Embedding

1. OpenAI Embedding (Paid)

✅ Summary

OpenAI Embedding is a method to convert text into **numerical vectors** using OpenAI's pre-trained models. These vectors capture the **meaning** of text so that computers can compare, search, and analyze language more intelligently.

You **don't need to know deep math** to use it, but it's important to understand how it works and where it fits into real projects like search engines, chatbots, or document retrieval.

🔷 What is an Embedding?

- Think of **text embedding** as turning words/sentences into **lists of numbers**.
- Each number in that list represents a **feature** of the meaning or context.

📌 **Example:**

Text	Embedding (simplified)
"Apple"	[0.12, 0.94, 0.33, ...]
"Orange"	[0.13, 0.92, 0.36, ...]
"Dog"	[0.91, 0.23, 0.05, ...]

🧠 Similar texts = similar vectors.

What are OpenAI Embeddings?

- Numerical representations (vectors) of text that capture semantic meaning
- Each embedding is a **1536-dimensional vector** (for `text-embedding-3-small`)
- Similar content → Similar vectors → Can be compared mathematically

Key Features

Model	Dimensions	Cost/Million Tokens	Best For
`text-embedding-3-small`	1536	$0.02	Most use cases
`text-embedding-3-large`	3072	$0.13	High precision
`text-embedding-ada-002` (legacy)	1536	$0.10	Backwards compatibility

🔷 Why Use OpenAI Embeddings?

OpenAI provides **state-of-the-art embeddings** via an API. They are:

- **Pre-trained** on a massive amount of data.
- Fast and easy to use.
- Perfect for use cases like:
  - Semantic Search
  - Document similarity
  - Q&A over large texts
  - Chatbots with memory
  - Clustering

🔷 Code Example: OpenAI Embedding

✅ Step 1: Install Required Packages

```
pip install openai langchain
```

✅ Step 2: Code (LangChain + OpenAI Embeddings)

```
from langchain.embeddings import OpenAIEmbeddings
```

```
# Initialize the embedding class
embedding = OpenAIEmbeddings(openai_api_key="YOUR_API_KEY")

# Sample text
texts = ["Quantum computing is the future.", "Dogs are loyal animals."]

# Generate embeddings
vectors = embedding.embed_documents(texts)

print(vectors[0][:5])  # print first 5 numbers of first vector
```

✅ Output:

You'll get vectors like:

```
[0.0194, -0.0342, 0.0821, ..., 0.0001]  # 1536-dimensional by default
```

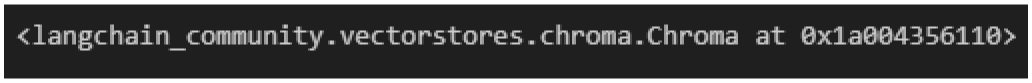
🔷 Under the Hood: Parameters of `OpenAIEmbeddings`

Parameter	Description
`openai_api_key`	Your secret API key from OpenAI
`model`	Model to use. Default: `"text-embedding-ada-002"`
`batch_size`	Number of texts sent together (default: 100)
`show_progress_bar`	Whether to show a loading bar (default: False)
`request_timeout`	Timeout in seconds (default: None)
`headers`	Any custom headers to pass
`user_agent`	String for tracking requests

💡 🧠**text-embedding-ada-002** is fast, cheap, and accurate. Most users stick with it.

Vector Embedding And Vector `StoreDB`

```
from langchain_community.vectorstores import Chroma

db=Chroma.from_documents(final_documents,embeddings_1024)
db
```



```
<langchain_community.vectorstores.chroma.Chroma at 0x1a004356110>
```

```
### Retrieve the results from query vectorstore db
query="It will be all the easier for us to conduct ourselves as belligerents"

retrieved_results=db.similarity_search(query)

print(retrieved_results)
```

```
Output:

[Document(page_content='It will be all the easier for us to conduct ourselves
as belligerents in a high spirit of right and fairness because we act without ani
mus, not in enmity toward a people or with the desire to bring any injury or dis
advantage upon them, but only in armed opposition to an irresponsible govern
ment which has thrown aside all considerations of humanity and of right and i
s running amuck. We are, let me say again, the sincere friends of the German
people, and shall desire nothing so much as the early', metadata={'source': 's
peech.txt'}), Document(page_content='Just because we fight without rancor a
nd without selfish object, seeking nothing for ourselves but what we shall wis
h to share with all free peoples, we shall, I feel confident, conduct our operatio
ns as belligerents without passion and ourselves observe with proud punctilio
the principles of right and of fair play we profess to be fighting for.\n\n...', met
adata={'source': 'speech.txt'}), Document(page_content='and shall desire not
hing so much as the early reestablishment of intimate relations of mutual adva
ntage between us—however hard it may be for them, for the time being, to bel
ieve that this is spoken from our hearts.', metadata={'source': 'speech.txt'}), D
ocument(page_content='We have borne with their present government throug
h all these bitter months because of that friendship—exercising a patience an
d forbearance which would otherwise have been impossible. We shall, happil
y, still have an opportunity to prove that friendship in our daily attitude and act
ions toward the millions of men and women of German birth and native sympa
thy who live among us and share our life, and we shall be proud to prove it to
ward all who are in fact loyal to their neighbors and to the', metadata={'sourc
e': 'speech.txt'})]
```

2. Ollama Embeddings

- **Ollama Embeddings** are *text embeddings generated by models running **locally** on your machine* via **Ollama**—a tool that lets you run open-source LLMs offline.

This is an alternative to cloud-based embeddings like OpenAI's, meaning:

- **No API key**
- **No internet** required once models are downloaded
- Useful for **private, offline** or **cost-free** usage

What Is Ollama?

Ollama is a tool that lets you:

- Download and run open-source LLMs locally (like LLaMA, Mistral, DeepSeek, etc.)
- Use them via a simple command line or Python interface
- Serve them through a **local HTTP endpoint** (default: `http://localhost:11434`)

Ollama now supports **embeddings** too.

You can generate embeddings just like OpenAI's API, but offline.

🔷 How Ollama Embeddings Work?

When using `OllamaEmbeddings`, you:

- Specify the model (e.g., `"nomic-embed-text"`)
- Send your text to the **local Ollama server**
- Get back a **vector** (list of numbers) that represents the meaning of the text

🧠 **These vectors are used for:**

- Search & similarity
- Retrieval-Augmented Generation (RAG)
- Matching user questions to documents

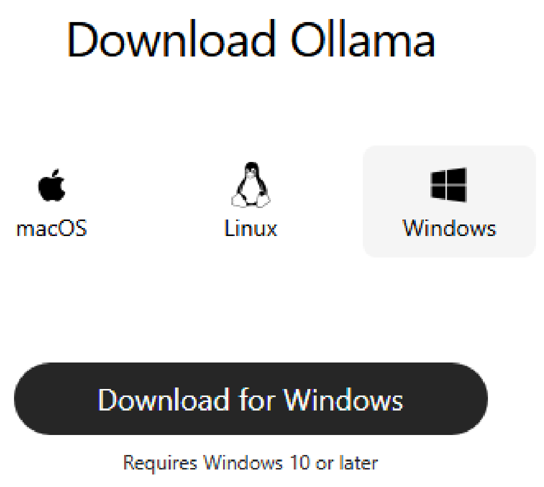
🔷 Ollama Embedding with LangChain (Python Example)

✅ Step 1: Install Requirements

```
pip install langchain langchain-community
```

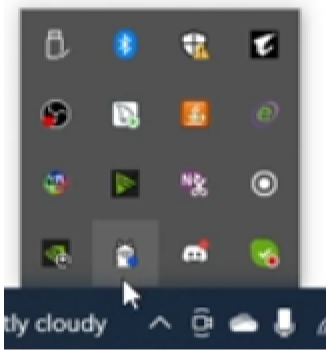
✅ Step 2: Run Ollama Locally

Install Ollama: https://ollama.com/download


Download Ollama

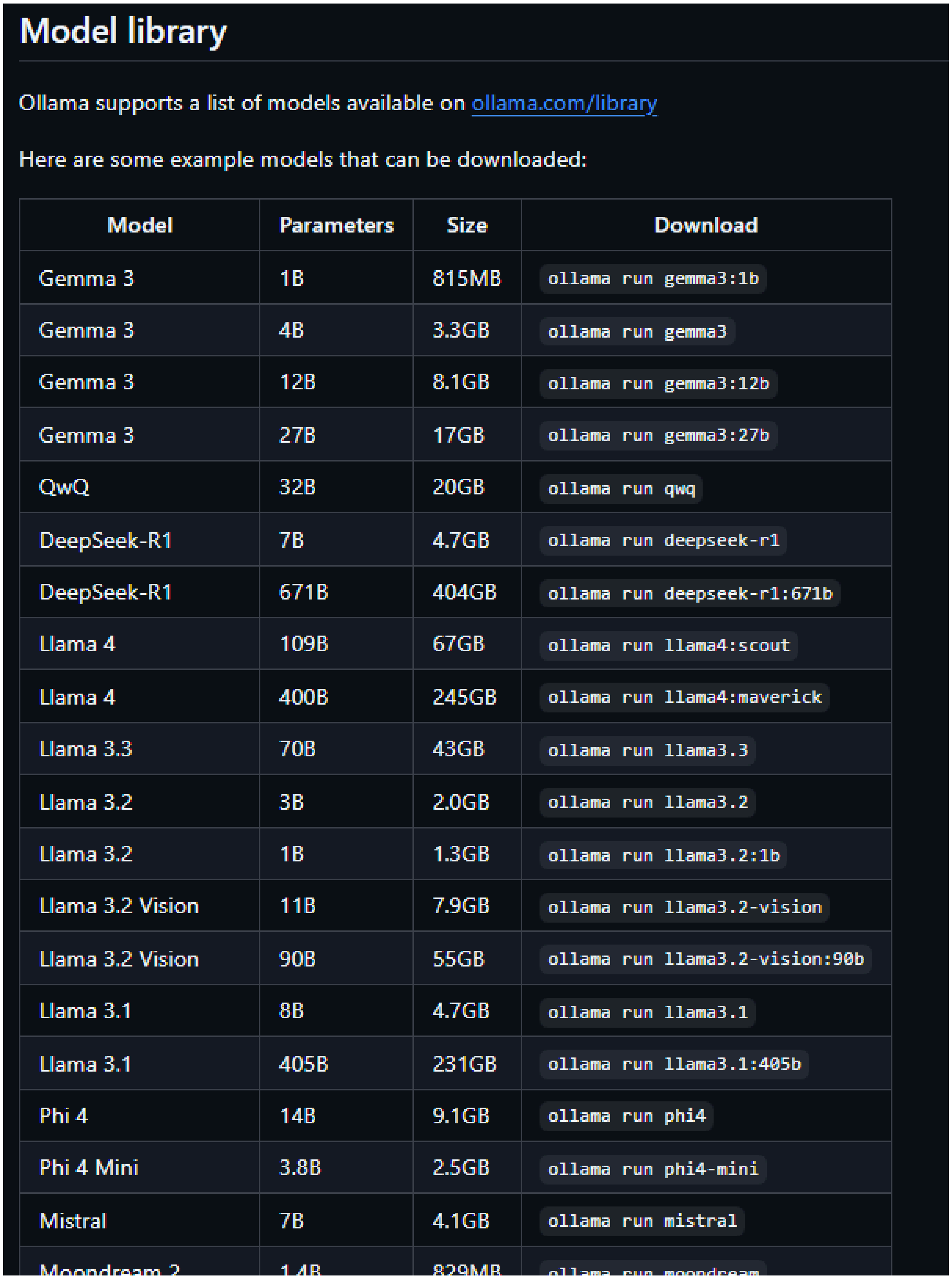
Requires Windows 10 or later

Once installed, you'll see an icon:



Available models:

https://github.com/ollama/ollama



Model library

Ollama supports a list of models available on ollama.com/library

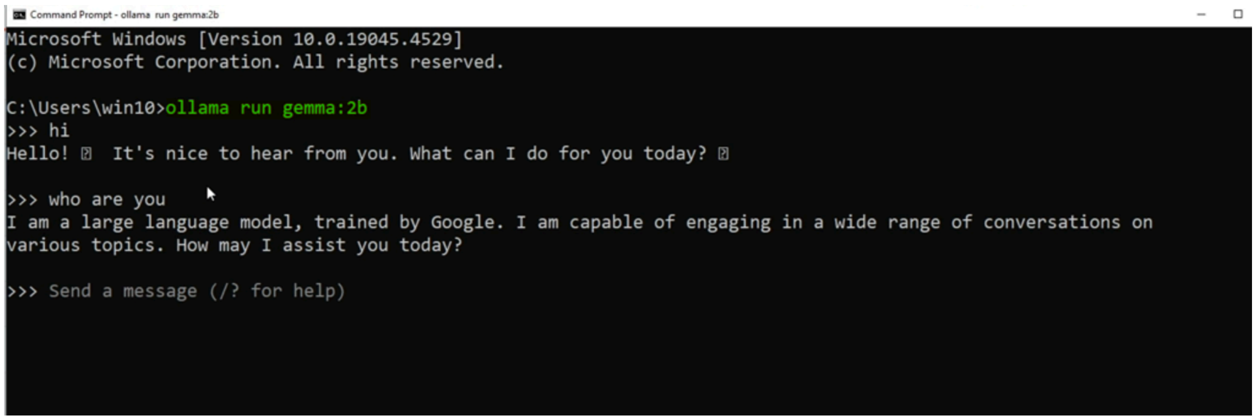
Here are some example models that can be downloaded:

Model	Parameters	Size	Download
Gemma 3	1B	815MB	`ollama run gemma3:1b`
Gemma 3	4B	3.3GB	`ollama run gemma3`
Gemma 3	12B	8.1GB	`ollama run gemma3:12b`
Gemma 3	27B	17GB	`ollama run gemma3:27b`
QwQ	32B	20GB	`ollama run qwq`
DeepSeek-R1	7B	4.7GB	`ollama run deepseek-r1`
DeepSeek-R1	671B	404GB	`ollama run deepseek-r1:671b`
Llama 4	109B	67GB	`ollama run llama4:scout`
Llama 4	400B	245GB	`ollama run llama4:maverick`
Llama 3.3	70B	43GB	`ollama run llama3.3`
Llama 3.2	3B	2.0GB	`ollama run llama3.2`
Llama 3.2	1B	1.3GB	`ollama run llama3.2:1b`
Llama 3.2 Vision	11B	7.9GB	`ollama run llama3.2-vision`
Llama 3.2 Vision	90B	55GB	`ollama run llama3.2-vision:90b`
Llama 3.1	8B	4.7GB	`ollama run llama3.1`
Llama 3.1	405B	231GB	`ollama run llama3.1:405b`
Phi 4	14B	9.1GB	`ollama run phi4`
Phi 4 Mini	3.8B	2.5GB	`ollama run phi4-mini`
Mistral	7B	4.1GB	`ollama run mistral`
Moondream 2	1.4B	829MB	`ollama run moondream`

- Open command prompt and run a model

```
ollama run gemma3:1b
```

- This will download the model in your local machine
- After this, you'll be able to chat with the model



```
Command Prompt - ollama run gemma:2b
Microsoft Windows [Version 10.0.19045.4529]
(c) Microsoft Corporation. All rights reserved.

C:\Users\win10>ollama run gemma:2b
>>> hi
Hello! ⍰  It's nice to hear from you. What can I do for you today? ⍰

>>> who are you
I am a large language model, trained by Google. I am capable of engaging in a wide range of conversations on
various topics. How may I assist you today?

>>> Send a message (/? for help)
```

✅ Step 3: Code (LangChain + Ollama Embeddings)

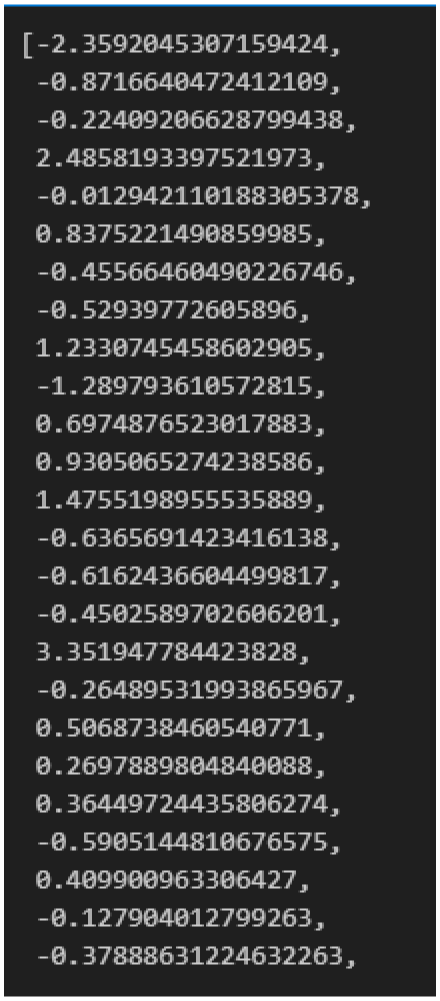
```
from langchain_community.embeddings import OllamaEmbeddings

embeddings=(
          OllamaEmbeddings(model="gemma:2b")  ##by default it ues llama2
)
embeddings
```

```
Output:

OllamaEmbeddings(base_url='http://localhost:11434', model='gemma:2b', em
bed_instruction='passage: ', query_instruction='query: ', mirostat=None, miros
tat_eta=None, mirostat_tau=None, num_ctx=None, num_gpu=None, num_thre
ad=None, repeat_last_n=None, repeat_penalty=None, temperature=None, sto
p=None, tfs_z=None, top_k=None, top_p=None, show_progress=False, heade
rs=None, model_kwargs=None)
```

```
r1=embeddings.embed_documents(
    [
       "Alpha is the first letter of Greek alphabet",
       "Beta is the second letter of Greek alphabet",
    ]
)
```

```
len(r1[0])

2048
```

```
r1[1]
```

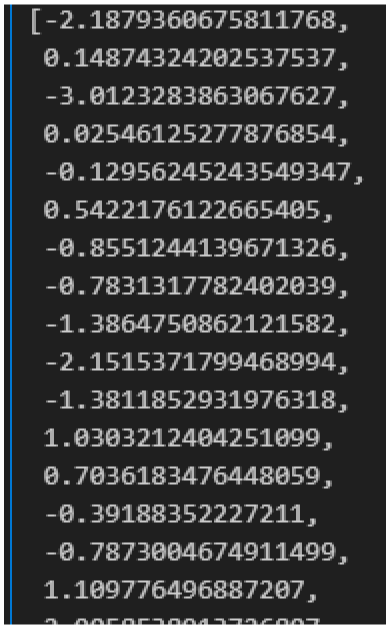


```
[-2.3592045307159424,
 -0.8716640472412109,
 -0.22409206628799438,
 2.4858193397521973,
 -0.012942110188305378,
 0.8375221490859985,
 -0.45566460490226746,
 -0.52939772605896,
 1.2330745458602905,
 -1.289793610572815,
 0.6974876523017883,
 0.9305065274238586,
 1.4755198955535889,
 -0.6365691423416138,
 -0.6162436604499817,
 -0.4502589702606201,
 3.351947784423828,
 -0.26489531993865967,
 0.5068738460540771,
 0.2697889804840088,
 0.36449724435806274,
 -0.5905144810676575,
 0.409900963306427,
 -0.127904012799263,
 -0.37888631224632263,
```

embed_query

Function	Use it for...	Example
`embed_documents()`	Text **you want to search into**	PDF pages, notes, articles
`embed_query()`	The **question asked by the user**	"What is LangChain?"

```
embeddings.embed_query("What is the second letter of Greek alphabet ")
```



```
[-2.1879360675811768,
 0.14874324202537537,
 -3.0123283863067627,
 0.02546125277876854,
 -0.12956245243549347,
 0.5422176122665405,
 -0.8551244139671326,
 -0.7831317782402039,
 -1.3864750862121582,
 -2.1515371799468994,
 -1.3811852931976318,
 1.0303212404251099,
 0.7036183476448059,
 -0.39188352227211,
 -0.7873004674911499,
 1.109776496887207,
```

🧠 Imagine This...

You have a **library of documents**.

1. You convert all the **documents** using:

   ```
   embed_documents(["doc1", "doc2", "doc3", ...])
   ```

   This builds a **vector index**.

2. Now a user asks:

   ```
   "Tell me about gravity"
   ```

   You use:

   ```
   embed_query("Tell me about gravity")
   ```

   to turn that query into a **vector**, and then you compare it to all document vectors to find the **most similar match**.

🔷 Ollama Embeddings vs OpenAI

Feature	Ollama	OpenAI
Cost	Free (after setup)	$0.0001 per 1K tokens
Internet Needed?	❌ No	✅ Yes
API Key?	❌ No	✅ Yes
Hardware Required	✅ Yes (your CPU/GPU)	❌ Runs in cloud
Privacy	✅ 100% local	❌ Data leaves your machine
Setup Time	Takes initial setup	Works instantly with API

🧠 Trivia

- Embeddings from Ollama use the **same format** (dense vectors) as OpenAI or HuggingFace.
- You can **store** them in FAISS, Chroma, or any vector DB.
- You can **combine Ollama embeddings + Ollama LLMs** for full offline systems.

3. Huggingface Embeddings

```
pip install langchain sentence-transformers

pip install langchain huggingface_hub langchain-huggingface
```

🔷 HuggingFaceHub vs HuggingFaceEmbeddings

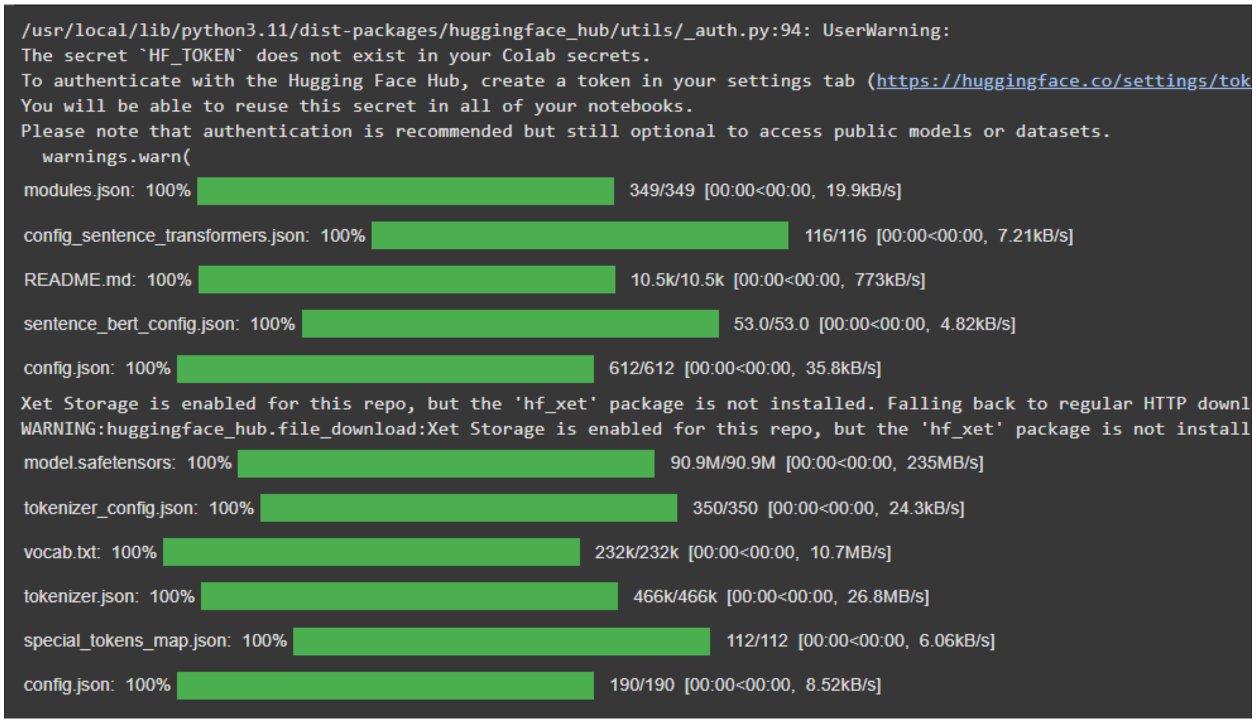
LangChain gives you two ways to use Hugging Face:

Method	Description
`HuggingFaceEmbeddings`	Runs the model **locally** using `transformers` (no token needed)

Download Model Local:

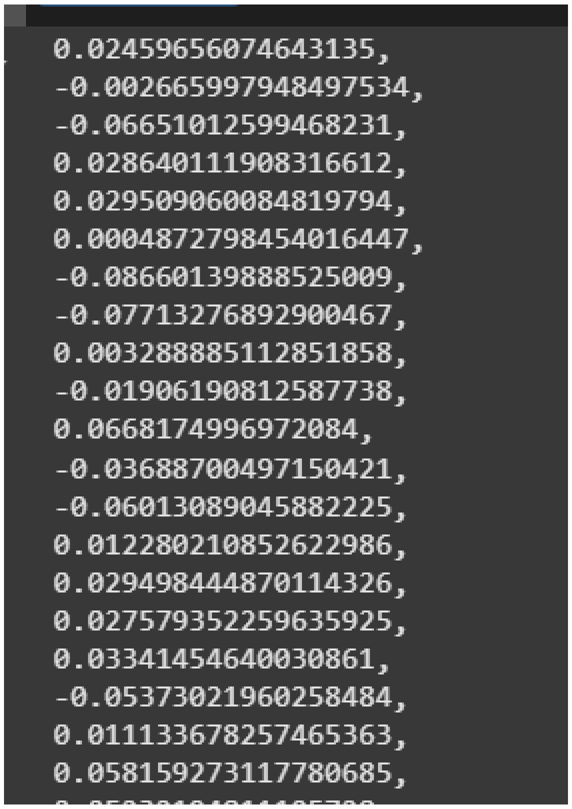
💡 **!! 👇This will download the model locally**

```
from langchain_huggingface import HuggingFaceEmbeddings
embeddings=HuggingFaceEmbeddings(model_name="all-MiniLM-L6-v2")
```



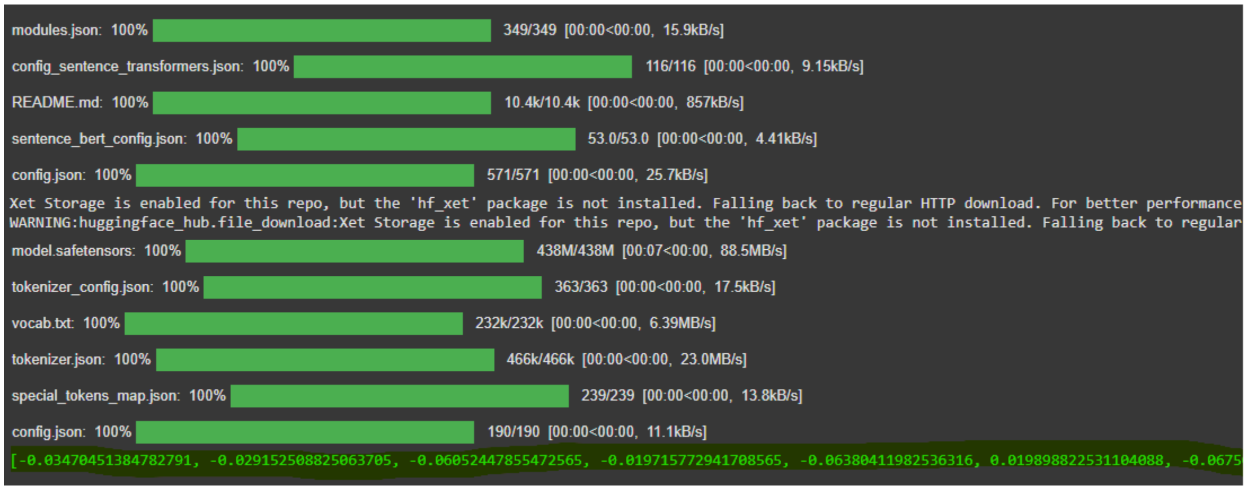
```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tok
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
modules.json: 100% 349/349 [00:00<00:00, 19.9kB/s]
config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 7.21kB/s]
README.md: 100% 10.5k/10.5k [00:00<00:00, 773kB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 4.82kB/s]
config.json: 100% 612/612 [00:00<00:00, 35.8kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP downl
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not install
model.safetensors: 100% 90.9M/90.9M [00:00<00:00, 235MB/s]
tokenizer_config.json: 100% 350/350 [00:00<00:00, 24.3kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 10.7MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 26.8MB/s]
special_tokens_map.json: 100% 112/112 [00:00<00:00, 6.06kB/s]
config.json: 100% 190/190 [00:00<00:00, 8.52kB/s]
```

```
text="this is atest documents"

query_result=embeddings.embed_query(text)
query_result
```



```
0.02459656074643135,
-0.002665997948497534,
-0.0665101259946823l,
0.028640111908316612,
0.029509060084819794,
0.0004872798454016447,
-0.08660139888525009,
-0.07713276892900467,
0.003288885112851858,
-0.01906190812587738,
0.0668174996972084,
-0.03688700497150421,
-0.06013089045882225,
0.012280210852622986,
0.029498444870114326,
0.027579352259635925,
0.03341454640030861,
-0.05373021960258484,
0.011133678257465363,
0.058159273117780685,
```

```
from langchain_huggingface import HuggingFaceEmbeddings

# Initialize the embeddings class with your model
embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-mpnet-base-v2")

# Example texts to embed
texts = ["What is quantum computing?", "Apples are fruits."]

# Generate embeddings
vector_list = embeddings.embed_documents(texts)

# Display the first 5 dimensions of the first embedding
print(vector_list[0])
```



```
modules.json: 100% 349/349 [00:00<00:00, 15.9kB/s]
config_sentence_transformers.json: 100% 116/116 [00:00<00:00, 9.15kB/s]
README.md: 100% 10.4k/10.4k [00:00<00:00, 857kB/s]
sentence_bert_config.json: 100% 53.0/53.0 [00:00<00:00, 4.41kB/s]
config.json: 100% 571/571 [00:00<00:00, 25.7kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better performance
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular
model.safetensors: 100% 438M/438M [00:07<00:00, 88.5MB/s]
tokenizer_config.json: 100% 363/363 [00:00<00:00, 17.5kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 6.39MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 23.0MB/s]
special_tokens_map.json: 100% 239/239 [00:00<00:00, 13.8kB/s]
config.json: 100% 190/190 [00:00<00:00, 11.1kB/s]
[-0.03470451384782791, -0.029152508825063705, -0.06052447855472565, -0.019715772941708565, -0.06380411982536316, 0.019898822531104088, -0.0675
```

💡 **None of the API methods are working**

🧠 Use Case Flow

LOAD → SPLIT TEXT → EMBED (HuggingFaceEmbeddings) → STORE IN FAIS S/CHROMA → SEARCH/RETRIEVE