

Using AI Models from Python: Open Models, APIs, Safe Keys and a Tiny RAG

Run a ready-made open model, call a hosted model through an API, keep API keys safe, build prompts and validate replies in code, and build a tiny RAG in plain Python.

25 min read · Content last verified 20 October 2026 · Reviewed by Shravya M

Free awareness guide. Not legal or financial advice. <https://www.shravyadigital.com/guides/python-using-ai-models>

CONTENTS

1. Before you start
2. Chapter 1: Two ways to use an AI model
3. Chapter 2: Running a ready-made model
4. Chapter 3: Calling a hosted model: APIs
5. Chapter 4: Keeping API keys safe
6. Chapter 5: Building prompts and reading replies in code
7. Chapter 6: A tiny RAG in plain Python
8. Chapter 7: Responsible use, and where to go next
9. Key words

Before you start

This guide is the reading for **Course 5**, the last course of the Python for AI Pathway. It takes about 25 minutes and builds on Courses 1 to 4 and on the AI Foundations Pathway.

You'll learn the two ways to use AI models from Python: **running a ready-made model yourself**, and **calling a hosted model through an API**. You'll also learn how to keep API keys safe, how to build prompts and read replies in code, and how to build a tiny version of RAG in plain Python.

Examples marked `# not run` need an internet connection or an API key, so their output is described in words. Everything else you can run as it is.

All names and examples here are made up.

Chapter 1: Two ways to use an AI model

1. **Run a model yourself.** Download a ready-made, open model and run it in your notebook or on your computer. Your data stays with you, and there's no per-use charge, but big models need powerful hardware.
2. **Call a hosted model through an API.** Send your text over the internet to a company's model, such as an LLM, and get a reply. You get the most capable models without any hardware, but you pay per use, and your data goes to that company.

Most real projects use both: small open models for simple jobs, hosted models for hard ones.

Chapter 2: Running a ready-made model

Hugging Face hosts thousands of free, open models. Its `transformers` library runs one in a few lines. Colab already has it installed.

```
# not run: downloads a model from the internet the first time
from transformers import pipeline

classifier = pipeline("sentiment-analysis")
print(classifier("The delivery was quick and the product is excellent!"))
```

The first run downloads a small model, then prints its answer as a list with a **label** (here `POSITIVE`) and a **score** close to 1, the model's confidence. The same `pipeline` idea works for translation, summarising, answering questions and more: you change the task name and, optionally, choose a model.

This is the same create, use and check pattern as Course 4, except someone else has already done the training.

Chapter 3: Calling a hosted model: APIs

An **API** (application programming interface) is a way for one program to talk to another over the internet. To use a hosted model you:

1. Create an account with the provider and get an **API key**, a secret password for your code.
2. Send a **request**: your prompt plus settings, usually as **JSON**.
3. Receive a **response**: the model's reply, also as JSON.

The general shape looks like this (each provider documents its own address, fields and Python library, so follow their documentation):

```
# not run: needs a real provider address and API key
import os
import requests

response = requests.post(
    "https://api.example-ai-provider.com/v1/chat",
    headers={"Authorization": f"Bearer {os.environ['AI_API_KEY']}"},
    json={"model": "some-model", "messages": [{"role": "user", "content": "Explain UPI in
one line."}]},
    timeout=30,
)
print(response.json())
```

Things to know before you start:

- **You pay per token**, for both your input and the reply. Set a monthly spending limit in your account.
- **Data leaves your machine**. Don't send personal or confidential data unless you're allowed to.
- **Requests can fail**: network errors, rate limits, timeouts. Real code checks for errors and retries politely.

Chapter 4: Keeping API keys safe

An API key is like a debit card PIN for your account. If it leaks, anyone can run up your bill.

Never type a key directly into code you share, upload to GitHub or post in a forum. Bots scan public code for keys within minutes.

Keep it outside your code instead. In Colab, use the **Secrets** panel (the key icon on the left), then read it like this:

```
# not run: works only inside Colab, with a secret named AI_API_KEY
from google.colab import userdata

api_key = userdata.get("AI_API_KEY")
```

On your own computer, use an **environment variable**:

```
import os

api_key = os.environ.get("AI_API_KEY")
print("Key loaded" if api_key else "No key found: set AI_API_KEY first")
```

Output:

```
No key found: set AI_API_KEY first
```

If a key ever leaks, **revoke it** in the provider's dashboard and create a new one straight away.

The rule: Keys live in secrets or environment variables, never in the code itself.

Chapter 5: Building prompts and reading replies in code

The power of code is doing the same thing many times. Build a prompt from a template with an f-string:

```
template = "Classify this customer message as Complaint, Question or Praise. Reply with one word.\nMessage: {msg}"
messages = ["My refund still hasn't arrived.", "Do you deliver to Shillong?"]
for m in messages:
    prompt = template.format(msg=m)
    print(prompt.splitlines()[-1])
```

Output:

```
Message: My refund still hasn't arrived.
Message: Do you deliver to Shillong?
```

Asking the model to reply in **JSON** makes the answer easy to use in code. Then **check** it before trusting it, because models can return the wrong format or an answer outside your list:

```
import json

reply = '{"category": "Complaint", "urgent": true}'
data = json.loads(reply)
allowed = {"Complaint", "Question", "Praise"}
if data.get("category") in allowed:
    print("Category:", data["category"], "| urgent:", data["urgent"])
else:
    print("Unexpected answer, send to a person")
```

Output:

```
Category: Complaint | urgent: True
```

Always plan for the reply being wrong: validate it, and send anything unexpected to a person.

Chapter 6: A tiny RAG in plain Python

In the AI Foundations Pathway you learned **RAG**: find the relevant text first, then give it to the model. Here is the idea in a few lines of plain Python, using simple word-matching instead of embeddings:

```
docs = {
    "refunds": "Refunds reach your bank account within 7 working days of approval.",
    "delivery": "We deliver to all Indian pin codes, including the North East.",
    "payment": "We accept UPI, debit cards and net banking. We never ask for your UPI PIN.",
}

def retrieve(question):
    words = set(question.lower().replace("?", "").split())
    scores = {name: len(words & set(text.lower().split())) for name, text in docs.items()}
    return max(scores, key=scores.get)

question = "How many days for my refund to reach my bank?"
best = retrieve(question)
prompt = f"Answer using only this text:\n{docs[best]}\n\nQuestion: {question}"
print(best)
print(prompt)
```

Output:

```
refunds
Answer using only this text:
Refunds reach your bank account within 7 working days of approval.

Question: How many days for my refund to reach my bank?
```

You'd then send `prompt` to a model, as in Chapter 3. Real systems use embeddings and a vector database to match meaning rather than exact words, but the shape is exactly this: **retrieve, then generate**.

Chapter 7: Responsible use, and where to go next

- **Check outputs**, especially facts, numbers and anything about people.
- **Protect data**: don't send personal or confidential information to an API without permission.
- **Watch costs**: test on a few examples before running on thousands.
- **Respect licences**: open models come with licences; check what you're allowed to do.
- **Tell people** when they're talking to AI, or reading something it wrote.

Where to go next:

- Build a small project: summarise your own notes, sort your emails into folders, or analyse a spreadsheet and explain it in plain words.
- Learn more pandas and scikit-learn, then try a neural network course.
- Read your chosen provider's documentation and build a simple chatbot over your own documents.

Pass all five courses to earn the **Python for AI Pathway Certificate**.

Key words

- **Hugging Face / transformers / pipeline:** a home for open models, and the library that runs them.
- **API:** a way for programs to talk over the internet.
- **API key:** a secret that identifies and bills your account.
- **Request / response / JSON:** what you send, what you get back, and the text format both use.
- **Environment variable / Colab Secrets:** safe places to keep keys outside your code.
- **Prompt template:** a reusable prompt with blanks filled in by code.
- **Validation:** checking a model's reply before you use it.
- **RAG:** retrieve relevant text first, then ask the model to answer from it.

Take the course for what you just read

Take the free 10-question assessment and pass with 8 or more. Every course you pass counts towards a Pathway Certificate issued by Shravya Digital, earned when you pass all five courses in the pathway: <https://digital.shravyaworld.org/courses/python-using-ai-models>

Sources

Checked against these primary sources on 20 October 2026. Rules change: if something here looks out of date, trust the source.

- Hugging Face: transformers pipelines (https://huggingface.co/docs/transformers/main_classes/pipelines)
- Google Colab: secrets (FAQ) (<https://research.google.com/colaboratory/faq.html>)
- Python documentation: json (<https://docs.python.org/3/library/json.html>)
- Python documentation: os.environ (<https://docs.python.org/3/library/os.html#os.environ>)
- OWASP Top 10 for LLM Applications (<https://genai.owasp.org/llm-top-10/>)