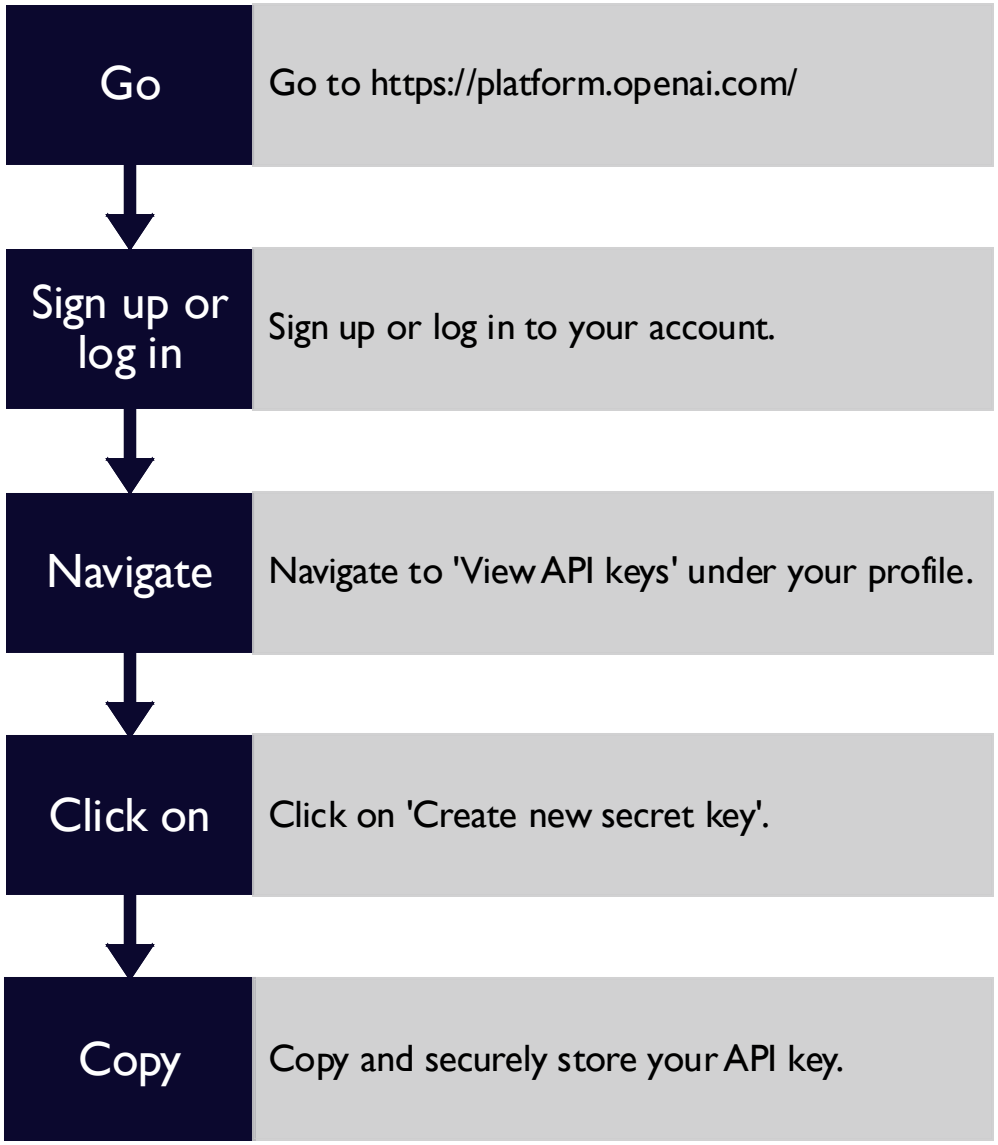
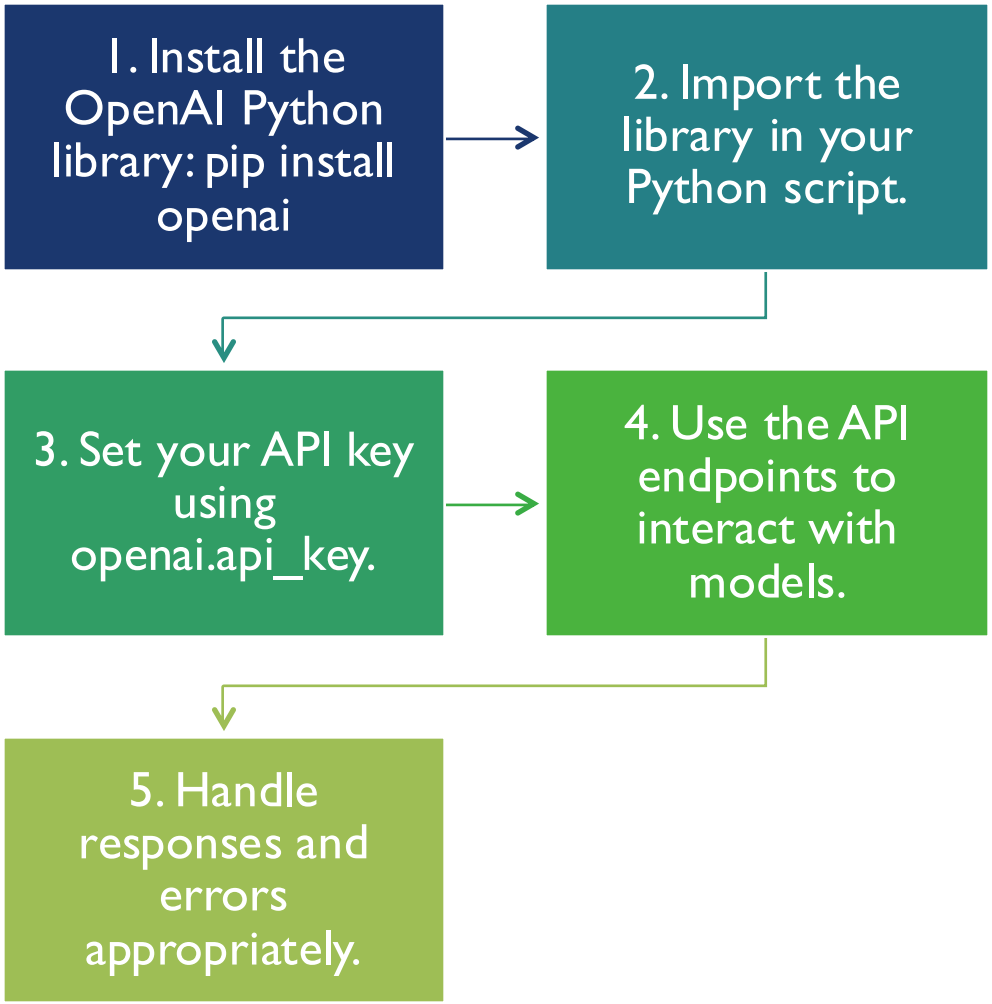
HOW TO USE OPENAI API

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
```

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)
```

```
# Installing Gemini API package

!pip install -qU google-genai
```

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

```
# Interacting with client after configuring Gemini_api_key

from google import genai

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)
```

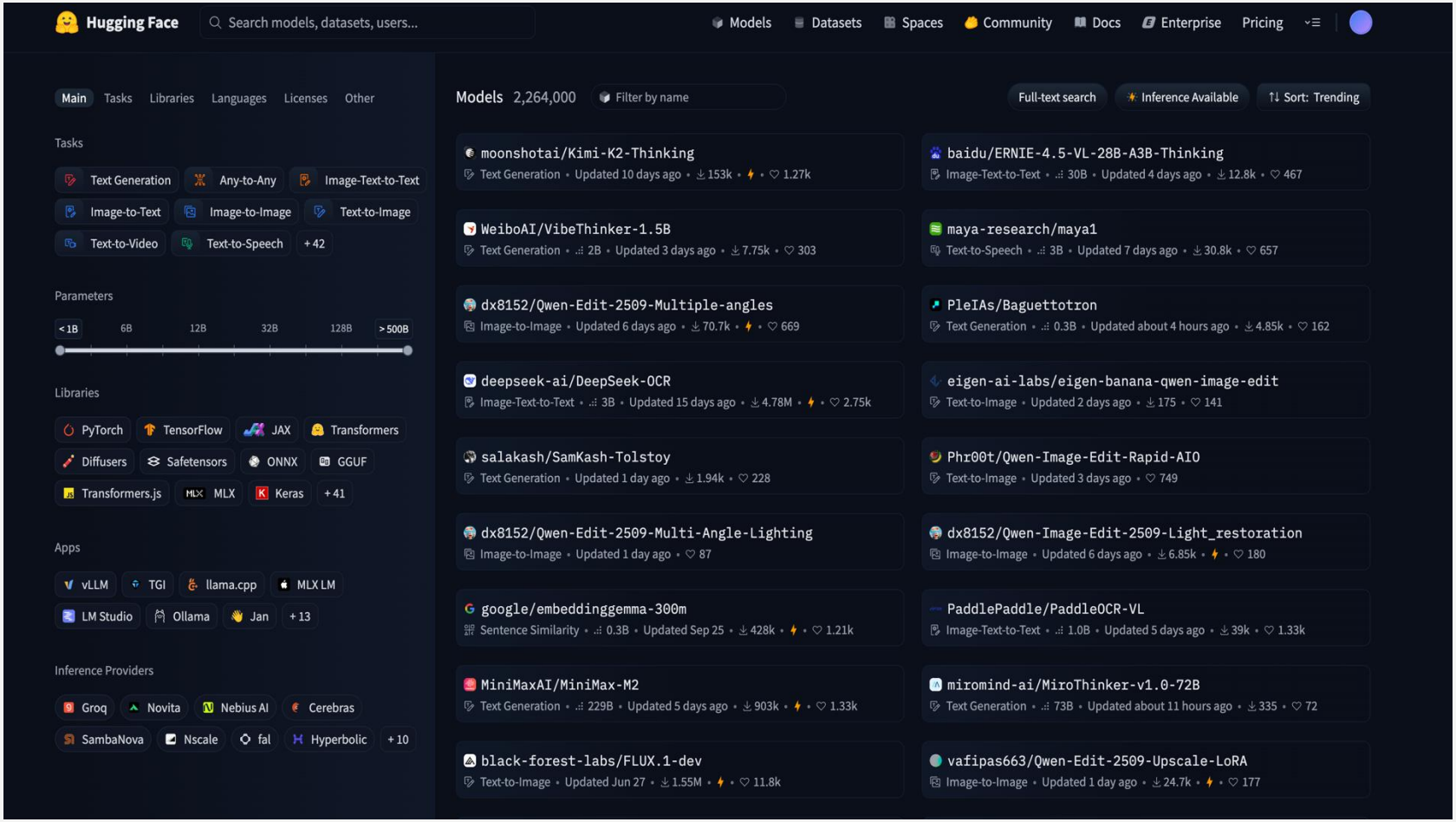
```
It learns patterns from data to make smart decisions.
```

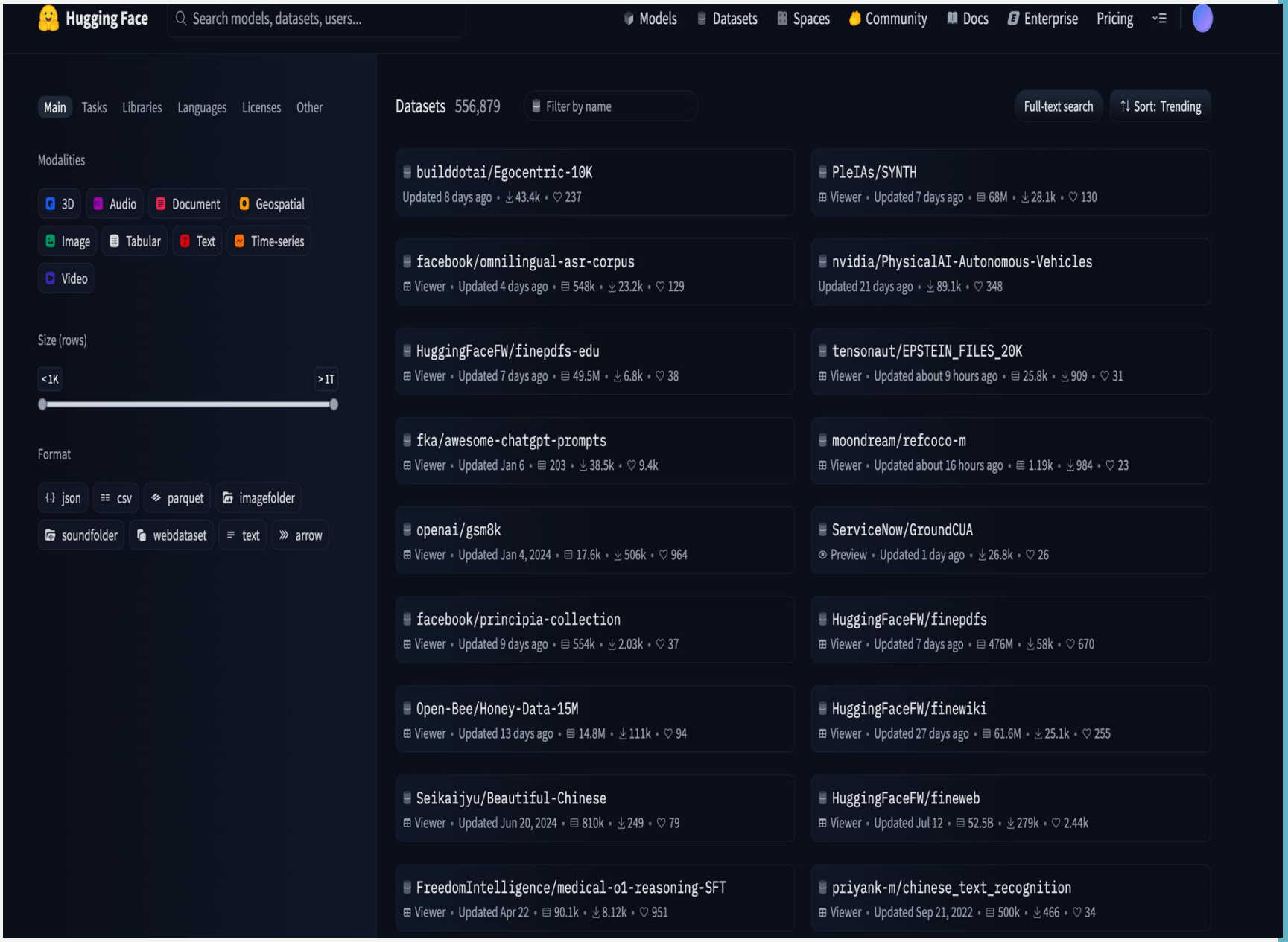
```python
# Thinking with Gemini 2.5
from google import genai
from google.genai import types

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)
```

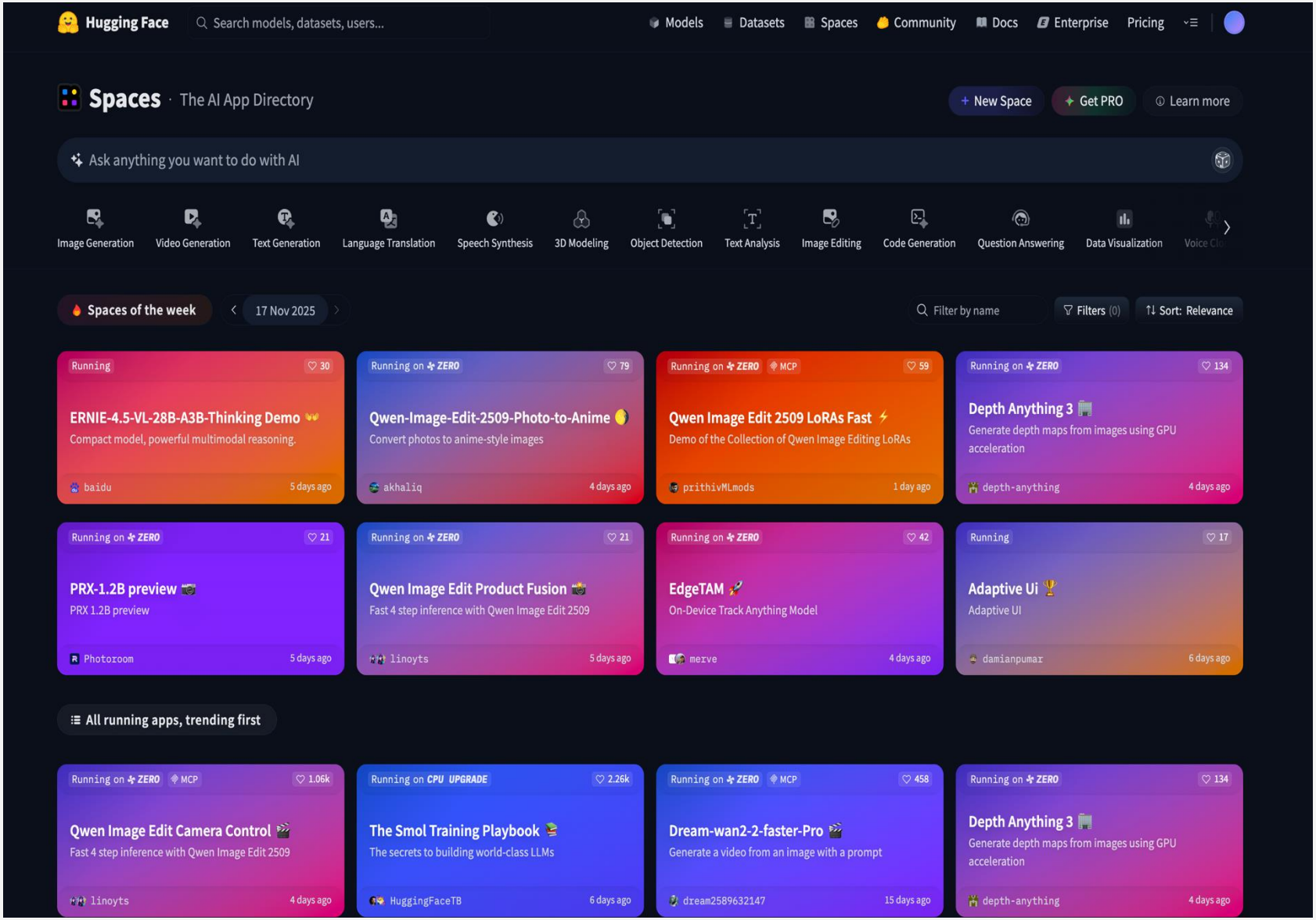
```python
#Working with Images ( Multi Model)
from PIL import Image
from google import genai

client = genai.Client()

image = Image.open("/path/to/organ.png")
response = client.models.generate_content(
    model="gemini-2.5-flash",
    contents=[image, "Tell me about this instrument"]
)
print(response.text)
```

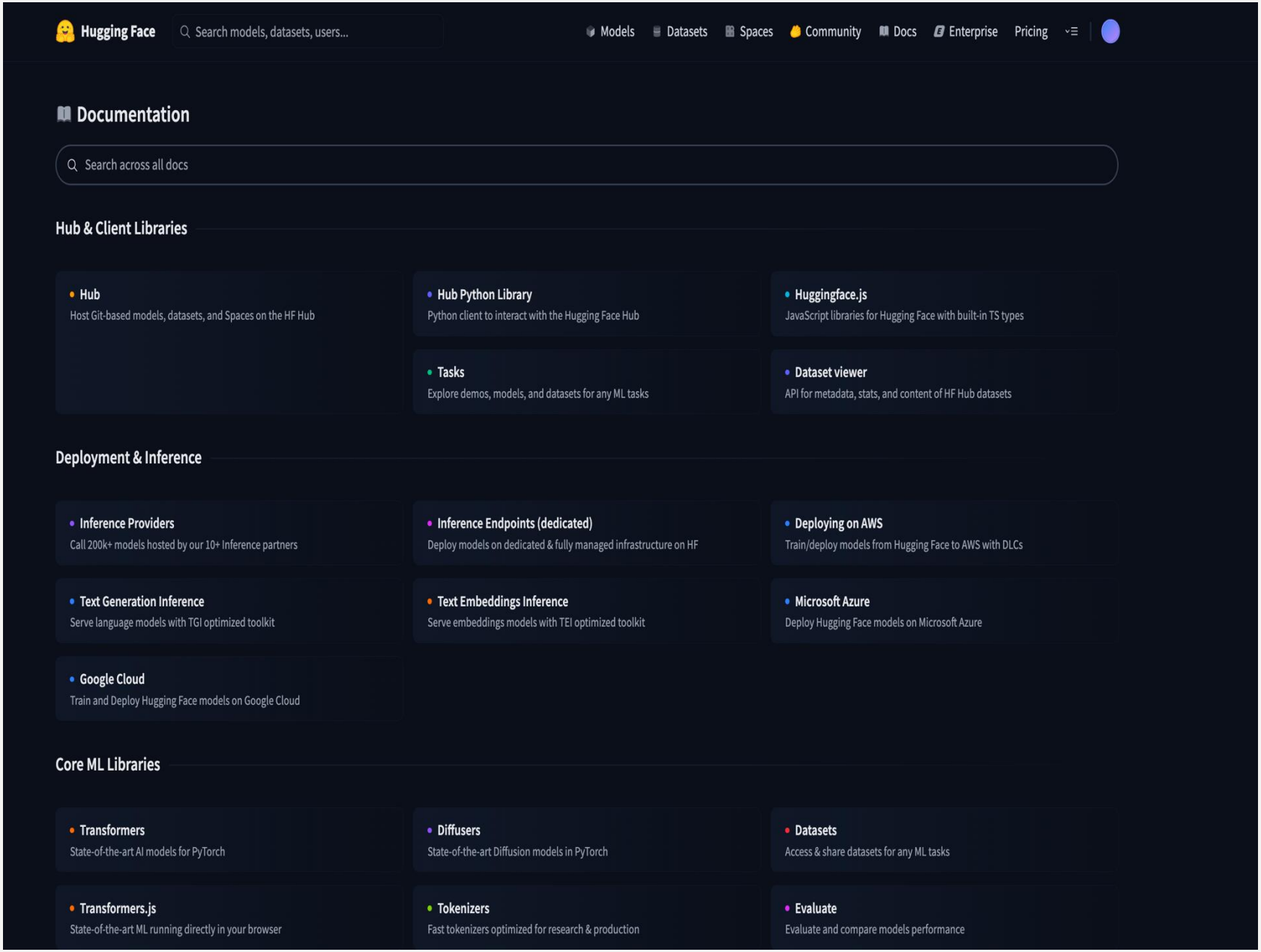
```python
import cohere
import os
from google.colab import userdata
os.environ["COHERE_API_KEY"] = userdata.get('COHERE_API_KEY')
co = cohere.ClientV2(os.environ["COHERE_API_KEY"])
response = co.chat(
    model="command-a-03-2025",
    messages=[{"role": "user", "content": "hello world!"}]
)

print(response.message)
```

HUGGINGFACE









Downloading and working with Tasks from HuggingFace

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

```
# Use a pipeline as a high-level helper
from transformers import pipeline

pipe = pipeline("text-generation", model="moonshotai/Kimi-K2-Thinking", trust_remote_code=True)
messages = [
    {"role": "user", "content": "Who are you?"},
]
pipe(messages)
```

WORKING WITH LANGCHAIN

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

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

```
# using init_chat
import os
from langchain.chat_models import init_chat_model
from google.colab import userdata
os.environ["OPENAI_API_KEY"] = userdata.get('OPENAI_API_KEY')
model = init_chat_model("gpt-4.1")
response = model.invoke("Why do birds fly")
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.