and update the leaderboard.

The submit function will compare the answer to the ground truth in an EXACT MATCH manner, hence prompt it well ! The GAIA team shared a prompting example for your agent here

■ Make the Template Your Own!

To demonstrate the process of interacting with the API, we've included a basic template as a starting point. Please feel free—and actively encouraged—to change, add to, or completely restructure it! Modify it in any way that best suits your approach and creativity.

In order to submit this templates compute 3 things needed by the API :

- bullet Username: Your Hugging Face username (here obtained via Gradio login), which is used to identify your submission.
- bullet Code Link (`agent_code`): the URL linking to your Hugging Face Space code (`.../tree/main`) for verification purposes, so please keep you space public.
- bullet Answers (`answers`): The list of responses (`{"task_id": ..., "submitted_answer": ...}`) generated by your Agent for scoring.

Hence we encourage you to start by duplicating this template on your own huggingface profile.

■ Check out the leaderboard here

A friendly note: This leaderboard is meant for fun! We know it's possible to submit scores without full verification. If we see too many high scores posted without a public link to back them up, we might need to review, adjust, or remove some entries to keep the leaderboard useful. The leaderboard will show the link to your space code-base, since this leaderboard is for students only, please keep your space public if you get a score you're proud of.

What is GAIA? - Hugging Face Agents Course

Source: <https://huggingface.co/learn/agents-course/en/unit4/what-is-gaia>

Agents Course

What is GAIA?

GAIA is a benchmark designed to evaluate AI assistants on real-world tasks that require a combination of core capabilities—such as reasoning, multimodal understanding, web browsing, and proficient tool use.

It was introduced in the paper "GAIA: A Benchmark for General AI Assistants".

The benchmark features 466 carefully curated questions that are conceptually simple for humans, yet remarkably challenging for current AI systems.

To illustrate the gap:

- bullet Humans : ~92% success rate
- bullet GPT-4 with plugins : ~15%
- bullet Deep Research (OpenAI) : 67.36% on the validation set

GAIA highlights the current limitations of AI models and provides a rigorous benchmark to evaluate progress toward truly general-purpose AI assistants.

■ GAIA's Core Principles

GAIA is carefully designed around the following pillars:

- bullet ■ Real-world difficulty : Tasks require multi-step reasoning, multimodal understanding, and tool interaction.
- bullet ■ Human interpretability : Despite their difficulty for AI, tasks remain conceptually simple and easy to follow for humans.
- bullet ■■ Non-gameability : Correct answers demand full task execution, making brute-forcing ineffective.
- bullet ■ Simplicity of evaluation : Answers are concise, factual, and unambiguous—ideal for benchmarking.

Difficulty Levels

GAIA tasks are organized into three levels of increasing complexity , each testing specific skills:

- bullet Level 1 : Requires less than 5 steps and minimal tool usage.
- bullet Level 2 : Involves more complex reasoning and coordination between multiple tools and 5-10 steps.
- bullet Level 3 : Demands long-term planning and advanced integration of various tools.

Example of a Hard GAIA Question

As you can see, this question challenges AI systems in several ways:

- bullet Requires a structured response format
- bullet Involves multimodal reasoning (e.g., analyzing images)
- bullet Demands multi-hop retrieval of interdependent facts: Identifying the fruits in the painting
Discovering which ocean liner was used in The Last Voyage Looking up the breakfast menu from October 1949 for that ship
- bullet Identifying the fruits in the painting
- bullet Discovering which ocean liner was used in The Last Voyage
- bullet Looking up the breakfast menu from October 1949 for that ship
- bullet Needs correct sequencing and high-level planning to solve in the right order

This kind of task highlights where standalone LLMs often fall short, making GAIA an ideal benchmark for agent-based systems that can reason, retrieve, and execute over multiple steps and modalities.

Live Evaluation

To encourage continuous benchmarking, GAIA provides a public leaderboard hosted on Hugging Face , where you can test your models against 300 testing questions .

■ Check out the leaderboard here

Want to dive deeper into GAIA?

- bullet ■ Read the full paper
- bullet ■ Deep Research release post by OpenAI
- bullet ■ Open-source DeepResearch – Freeing our search agents