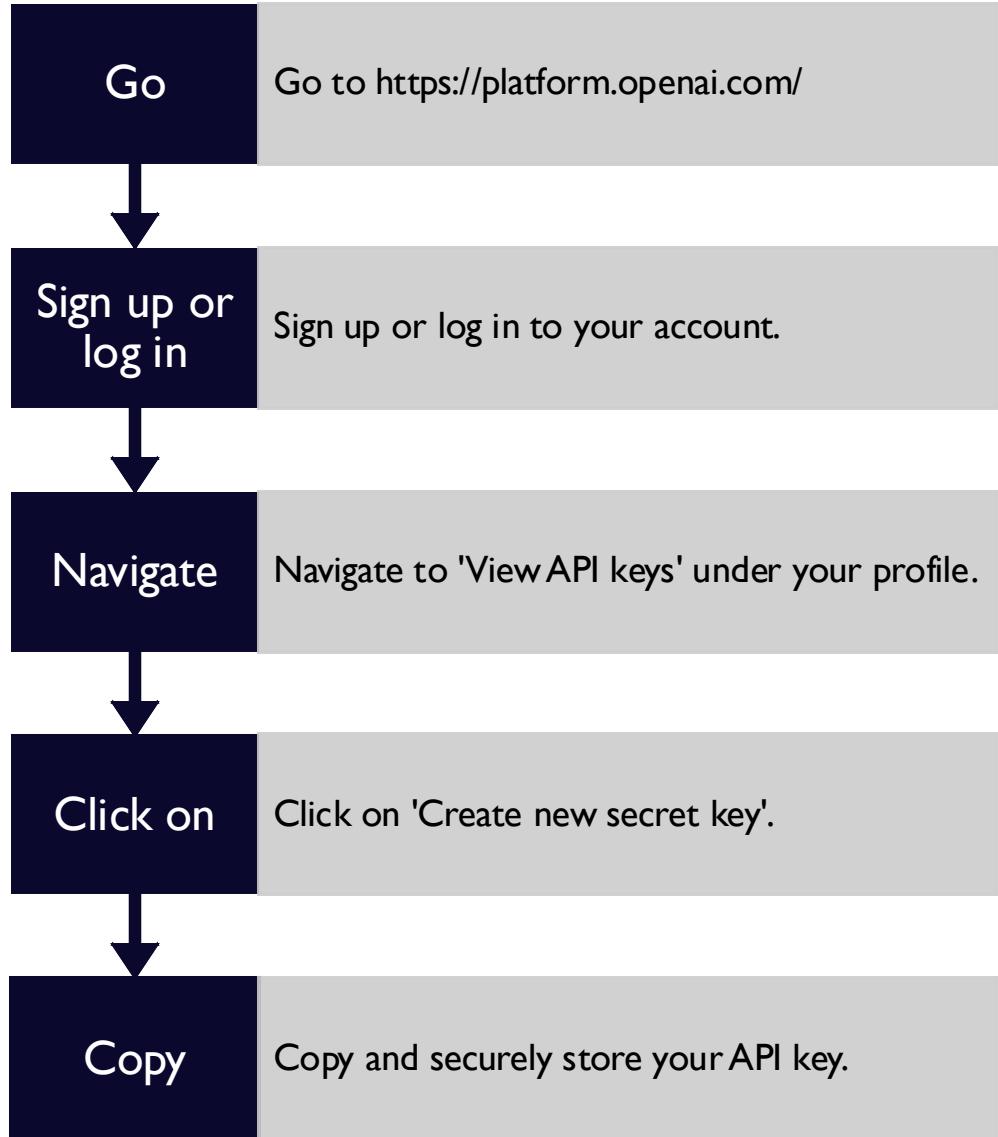


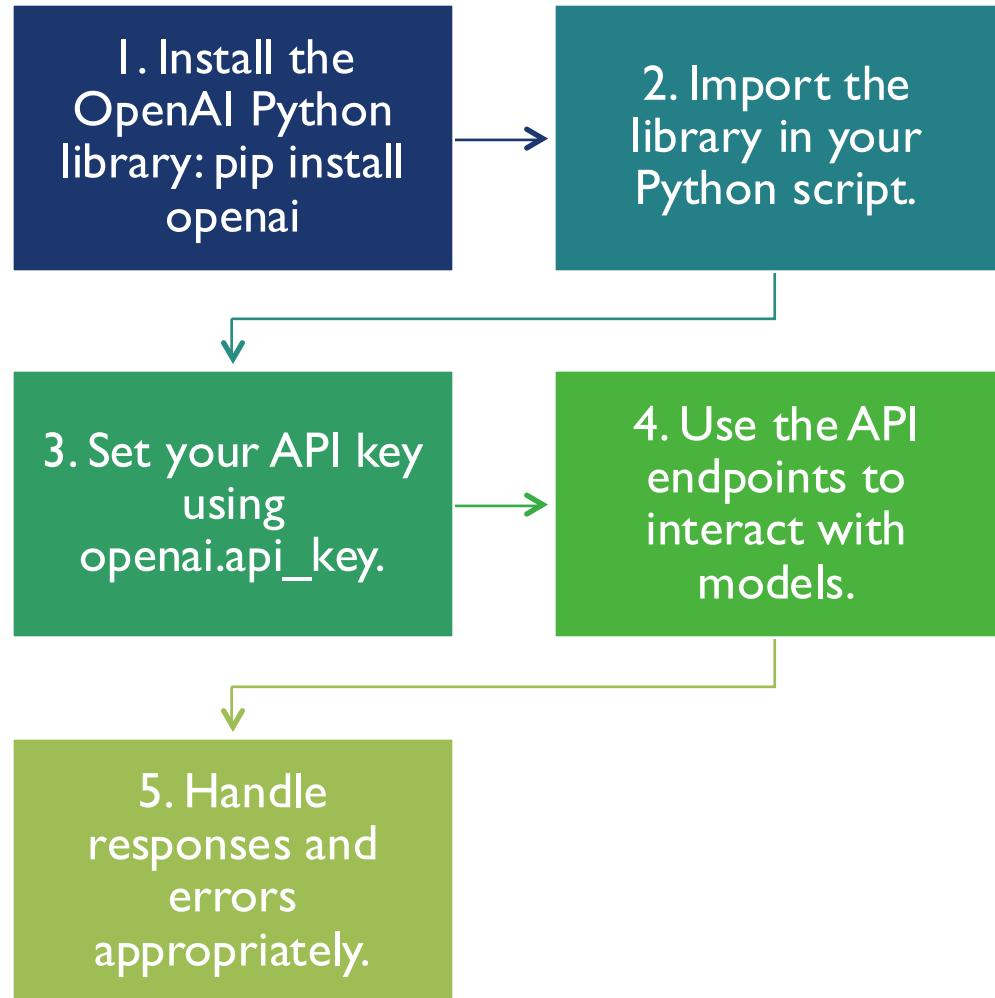
HOW TO USE OPENAI API

**CREATING API KEY AND INTERACTING
WITH OPENAI API**

HOW TO CREATE OPENAI API KEY



HOW TO INTERACT WITH OPENAI API



PYTHON CODE EXAMPLE - SETUP

```
import openai

# Set your API key
openai.api_key =
"YOUR_API_KEY"
```

PYTHON CODE EXAMPLE - TEXT COMPLETION

```
response =  
openai.Completion.create(  
    model="text-davinci-003",  
    prompt="Write a short poem  
about AI",  
    max_tokens=50  
)  
  
print(response.choices[0].text.strip()  
)
```

PYTHON CODE EXAMPLE - CHAT COMPLETION

```
response =  
openai.ChatCompletion.create(  
    model="gpt-3.5-turbo",  
    messages=[  
        {"role": "system", "content":  
            "You are a helpful assistant."},  
        {"role": "user", "content":  
            "Explain how to use OpenAI API."}  
    ]  
)  
  
print(response.choices[0].message['  
content'])
```

GENERATING EMBEDDINGS

```
# Example: Generating embeddings using
# OpenAI API

from openai import OpenAI
client = OpenAI(api_key="YOUR_API_KEY")

response = client.embeddings.create(
    model="text-embedding-3-small",
    input="OpenAI provides powerful AI
models."
)

print(response.data[0].embedding[:10]) #  
Print first 10 values of embedding
```

A detailed 3D rendering of a futuristic city skyline at sunset. The buildings are tall, rectangular structures with metallic or glass facades, illuminated from within with a warm orange glow. The city is densely packed, with many skyscrapers of varying heights. The sky above is a clear blue with a few wispy clouds. In the foreground, there's a dark, curved shape that looks like a road or a bridge.

IMAGE GENERATION API

```
# Example: Generating an image using OpenAI API
from openai import OpenAI
client = OpenAI(api_key="YOUR_API_KEY")

response = client.images.generate(
    model="gpt-image-1",
    prompt="A futuristic city skyline at sunset",
    size="1024x1024"
)

print(response.data[0].url) # URL of the generated image
```

HANDLING ERRORS AND RETRIES

```
# Example: Handling errors and retries
import time
from openai import OpenAI, error

client = OpenAI(api_key="YOUR_API_KEY")

for attempt in range(3):
    try:
        response = client.responses.create(
            model="gpt-4.1-mini",
            input="Hello, world!"
        )
        print(response.output_text)
        break
    except error.OpenAIError as e:
        print(f"Error: {e}. Retrying...")
        time.sleep(2)
```

STREAMING RESPONSES

```
# Example: Streaming responses from OpenAI API
from openai import OpenAI
client = OpenAI(api_key="YOUR_API_KEY")

with client.responses.stream(
    model="gpt-4.1-mini",
    input="Write a short poem about AI."
) as stream:
    for event in stream:
        if event.type ==
"response.output_text.delta":
            print(event.delta, end="")
    print("\nStreaming complete.")
```

GENERATING EMBEDDINGS

```
# Example: Generating embeddings using OpenAI API
from openai import OpenAI
client = OpenAI(api_key="YOUR_API_KEY")

response = client.embeddings.create(
    model="text-embedding-3-small",
    input="OpenAI provides powerful AI models."
)

print(response.data[0].embedding[:10]) # Print first 10 values of
embedding
```

IMAGE GENERATION API

```
# Example: Generating an image using OpenAI API
from openai import OpenAI
client = OpenAI(api_key="YOUR_API_KEY")

response = client.images.generate(
    model="gpt-image-1",
    prompt="A futuristic city skyline at sunset",
    size="1024x1024"
)

print(response.data[0].url) # URL of the generated image
```

HANDLING ERRORS AND RETRIES

```
# Example: Handling errors and retries

import time
from openai import OpenAI, error

client = OpenAI(api_key="YOUR_API_KEY")

for attempt in range(3):
    try:
        response = client.responses.create(
            model="gpt-4.1-mini",
            input="Hello, world!"
        )
        print(response.output_text)
        break
    except error.OpenAIError as e:
        print(f"Error: {e}. Retrying...")
        time.sleep(2)
```

STREAMING RESPONSES

```
• # Example: Streaming responses from OpenAI API
• from openai import OpenAI
• client = OpenAI(api_key="YOUR_API_KEY")
•
• with client.responses.stream(
•     model="gpt-4.1-mini",
•     input="Write a short poem about AI."
• ) as stream:
•     for event in stream:
•         if event.type == "response.output_text.delta":
•             print(event.delta, end="")
•     print("\nStreaming complete.")
•
```

WORKING WITH GEMINI API

```
pip install -q -U google-genai
```

```
from google import genai
```

```
# The client gets the API key from the environment variable  
`GEMINI_API_KEY`.
```

```
client = genai.Client()
```

```
response = client.models.generate_content(
```

```
    model="gemini-2.5-flash", contents="Explain how AI works in a few  
words"
```

```
)
```

```
print(response.text)
```

```
1 # Installing Gemini API package
2
3 !pip install -qU google-genai

1 # importing userdat and environmnet information
2 from google.colab import userdata
3 import os
4 os.environ["GOOGLE_API_KEY"] = userdata.get('GOOGLE_API_KEY')

1 # Interacting with client after configuring Gemini_api_key
2
3 from google import genai
4
5 client = genai.Client()
6
7 response = client.models.generate_content(
8     model="gemini-2.5-flash",
9     contents="Explain how AI works in a few words",
10 )
11
12 print(response.text)
```

It learns patterns from data to make smart decisions.

```
# Thinking with Gemini 2.5
from google import genai
from google.genai import types

client = genai.Client()

response = client.models.generate_content(
    model="gemini-2.5-flash",
    contents="How does AI work?",
    config=types.GenerateContentConfig(
        thinking_config=types.ThinkingConfig(thinking_budget=0) # Disables
        thinking
    ),
)
print(response.text)
```

```
1 #Working with Images ( Multi Model)
2 from PIL import Image
3 from google import genai
4
5 client = genai.Client()
6
7 image = Image.open("/path/to/organ.png")
8 response = client.models.generate_content(
9     model="gemini-2.5-flash",
10    contents=[image, "Tell me about this instrument"]
11 )
12 print(response.text)
```

```
1 import cohere
2 import os
3 from google.colab import userdata
4 os.environ["COHERE_API_KEY"] = userdata.get('COHERE_API_KEY')
5 co = cohere.ClientV2(os.environ["COHERE_API_KEY"])
6 response = co.chat(
7     model="command-a-03-2025",
8     messages=[{"role": "user", "content": "hello world!"}]
9 )
10
11 print(response.message)
```

HUGGINGFACE

The screenshot shows the Hugging Face website's main dashboard. At the top, there is a search bar with placeholder text "Search models, datasets, users...". Below the search bar is a navigation menu with links to "Models", "Datasets", "Spaces", "Community", "Docs", "Enterprise", and "Pricing". On the far right of the header is a user profile icon.

The main content area is divided into several sections:

- Tasks:** Buttons for "Text Generation", "Any-to-Any", "Image-Text-to-Text", "Image-to-Text", "Image-to-Image", "Text-to-Image", "Text-to-Video", "Text-to-Speech", and "+42".
- Parameters:** A slider scale from "<1B" to ">500B" with tick marks at 6B, 12B, 32B, and 128B.
- Libraries:** Buttons for "PyTorch", "TensorFlow", "JAX", "Transformers", "Diffusers", "Safetensors", "ONNX", "GGUF", "Transformers.js", "MLX", "MLX", "Keras", and "+41".
- Apps:** Buttons for "vLLM", "TGI", "llama.cpp", "MLX LM", "LM Studio", "Ollama", "Jan", and "+13".
- Inference Providers:** Buttons for "Grog", "Novita", "Nebius AI", "Cerebras", "SambaNova", "Nscale", "fal", "Hyperbolic", and "+10".
- Models:** A large section displaying a grid of model cards. Each card includes the model name, description, last update, size, inference status, and star count. Some cards have a "Full-text search" button.

Below the main content area, there is a footer with the text "All rights reserved : Author @ Rajendra Phani" and a page number "20".

Main Tasks Libraries Languages Licenses Other

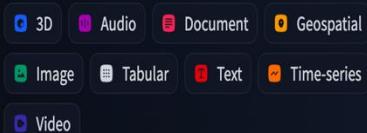
Datasets 556,879

Filter by name

Full-text search

Sort: Trending

Modalities



Size (rows)

<1K

>1T

Format



- [builddotai/Egocentric-10K](#)
Updated 8 days ago • ↓ 43.4k • ❤ 237
- [PleIAs/SYNTH](#)
Viewer • Updated 7 days ago • ⏱ 68M • ↓ 28.1k • ❤ 130
- [facebook/omnilingual-asr-corpus](#)
Viewer • Updated 4 days ago • ⏱ 548k • ↓ 23.2k • ❤ 129
- [nvidia/PhysicalAI-Autonomous-Vehicles](#)
Updated 21 days ago • ↓ 89.1k • ❤ 348
- [HuggingFaceFW/finepdfs-edu](#)
Viewer • Updated 7 days ago • ⏱ 49.5M • ↓ 6.8k • ❤ 38
- [tensonaut/EPSTEIN_FILES_20K](#)
Viewer • Updated about 9 hours ago • ⏱ 25.8k • ↓ 909 • ❤ 31
- [fka/awesome-chatgpt-prompts](#)
Viewer • Updated Jan 6 • ⏱ 203 • ↓ 38.5k • ❤ 9.4k
- [moondream/refcoco-m](#)
Viewer • Updated about 16 hours ago • ⏱ 1.19k • ↓ 984 • ❤ 23
- [openai/gsm8k](#)
Viewer • Updated Jan 4, 2024 • ⏱ 17.6k • ↓ 506k • ❤ 964
- [ServiceNow/GroundCUA](#)
Preview • Updated 1 day ago • ↓ 26.8k • ❤ 26
- [facebook/principia-collection](#)
Viewer • Updated 9 days ago • ⏱ 554k • ↓ 2.03k • ❤ 37
- [HuggingFaceFW/finepdfs](#)
Viewer • Updated 7 days ago • ⏱ 476M • ↓ 58k • ❤ 670
- [Open-Bee/Honey-Data-15M](#)
Viewer • Updated 13 days ago • ⏱ 14.8M • ↓ 111k • ❤ 94
- [HuggingFaceFW/finewiki](#)
Viewer • Updated 27 days ago • ⏱ 61.6M • ↓ 25.1k • ❤ 255
- [Seikaijuy/Beautiful-Chinese](#)
Viewer • Updated Jun 20, 2024 • ⏱ 810k • ↓ 249 • ❤ 79
- [HuggingFaceFW/fineweb](#)
Viewer • Updated Jul 12 • ⏱ 52.5B • ↓ 279k • ❤ 2.44k
- [FreedomIntelligence/medical-o1-reasoning-SFT](#)
Viewer • Updated Apr 22 • ⏱ 90.1k • ↓ 8.12k • ❤ 951
- [priyank-m/chinese_text_recognition](#)
Viewer • Updated Sep 21, 2022 • ⏱ 500k • ↓ 466 • ❤ 34

 **Hugging Face**

Models Datasets Spaces Community Docs Enterprise Pricing | 

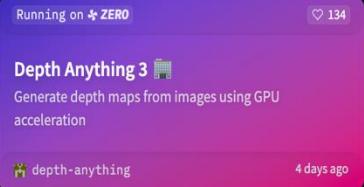
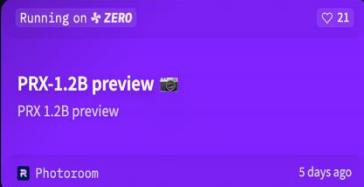
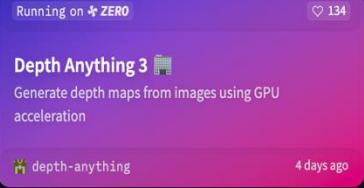
Spaces · The AI App Directory

+ New Space  Learn more 

Ask anything you want to do with AI

Image Generation Video Generation Text Generation Language Translation Speech Synthesis 3D Modeling Object Detection Text Analysis Image Editing Code Generation Question Answering Data Visualization Voice Clip

Spaces of the week < 17 Nov 2025 >  

 ERNIE-4.5-VL-28B-A3B-Thinking Demo  Compact model, powerful multimodal reasoning.  5 days ago	 Qwen-Image-Edit-2509-Photo-to-Anime  Convert photos to anime-style images  4 days ago	 Qwen Image Edit 2509 LoRAs Fast  Demo of the Collection of Qwen Image Editing LoRAs  1 day ago	 Depth Anything 3  Generate depth maps from images using GPU acceleration  4 days ago
 PRX-1.2B preview  PRX 1.2B preview  5 days ago	 Qwen Image Edit Product Fusion  Fast 4 step inference with Qwen Image Edit 2509  5 days ago	 EdgeTAM  On-Device Track Anything Model  4 days ago	 Adaptive UI  Adaptive UI  6 days ago
 Qwen Image Edit Camera Control  Fast 4 step inference with Qwen Image Edit 2509  4 days ago	 The Smol Training Playbook  The secrets to building world-class LLMs  6 days ago	 Dream-wan2-2-faster-Pro  Generate a video from an image with a prompt  15 days ago	 Depth Anything 3  Generate depth maps from images using GPU acceleration  4 days ago

All running apps, trending first

Documentation

Search across all docs

Hub & Client Libraries

- **Hub**

Host Git-based models, datasets, and Spaces on the HF Hub

- **Hub Python Library**

Python client to interact with the Hugging Face Hub

- **Huggingface.js**

JavaScript libraries for Hugging Face with built-in TS types

- **Tasks**

Explore demos, models, and datasets for any ML tasks

- **Dataset viewer**

API for metadata, stats, and content of HF Hub datasets

Deployment & Inference

- **Inference Providers**

Call 200k+ models hosted by our 10+ inference partners

- **Inference Endpoints (dedicated)**

Deploy models on dedicated & fully managed infrastructure on HF

- **Deploying on AWS**

Train/deploy models from Hugging Face to AWS with DLCs

- **Text Generation Inference**

Serve language models with TGI optimized toolkit

- **Text Embeddings Inference**

Serve embeddings models with TEI optimized toolkit

- **Microsoft Azure**

Deploy Hugging Face models on Microsoft Azure

- **Google Cloud**

Train and Deploy Hugging Face models on Google Cloud

Core ML Libraries

- **Transformers**

State-of-the-art AI models for PyTorch

- **Diffusers**

State-of-the-art Diffusion models in PyTorch

- **Datasets**

Access & share datasets for any ML tasks

- **Transformers.js**

State-of-the-art ML running directly in your browser

- **Tokenizers**

Fast tokenizers optimized for research & production

- **Evaluate**

Evaluate and compare models performance

Downloading and working with Tasks from HuggingFace

```
1 # installing transformers library
2 !pip install -qU transformers
3 !pip install -qU compressed-tensors
```

```
1 # Use a pipeline as a high-level helper
2 from transformers import pipeline
3
4 pipe = pipeline("text-generation", model="moonshotai/Kimi-K2-Thinking",
5     trust_remote_code=True)
6 messages = [
7     {"role": "user", "content": "Who are you?"},
8 ]
9 pipe(messages)
```

WORKING WITH LANGCHAIN

```
1 #Installing Langchain–OpenAI
2 !pip install -qU langchain-openai
```

```
1 # using Model Class
2 import os
3 from langchain_openai import ChatOpenAI
4 from google.colab import userdata
5 os.environ["OPENAI_API_KEY"] = userdata.get('OPENAI_API_KEY')
6 model = ChatOpenAI(model="gpt-4.1")
7 response = model.invoke("Why do parrots talk?")
8 print(response.content)
```

```
1 # using init_chat
2 import os
3 from langchain.chat_models import init_chat_model
4 from google.colab import userdata
5 os.environ["OPENAI_API_KEY"] = userdata.get('OPENAI_API_KEY')
6 model = init_chat_model("gpt-4.1")
7 response = model.invoke("Why do birds fly")
8 print(response.content)
```

RAG STEPS

- **Step 1 — Document Loaders:** Ingest data from sources (PDFs, web pages, databases, APIs). Normalize formats and metadata.
- **Step 2 — Chunking Documents:** Split long documents into semantically-coherent chunks (overlap optional). Typical chunk sizes: 200–1,000 tokens depending on model/context window.
- **Step 3 — Embedding Documents:** Encode chunks into vector representations using an embedding model (e.g., OpenAI, sentence-transformers).
- **Step 4 — Store in VectorDB:** Persist vectors + metadata in a Vector Database (Milvus, Pinecone, Weaviate, FAISS, etc.) with indexing for similarity search.
- **Step 5 — Retrieve:** Given a query, perform similarity search to fetch top-K relevant chunks; optionally re-rank or filter using metadata; then use retrieved chunks to augment the LLM prompt.

