# Ollama

The Llama 3.2 instruction-tuned text only models are optimized for multilingual dialogue use cases, including agentic retrieval and summarization tasks. They outperform many of the available open source and closed chat models on common industry benchmarks.

Links Ollama

Comandos: https://www.byteplus.com/en/topic/418081?title=ollama-commands-for-windows-a-complete-guide

* ollama run
* ollama list

https://ollama.org/

<https://ollama.com/>

Ollama can be executed directly in python through the corresponding library.

## IA Frontend Tools for Python

Gradio, Streamlit, and Chainlit are popular tools for building web interfaces in Python, especially for AI applications. But each one has its own focus. Here's a clear comparison:

🧰 Main Purpose

| Framework | Main Focus |
| --- | --- |
| Gradio | Quick demos of ML models |
| Streamlit | Interactive dashboards and data visualization |
| Chainlit | Conversational interfaces with LLMs |

⚙️ Key Features

Gradio

* Super easy to use: interface ready in minutes
* Ideal for sharing models on Hugging Face
* Fewer visual customization options
* Perfect for rapid prototyping

Streamlit

* Large community and great documentation
* Powerful for data apps and visualizations
* Requires more code for simple interfaces
* Supports multipage apps and custom components

Chainlit

* Designed for chatbots and assistants using LLMs
* Direct integration with LangChain
* Supports response streaming and file uploads
* Less versatile outside of conversational use

🧪 Which One to Choose?

Want to showcase an ML model with just a few clicks? 👉 Gradio Need a complete dashboard with charts and filters? 👉 Streamlit Building a chatbot with GPT or similar models? 👉 Chainlit

## Gradio

[Quickstart](https://www.gradio.app/guides/quickstart)

# Hugging Face

Hugging Face is one of the most important platforms in the modern Artificial Intelligence ecosystem. Let me explain clearly what it is, what it is used for, and why it is so relevant.

🧩 What is it for?

Hugging Face allows you to:

* Download or use AI models
* There are thousands of pre-trained models: text, image, audio, code generators, etc.
* You can try them online or download them for local use.
* Example: search for Stable Diffusion XL or LLaMA 3 in their catalog.
* Host your own models
* You can upload and share your custom models.
* Ideal if you train your own model and want others to use it easily.
* Try models without programming
* With the Inference Playground, you can write a prompt and see the results directly in your browser.
* Access via API
* They offer APIs to integrate models into your applications without having to train anything.
* Train and deploy models
* With Hugging Face Hub + Spaces, you can create AI apps and showcase them (e.g., model demos in Gradio or Streamlit).
* Translated with DeepL.com (free version)

**⚙️ Principales componentes**

| **Componente** | **Función** |
| --- | --- |
| **Hugging Face Hub** | Repositorio principal de modelos y datasets.olla |
| **Transformers** | Librería en Python para usar modelos de lenguaje (BERT, GPT, T5, etc.). |
| **Diffusers** | Librería para usar modelos de generación de imágenes (como Stable Diffusion). |
| **Datasets** | Repositorio de conjuntos de datos para entrenar IA. |
| **Spaces** | Plataforma para alojar y compartir apps web interactivas de IA. |

Links:

[Hugging Face Transformers: Leverage Open-Source AI in Python – Real Python](https://realpython.com/huggingface-transformers/#:~:text=Before%20starting%2C%20you%E2%80%99ll%20benefit%20from%20having%20an%20intermediate,Face%20Transformers%20to%20leverage%20open-source%20AI%20in%20Python.)

[Hugging Face – The AI community building the future.](https://huggingface.co/)

# RAG

**🧠 ¿Qué es un RAG?**

**RAG = Retrieval-Augmented Generation**  
Es una técnica que combina **búsqueda de información** con **modelos generativos (LLMs)**.

👉 En lugar de que el modelo lo “invente” todo, **busca primero en una base de conocimiento real** y luego **genera la respuesta basada en esos datos**.

**⚙️ 1. Arquitectura general**

🧑 Usuario

│

▼

[Pregunta o prompt]

│

▼

🧩 1. Embedding del texto de entrada

│

▼

🧮 2. Búsqueda en la base vectorial (RAG Retrieval)

│

▼

📚 3. Recupera los documentos más relevantes

│

▼

🧠 4. Envía esos textos + la pregunta al LLM

│

▼

💬 5. El LLM genera la respuesta final (Grounded Answer)

**🧩 2. Componentes principales del RAG**

| **Componente** | **Función** | **Herramientas típicas** |
| --- | --- | --- |
| **1. Embedding model** | Convierte texto en vectores para medir similitud. | OpenAI text-embedding-3-small, bge-large, all-MiniLM-L6-v2 |
| **2. Base vectorial** | Guarda los embeddings y permite buscar los más parecidos. | Chroma, FAISS, Pinecone, Weaviate, Milvus |
| **3. LLM (modelo generativo)** | Genera la respuesta final basándose en los documentos. | GPT-4, Claude, Mistral, Llama 3, Ollama |
| **4. Pipeline o framework** | Orquesta el flujo de datos. | LangChain, LlamaIndex, Haystack, Ollama + Chroma |

# Entorno Virtual python

**Activarlo con politica elevada**

Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope Process

. .\.venv\Scripts\Activate.ps1

**Instalar referencias:**

pip install -r requirements.txt

**Verificar que estas usando el venv:**

python -c "import sys; print(sys.executable)"

pip list

**⏹️ Desactivar el entorno**

Cuando termines, simplemente ejecuta:

deactivate

# Variables de entorno

Resumen rápido: puedes definirlas

* temporalmente en la sesión (solo dura mientras la ventana esté abierta),
* permanentemente con setx (requiere nueva sesión),
* en un archivo .env y cargarlas desde Python (recomendado para proyectos),
* en la configuración de ejecución de VS Code (útil para debug/run dentro del editor).

PowerShell — definir para la sesión actual (temporal)

# asignar

$env:OLLAMA\_MODEL = "llama2"

$env:OLLAMA\_CMD\_TEMPLATE = "ollama run {model} --prompt {prompt} --temperature 0.7"

# comprobar

echo $env:OLLAMA\_MODEL

# ejecutar Chainlit usando esa sesión

chainlit run tres3\_gpt\_chainlit.py

PowerShell — definir y ejecutar en una sola línea

* $env:OLLAMA\_MODEL='llama2'; chainlit run tres3\_gpt\_chainlit.py

PowerShell — definir permanentemente (persistente)

* # setx persiste la variable en el perfil del usuario; necesitarás abrir una nueva terminal para verla.
* setx OLLAMA\_MODEL "llama2"

Nota: setx no exporta la variable a la sesión actual; abre una nueva terminal para usarla.

CMD (símbolo del sistema)

* REM temporal para la sesión actual
* set OLLAMA\_MODEL=llama2
* REM persistente
* setx OLLAMA\_MODEL "llama2"

Usar un archivo .env (recomendado para desarrollo)

1. Crea un archivo .env junto al script:

* OLLAMA\_MODEL=llama2
* OLLAMA\_CMD\_TEMPLATE=ollama run {model} --prompt {prompt} --temperature 0.7

1. Para que Python cargue ese .env automáticamente, añade a [tres3\_gpt\_chainlit.py](vscode-file://vscode-app/c:/Users/viclo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) (al inicio):

* from dotenv import load\_dotenv
* load\_dotenv() # carga variables desde .env en el cwd

(Ya incluimos python-dotenv en [requirements.txt](vscode-file://vscode-app/c:/Users/viclo/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).)

Comprobación desde Python

* import os
* print(os.getenv("OLLAMA\_MODEL"))
* print(os.getenv("OLLAMA\_CMD\_TEMPLATE"))

Sobre OLLAMA\_CMD\_TEMPLATE

* Debe incluir las llaves {model} y {prompt}; el script actual hace un split simple por espacios para construir la lista de argumentos, así que evita quoting complejo en el template. Ejemplo recomendado seguro:  
  OLLAMA\_CMD\_TEMPLATE=ollama run {model} --prompt {prompt} --temperature 0.7
* Si necesitas pasar prompts con comillas o caracteres especiales, es mejor no usar template o modificar el script para usar shlex.split() o pasar el prompt como argumento separado (puedo ajustar esto).

VS Code — configuración de ejecución/debug  
En .vscode/launch.json puedes añadir:

"configurations": [

  {

    "name": "Run Chainlit",

    "type": "python",

    "request": "launch",

    "program": "${workspaceFolder}/tres3\_gpt\_chainlit.py",

    "env": {

      "OLLAMA\_MODEL": "llama2"

    }

  }

]