CN7050- Intelligent Systems Module

Week-05 Tutorial (5 Marks)

Part I: A Multi Model Interactive Agent with Text and Image Generation capabilities.

Part II: Develop a Research Learning Assistant by using LLM

Please note that this is a **graded tutorial**. You are required to upload a **Microsoft Word (.docx)** or **PDF** file containing your output screenshots and written code to **Moodle** for marking. Additionally, include **comments explaining your understanding** of the tutorial.

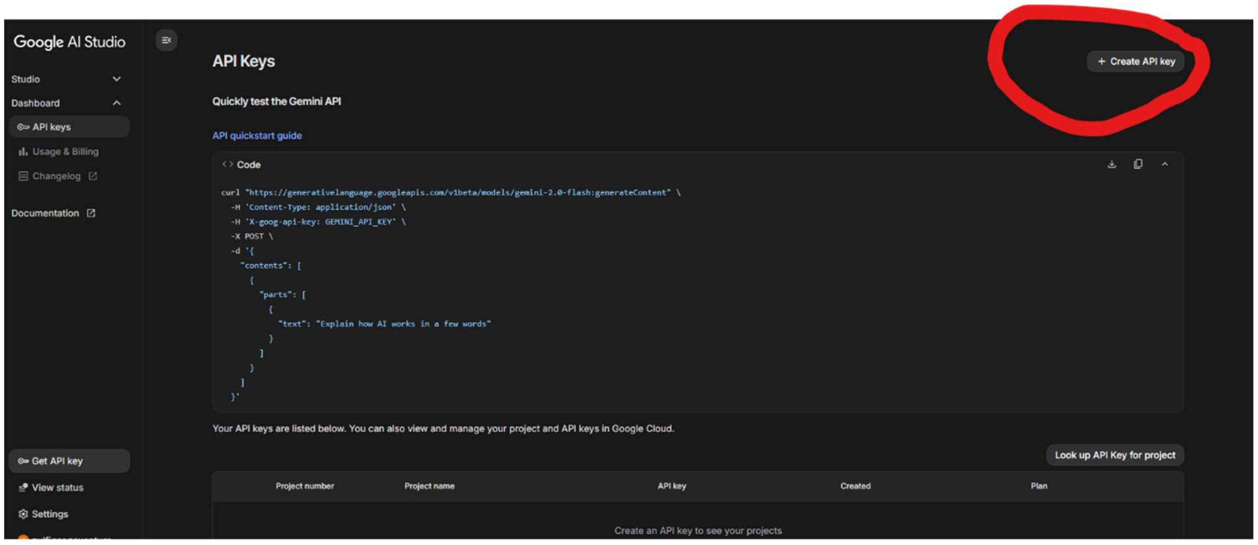
Part I: Learning Objectives

By the end of this lab, students will be able to:

- Understand the working of a multi model agent.
- Code to integrate multiple agents.
- Understands the different elements like LLM, Google Colab, AI Studio, Hugging face API and their integration.

Task 1: Account & API key

1. **Sign in to Google AI Studio (Gemini developer console).**
   Open: https://aistudio.google.com/ (Google AI Studio / Gemini developer area).



API Keys

Quickly test the Gemini API

API quickstart guide

Code

```
curl "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent" \
  -H 'Content-Type: application/json' \
  -H 'X-goog-api-key: GEMINI_API_KEY' \
  -X POST \
  -d '{
    "contents": [
      {
        "parts": [
          {
            "text": "Explain how AI works in a few words"
          }
        ]
      }
    ]
  }'
```

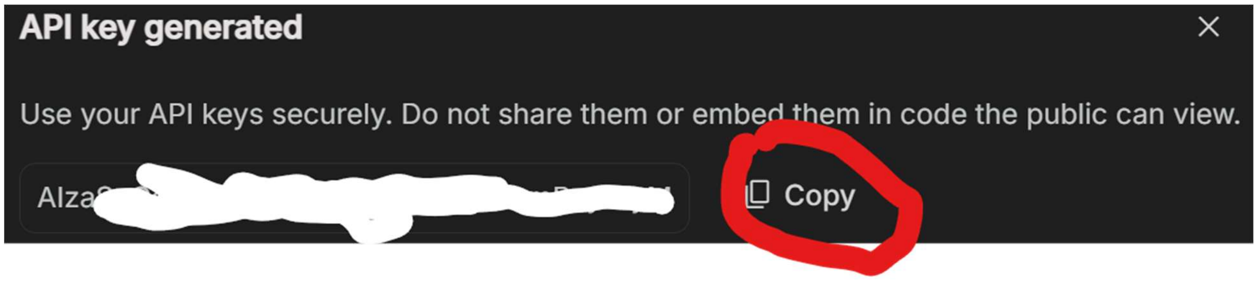
Your API keys are listed below. You can also view and manage your project and API keys in Google Cloud.

Project number	Project name	API key	Created	Plan
		Create an API key to see your projects		

2. **Create an API key (Gemini Developer API):**

   In AI Studio find API Keys / Get API Key (or Account → API Keys) and create one. For creating a new key, click the Create API Key button on top right. If you are new to Google AI Studio, a new project will automatically be created a new key will be generated; If, however, it is not your first time and you have generated the keys already, it will ask you to choose a project for key generation. You can choose a project in that case and then the new key will be generated in the selected project.

   Copy the key and store it temporarily, you will paste it into Colab for testing.



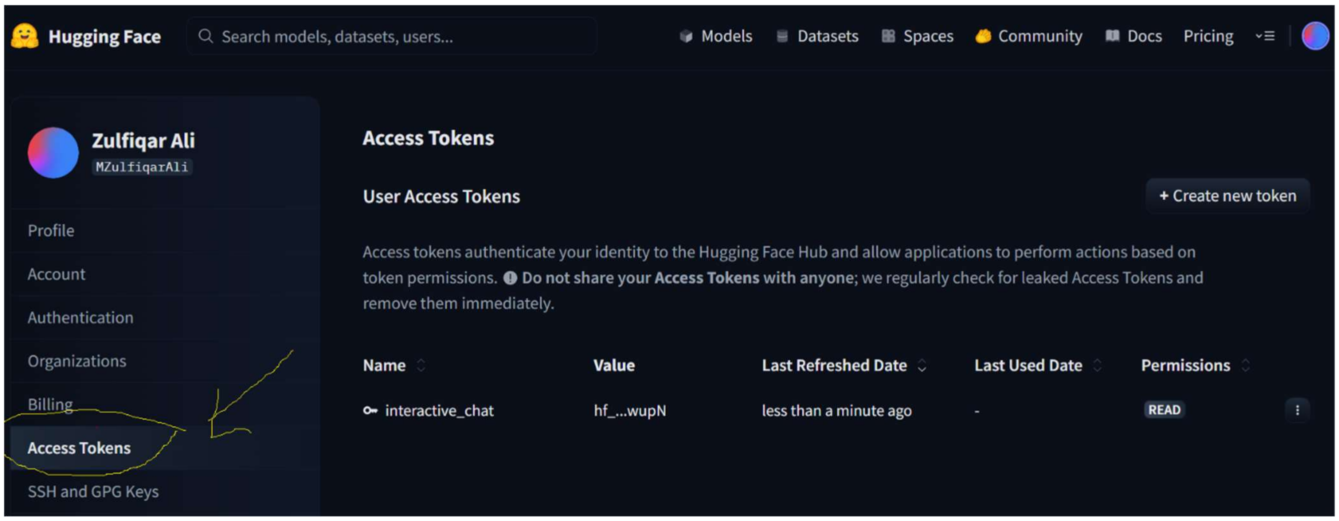
API key generated

Use your API keys securely. Do not share them or embed them in code the public can view.

AIza[redacted]

Note: Google publishes docs about rate limits / free tier. Check your API console for any usage limits (the free/dev tier and quotas can change). Read more about it at: https://ai.google.dev/gemini-api/docs/rate-limits

3. Create an access token by clicking the link highlighted in yellow in the following screenshot. Then click Create New Token button on right top. Select Read only and create a token. Once created, you will see the access token appearing on the right panel, copy the code once created and make a note of it for entering later on google colab.



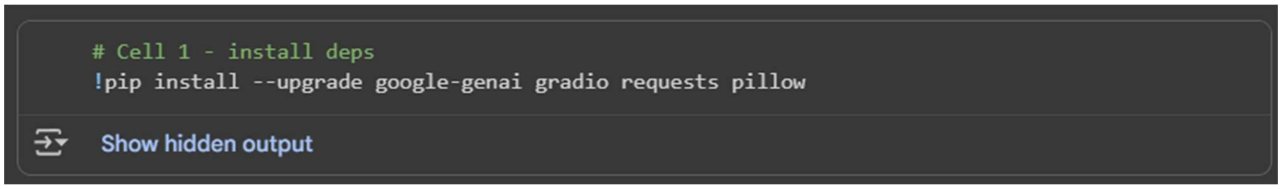
Access Tokens

User Access Tokens

Access tokens authenticate your identity to the Hugging Face Hub and allow applications to perform actions based on token permissions. **Do not share your Access Tokens with anyone**; we regularly check for leaked Access Tokens and remove them immediately.

Name	Value	Last Refreshed Date	Last Used Date	Permissions
interactive_chat	hf_...wupN	less than a minute ago	-	READ

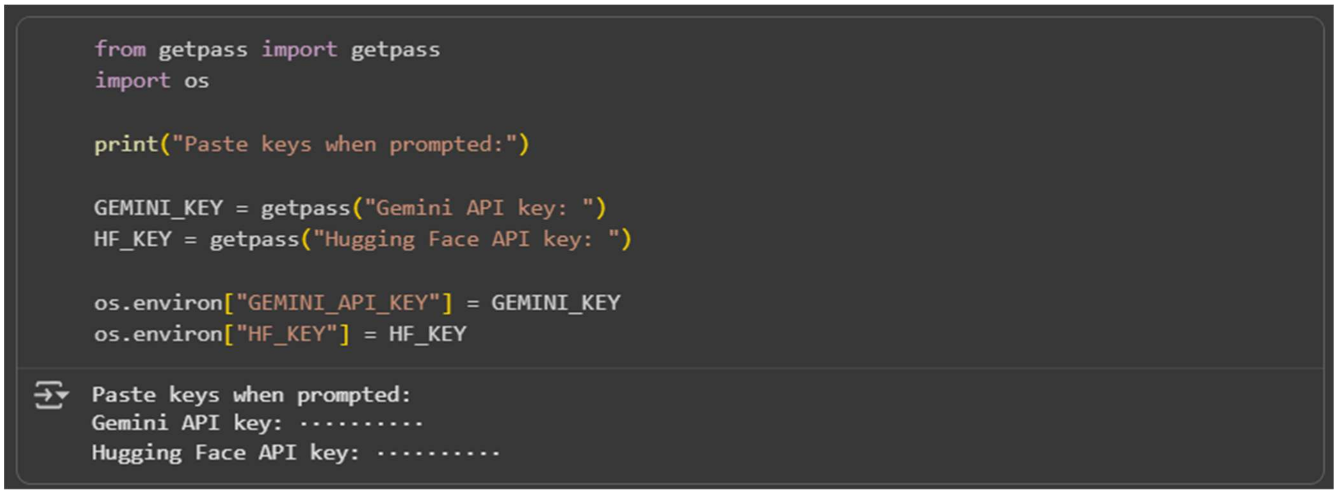
4. **Open a new Google Colab notebook.**
   Go to https://colab.research.google.com/ and create a new Python 3 notebook.
5. **Install the Google Gen AI SDK + helper libraries.**
   In a Colab cell write the following code:



```
# Cell 1 - install deps
!pip install --upgrade google-genai gradio requests pillow
```

google-genai is the official Python SDK for Gemini & Gen AI.

6. **Securely provide your API keys in Colab**
   In Colab use getpass so the key isn't visible in output, for that copy over the code in the below given screenshot:

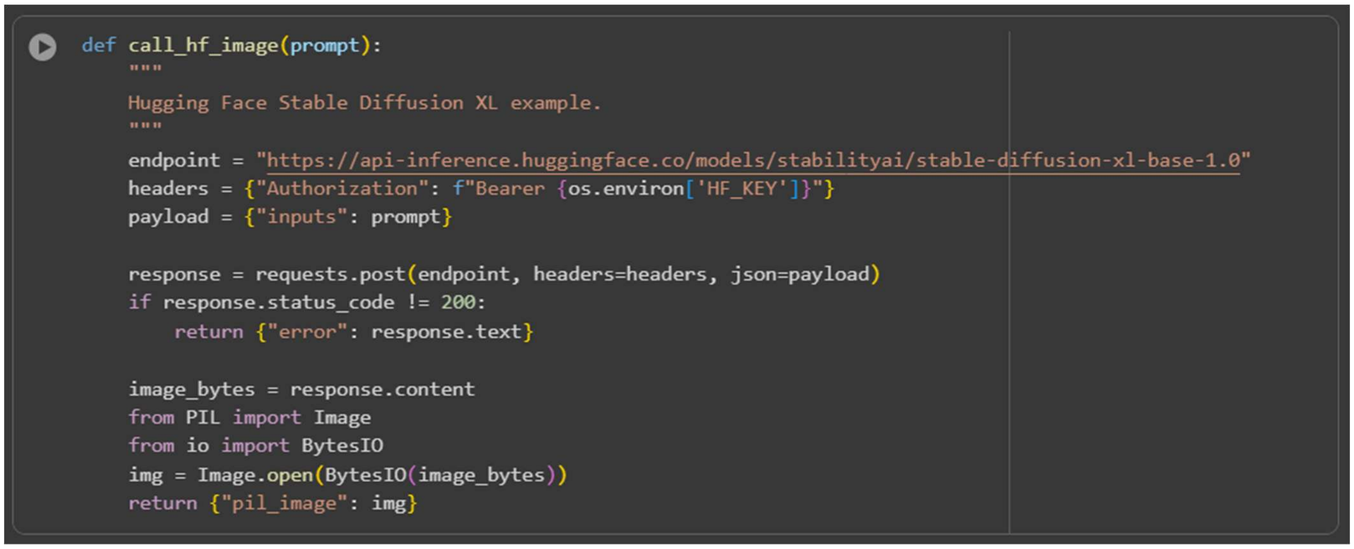


```
from getpass import getpass
import os

print("Paste keys when prompted:")

GEMINI_KEY = getpass("Gemini API key: ")
HF_KEY = getpass("Hugging Face API key: ")

os.environ["GEMINI_API_KEY"] = GEMINI_KEY
os.environ["HF_KEY"] = HF_KEY
```

```
Paste keys when prompted:
Gemini API key: ··········
Hugging Face API key: ··········
```

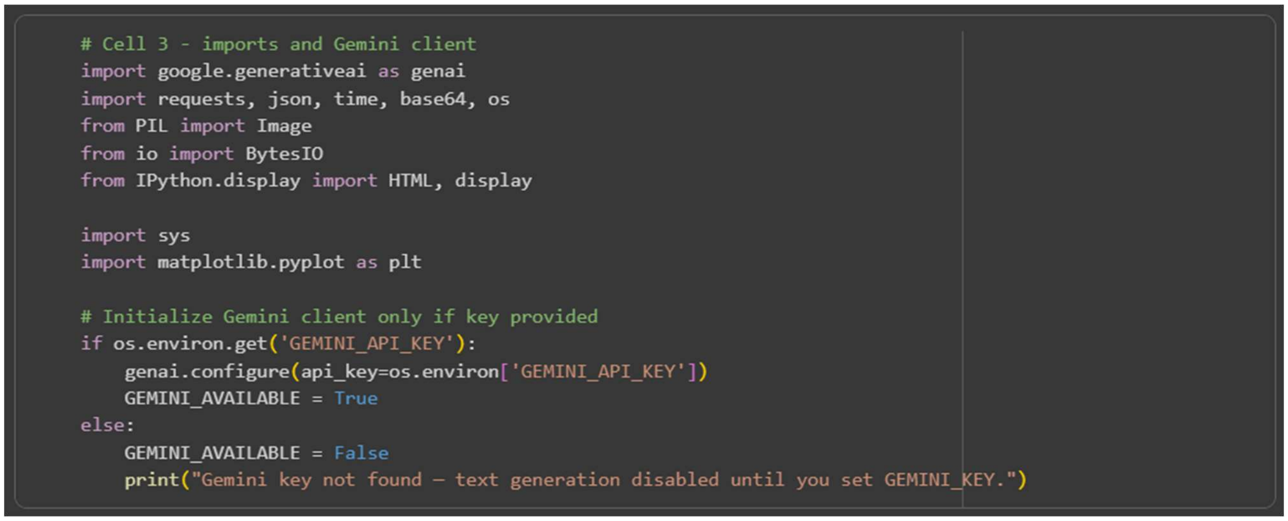
7. **Hugging Face Image Generation function.**

To generate the new image using the hugging face model, use the following code.



```python
def call_hf_image(prompt):
    """
    Hugging Face Stable Diffusion XL example.
    """
    endpoint = "https://api-inference.huggingface.co/models/stabilityai/stable-diffusion-xl-base-1.0"
    headers = {"Authorization": f"Bearer {os.environ['HF_KEY']}"}
    payload = {"inputs": prompt}

    response = requests.post(endpoint, headers=headers, json=payload)
    if response.status_code != 200:
        return {"error": response.text}

    image_bytes = response.content
    from PIL import Image
    from io import BytesIO
    img = Image.open(BytesIO(image_bytes))
    return {"pil_image": img}
```

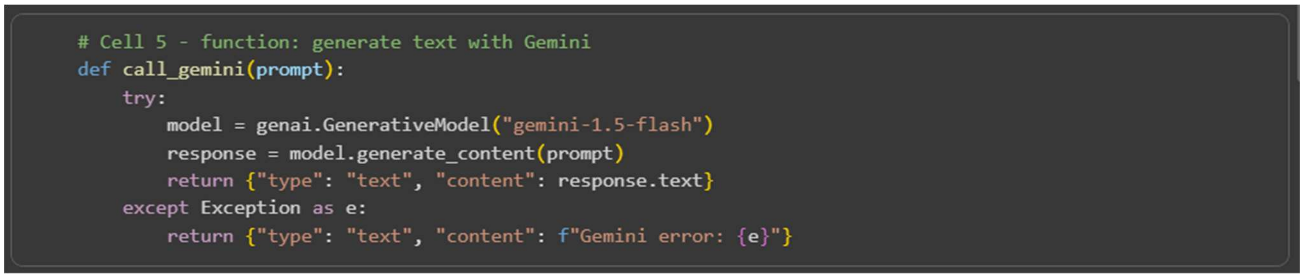
8. **Imports + Gemini client initialization.**
   Import the necessary libraries and initialize the Gemini client.



```python
# Cell 3 - imports and Gemini client
import google.generativeai as genai
import requests, json, time, base64, os
from PIL import Image
from io import BytesIO
from IPython.display import HTML, display

import sys
import matplotlib.pyplot as plt

# Initialize Gemini client only if key provided
if os.environ.get('GEMINI_API_KEY'):
    genai.configure(api_key=os.environ['GEMINI_API_KEY'])
    GEMINI_AVAILABLE = True
else:
    GEMINI_AVAILABLE = False
    print("Gemini key not found — text generation disabled until you set GEMINI_KEY.")
```

If Gemini key is not found, you will see the error message shown in the last code line.

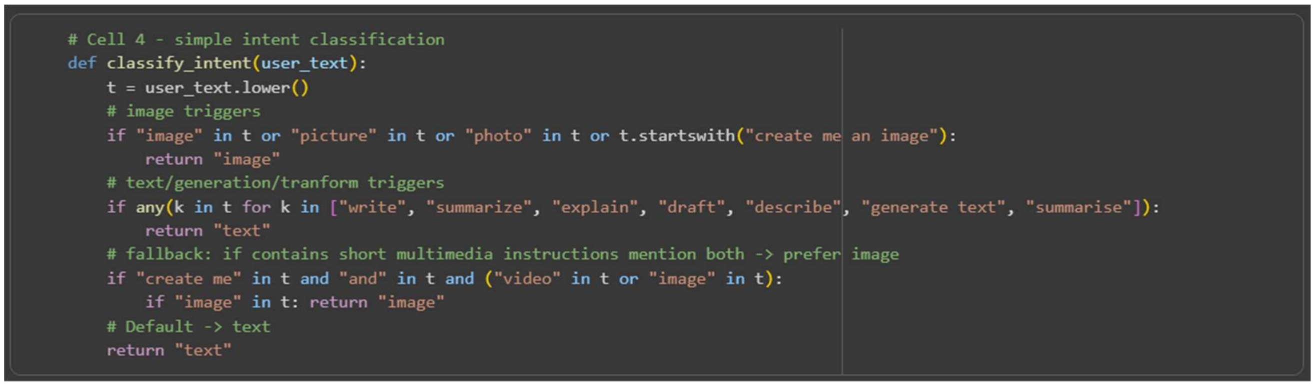
9. **Gemini text generation function.**
   Function to use and generate the text using Gemini LLM model.



```python
# Cell 5 - function: generate text with Gemini
def call_gemini(prompt):
    try:
        model = genai.GenerativeModel("gemini-1.5-flash")
        response = model.generate_content(prompt)
        return {"type": "text", "content": response.text}
    except Exception as e:
        return {"type": "text", "content": f"Gemini error: {e}"}
```

10. **Intent classifier (simple keyword routing)**

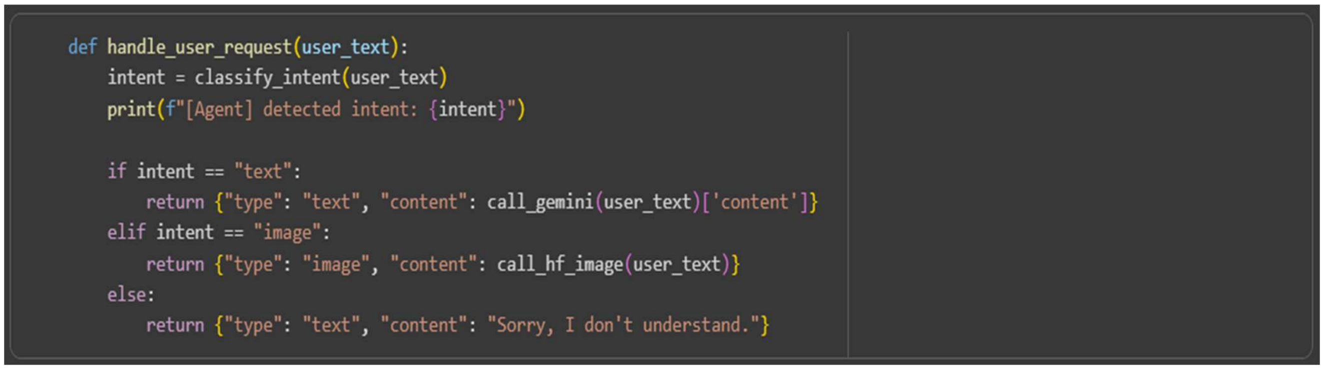
As we are creating a multi model agent, so we need to decide which model to use on the user's message. The following code will identify the user intent to use the image generation or text generation model.



```python
# Cell 4 - simple intent classification
def classify_intent(user_text):
    t = user_text.lower()
    # image triggers
    if "image" in t or "picture" in t or "photo" in t or t.startswith("create me an image"):
        return "image"
    # text/generation/tranform triggers
    if any(k in t for k in ["write", "summarize", "explain", "draft", "describe", "generate text", "summarise"]):
        return "text"
    # fallback: if contains short multimedia instructions mention both -> prefer image
    if "create me" in t and "and" in t and ("video" in t or "image" in t):
        if "image" in t: return "image"
    # Default -> text
    return "text"
```

11. Orchestrator: route user requests and call right API

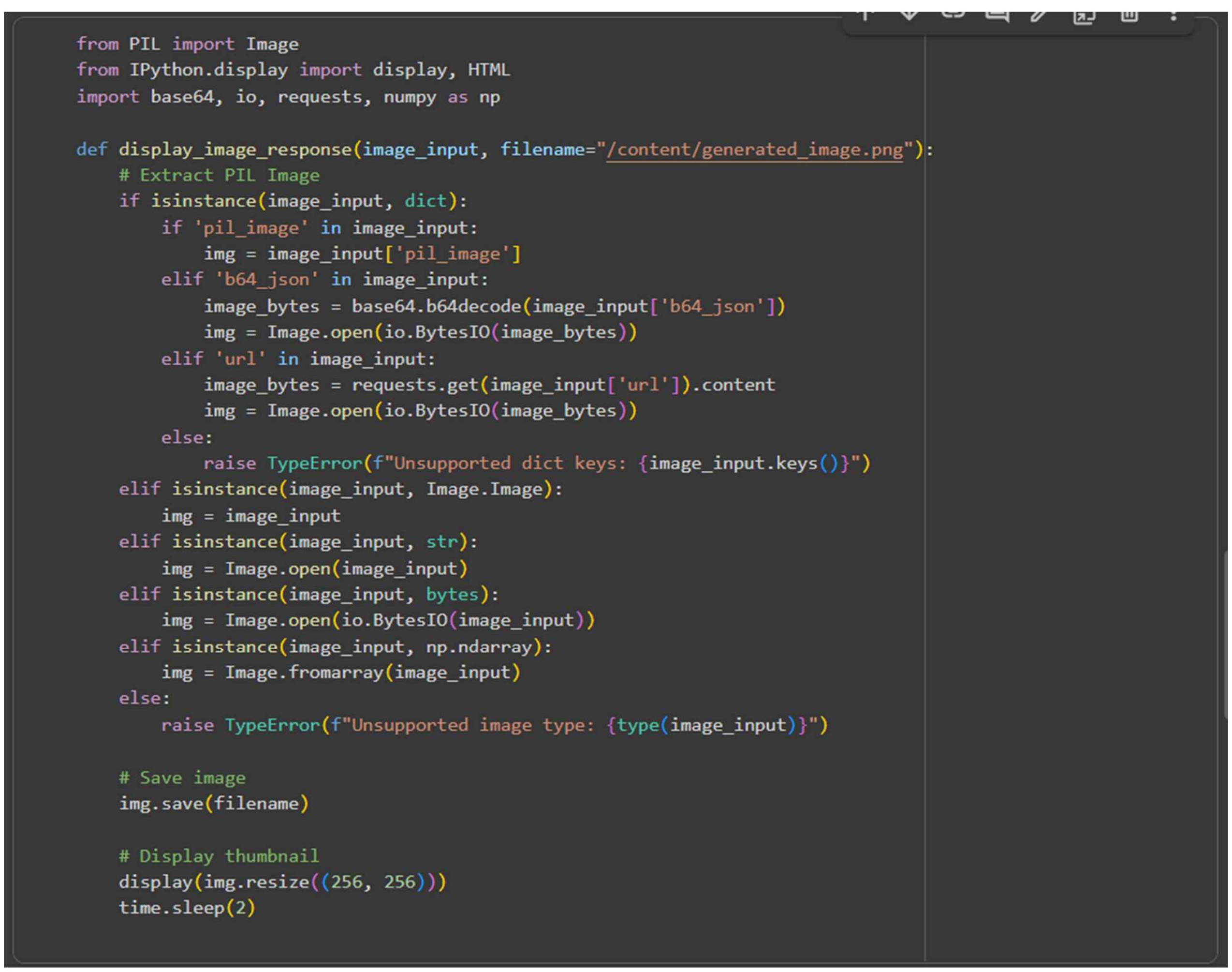
We will use the required function call in the following code to use the related model after identifying the intent in the above code.



```python
def handle_user_request(user_text):
    intent = classify_intent(user_text)
    print(f"[Agent] detected intent: {intent}")

    if intent == "text":
        return {"type": "text", "content": call_gemini(user_text)['content']}
    elif intent == "image":
        return {"type": "image", "content": call_hf_image(user_text)}
    else:
        return {"type": "text", "content": "Sorry, I don't understand."}
```

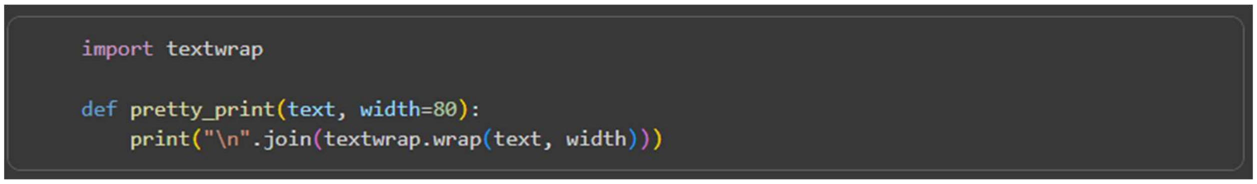
12. **Display image function**

The following code will be used to show the image generated by the hugging face model.



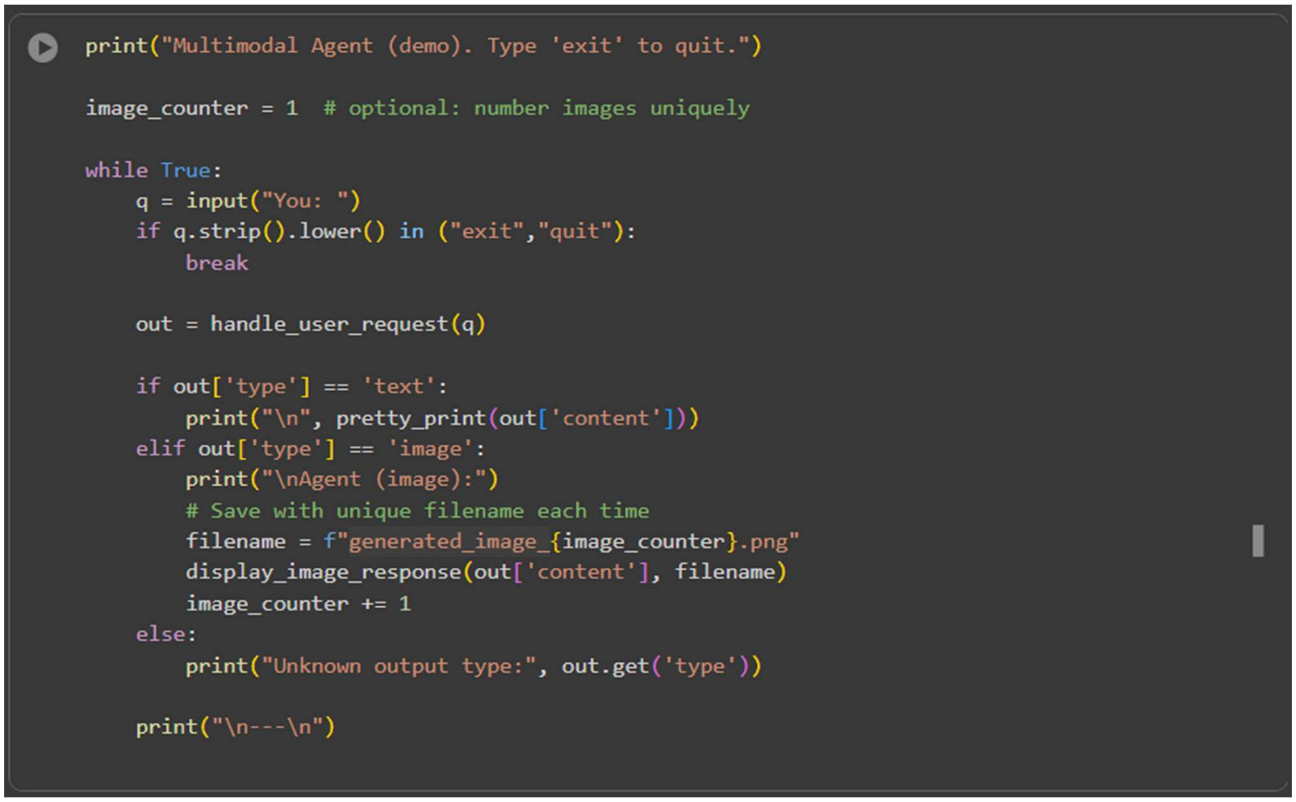
```python
from PIL import Image
from IPython.display import display, HTML
import base64, io, requests, numpy as np

def display_image_response(image_input, filename="/content/generated_image.png"):
    # Extract PIL Image
    if isinstance(image_input, dict):
        if 'pil_image' in image_input:
            img = image_input['pil_image']
        elif 'b64_json' in image_input:
            image_bytes = base64.b64decode(image_input['b64_json'])
            img = Image.open(io.BytesIO(image_bytes))
        elif 'url' in image_input:
            image_bytes = requests.get(image_input['url']).content
            img = Image.open(io.BytesIO(image_bytes))
        else:
            raise TypeError(f"Unsupported dict keys: {image_input.keys()}")
    elif isinstance(image_input, Image.Image):
        img = image_input
    elif isinstance(image_input, str):
        img = Image.open(image_input)
    elif isinstance(image_input, bytes):
        img = Image.open(io.BytesIO(image_input))
    elif isinstance(image_input, np.ndarray):
        img = Image.fromarray(image_input)
    else:
        raise TypeError(f"Unsupported image type: {type(image_input)}")

    # Save image
    img.save(filename)

    # Display thumbnail
    display(img.resize((256, 256)))
    time.sleep(2)
```

13. Function to show the text generated by the Gemini model in the wrapped form.



```python
import textwrap

def pretty_print(text, width=80):
    print("\n".join(textwrap.wrap(text, width)))
```

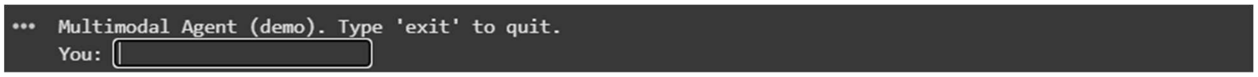
14. Code for interactive agent.



```python
print("Multimodal Agent (demo). Type 'exit' to quit.")

image_counter = 1  # optional: number images uniquely

while True:
    q = input("You: ")
    if q.strip().lower() in ("exit","quit"):
        break

    out = handle_user_request(q)

    if out['type'] == 'text':
        print("\n", pretty_print(out['content']))
    elif out['type'] == 'image':
        print("\nAgent (image):")
        # Save with unique filename each time
        filename = f"generated_image_{image_counter}.png"
        display_image_response(out['content'], filename)
        image_counter += 1
    else:
        print("Unknown output type:", out.get('type'))

    print("\n---\n")
```

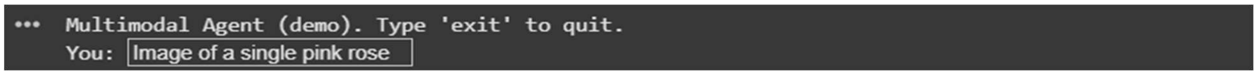
15. Output and result of the code.

When you will run the above code, you will see a prompt to generate image or text output. If you want to exit the prompt, you can enter exit and press enter. To generate the text, include prompt with keywords like “Image”, “photo” etc. for image generation and “describe”, “summarize” etc., as per step 10.



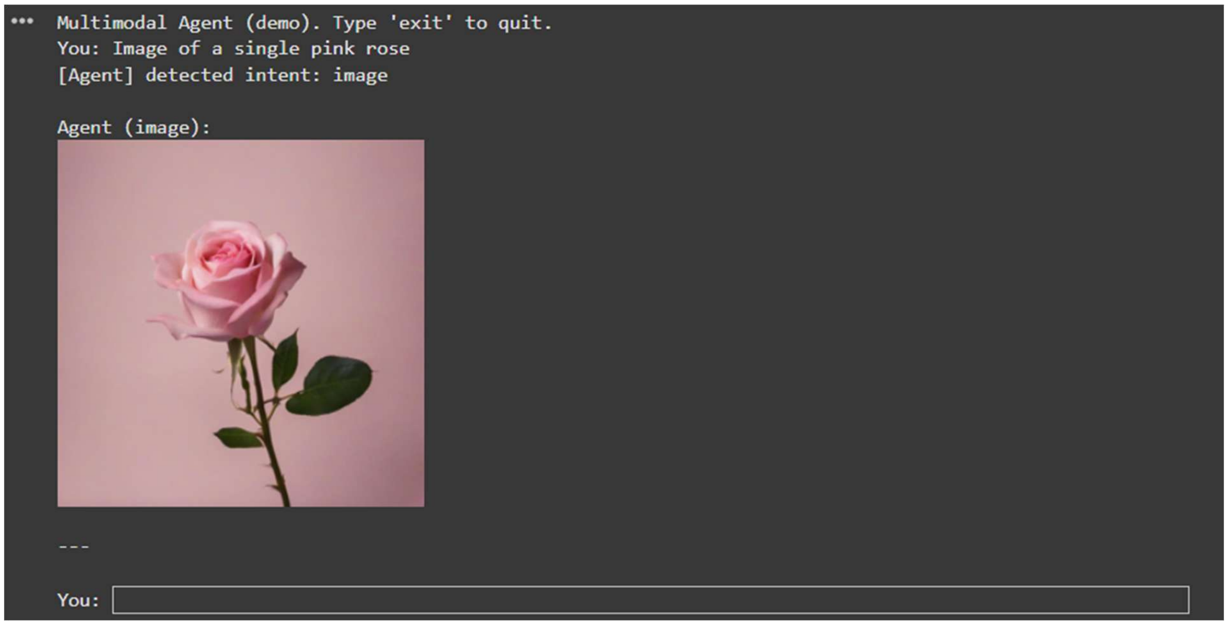
```
Multimodal Agent (demo). Type 'exit' to quit.
You:
```

Provide the prompt as provided in the following screenshot to generate a flower image.



```
Multimodal Agent (demo). Type 'exit' to quit.
You: Image of a single pink rose
```

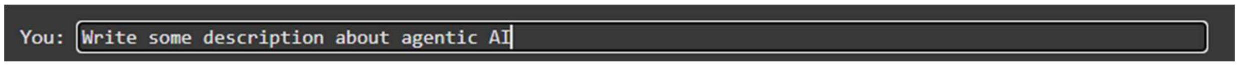
Once you will run this prompt, you will see a flower image. An example output is shown in the following image, but your output could be completely different.



```
Multimodal Agent (demo). Type 'exit' to quit.
You: Image of a single pink rose
[Agent] detected intent: image

Agent (image):
```

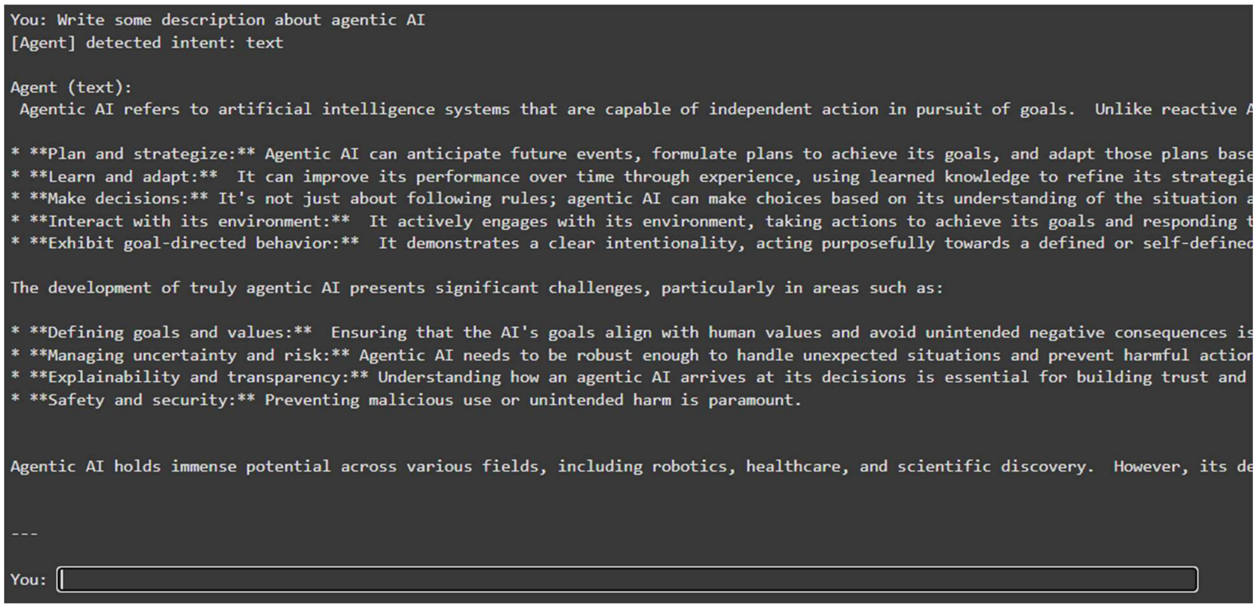
```
---

You:
```

Let’s generate some text by providing another prompt: “Write some description about agentic AI**”**



```
You: Write some description about agentic AI
```

Here is the output of the above prompt.



```
You: Write some description about agentic AI
[Agent] detected intent: text

Agent (text):
 Agentic AI refers to artificial intelligence systems that are capable of independent action in pursuit of goals.  Unlike reactive A

* **Plan and strategize:** Agentic AI can anticipate future events, formulate plans to achieve its goals, and adapt those plans base
* **Learn and adapt:**  It can improve its performance over time through experience, using learned knowledge to refine its strategie
* **Make decisions:** It's not just about following rules; agentic AI can make choices based on its understanding of the situation a
* **Interact with its environment:**  It actively engages with its environment, taking actions to achieve its goals and responding t
* **Exhibit goal-directed behavior:**  It demonstrates a clear intentionality, acting purposefully towards a defined or self-definec

The development of truly agentic AI presents significant challenges, particularly in areas such as:

* **Defining goals and values:**  Ensuring that the AI's goals align with human values and avoid unintended negative consequences is
* **Managing uncertainty and risk:** Agentic AI needs to be robust enough to handle unexpected situations and prevent harmful actior
* **Explainability and transparency:** Understanding how an agentic AI arrives at its decisions is essential for building trust and
* **Safety and security:** Preventing malicious use or unintended harm is paramount.


Agentic AI holds immense potential across various fields, including robotics, healthcare, and scientific discovery.  However, its de


---

You:
```

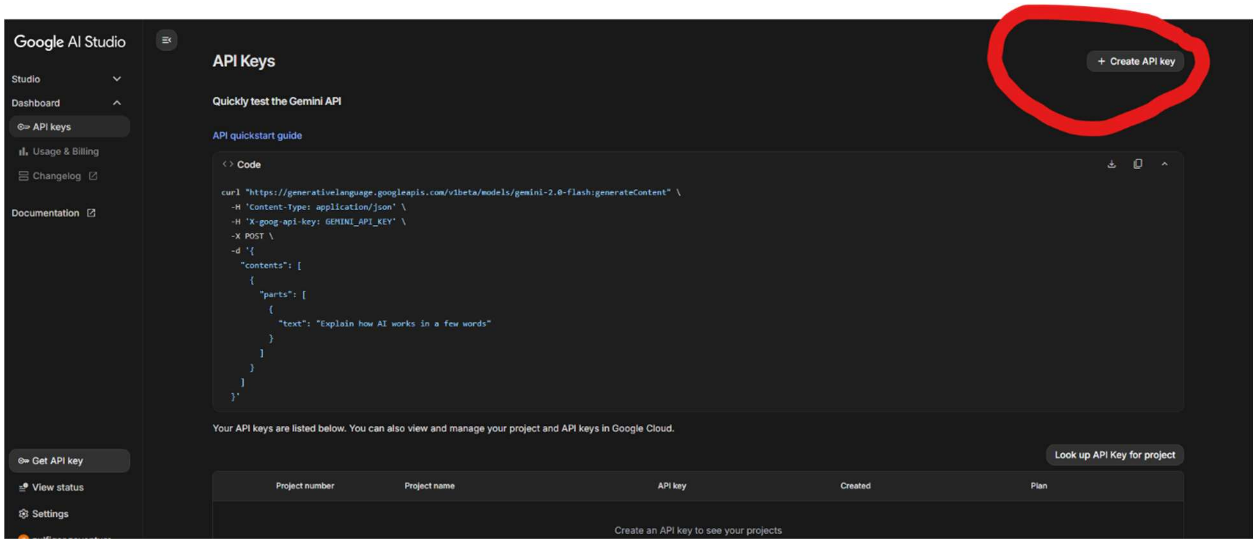
Part II: Learning Objectives

By the end of this lab, students will be able to:

- Understand the requirements of a LLM based agent.
- Code a research agent.
- Understands the different elements like LLM, google colab, AI Studio and their integration.

Task 1: Account & API key

1. **Sign in to Google AI Studio (Gemini developer console).**
   Open: https://aistudio.google.com/ (Google AI Studio / Gemini developer area).



Quickly test the Gemini API

```
curl "https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-flash:generateContent" \
  -H 'Content-Type: application/json' \
  -H 'X-goog-api-key: GEMINI_API_KEY' \
  -X POST \
  -d '{
    "contents": [
      {
        "parts": [
          {
            "text": "Explain how AI works in a few words"
          }
        ]
      }
    ]
  }'
```

Your API keys are listed below. You can also view and manage your project and API keys in Google Cloud.

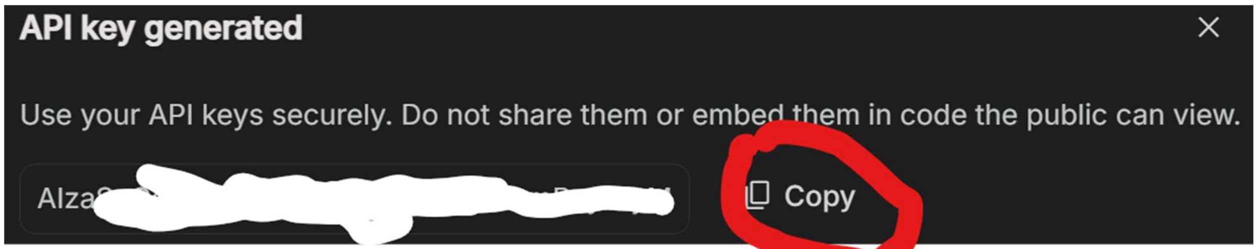
Project number	Project name	API key	Created	Plan

Create an API key to see your projects

2. **Create an API key (Gemini Developer API):**

   In AI Studio find API Keys / Get API Key (or Account → API Keys) and create one. For creating a new key, click the Create API Key button on top right. If you are new to Google AI Studio, a new project will automatically be created a new key will be generated; If, however, it is not your first time and you have generated the keys already, it will ask you to choose a project for key generation. You can choose a project in that case and then the new key will be generated in the selected project.

   Copy the key and store it temporarily, you will paste it into Colab for testing.

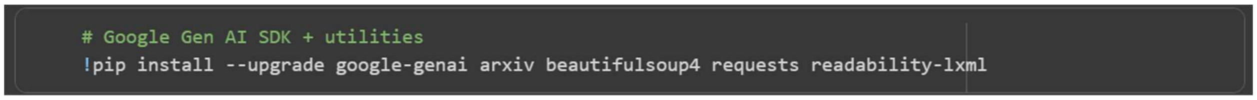

API key generated

Use your API keys securely. Do not share them or embed them in code the public can view.

AIza[redacted]

Note: Google publishes docs about rate limits / free tier. Check your API console for any usage limits (the free/dev tier and quotas can change). Read more about it at:
https://ai.google.dev/gemini-api/docs/rate-limits

3. **Open a new Google Colab notebook.**
   Go to https://colab.research.google.com/ and create a new Python 3 notebook.
4. **Install the Google Gen AI SDK + helper libraries.**
   In a Colab cell write the following code:

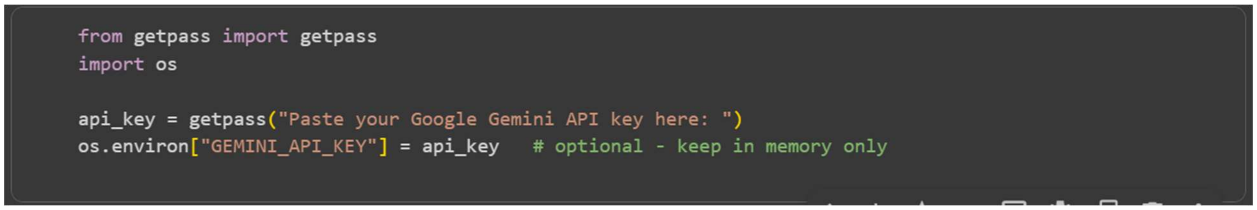


```
# Google Gen AI SDK + utilities
!pip install --upgrade google-genai arxiv beautifulsoup4 requests readability-lxml
```

google-genai is the official Python SDK for Gemini & Gen AI.

arxiv is a small Python wrapper for the arXiv API (free scholarly search).

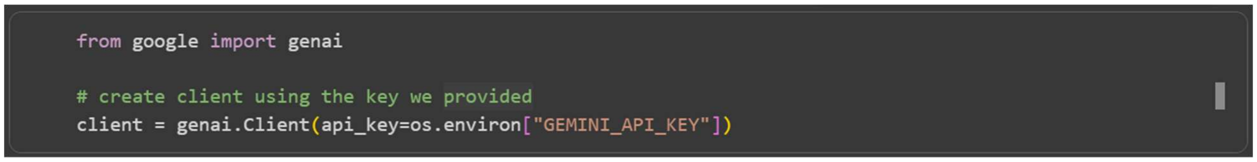
5. **Securely provide your API key in Colab**
   In Colab use getpass so the key isn't visible in output, for that copy over the code in the below given screenshot:



```
from getpass import getpass
import os

api_key = getpass("Paste your Google Gemini API key here: ")
os.environ["GEMINI_API_KEY"] = api_key   # optional - keep in memory only
```

6. **Initialize the Gen AI client in Colab.**

To initialize the Gen AI client, use the following code.

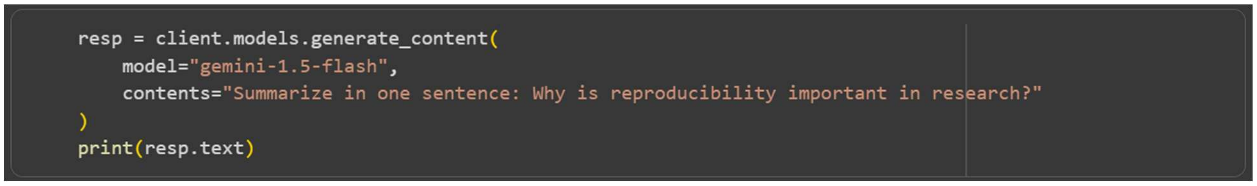


```python
from google import genai

# create client using the key we provided
client = genai.Client(api_key=os.environ["GEMINI_API_KEY"])
```

The SDK docs show the Client(api_key=...) usage and generation functions.

Quick "hello" test to confirm the API works

7. **Try a small generate call to confirm connectivity.**

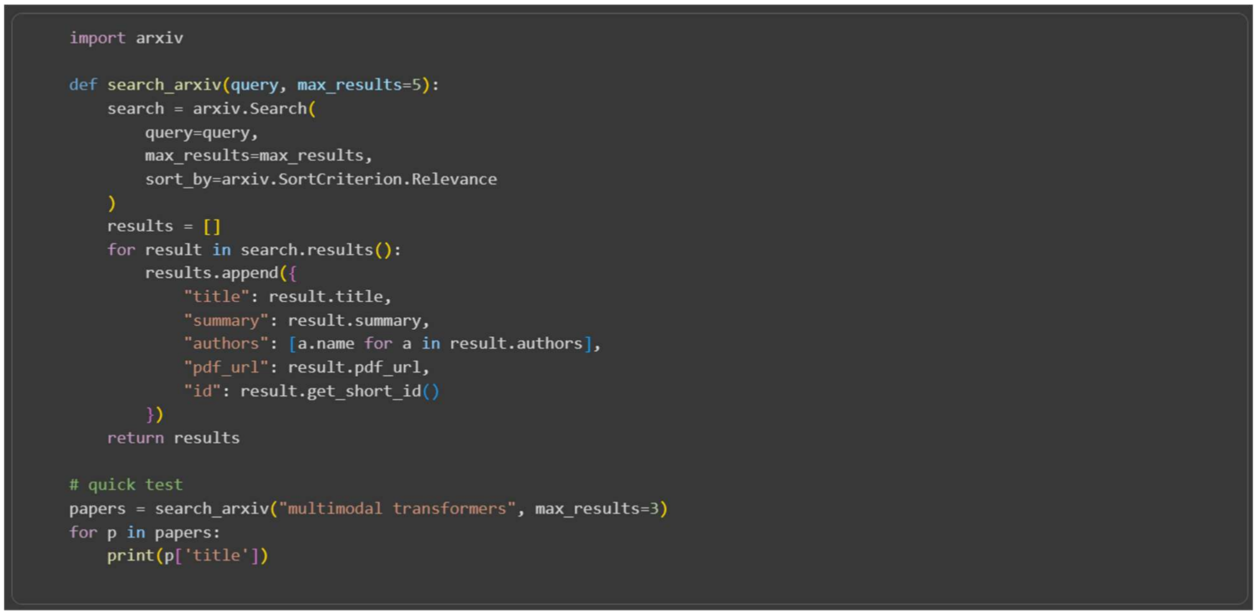


```python
resp = client.models.generate_content(
    model="gemini-1.5-flash",
    contents="Summarize in one sentence: Why is reproducibility important in research?"
)
print(resp.text)
```

If you get a reply text, the key is working. If you see an auth / permission error, re-check your key and AI Studio console.

Implementing search: arXiv (scholarly) + web snippets

8. **Search arXiv for scholarly papers.**
   Install/usage of the arxiv wrapper:



```python
import arxiv

def search_arxiv(query, max_results=5):
    search = arxiv.Search(
        query=query,
        max_results=max_results,
        sort_by=arxiv.SortCriterion.Relevance
    )
    results = []
    for result in search.results():
        results.append({
            "title": result.title,
            "summary": result.summary,
            "authors": [a.name for a in result.authors],
            "pdf_url": result.pdf_url,
            "id": result.get_short_id()
        })
    return results

# quick test
papers = search_arxiv("multimodal transformers", max_results=3)
for p in papers:
    print(p['title'])
```

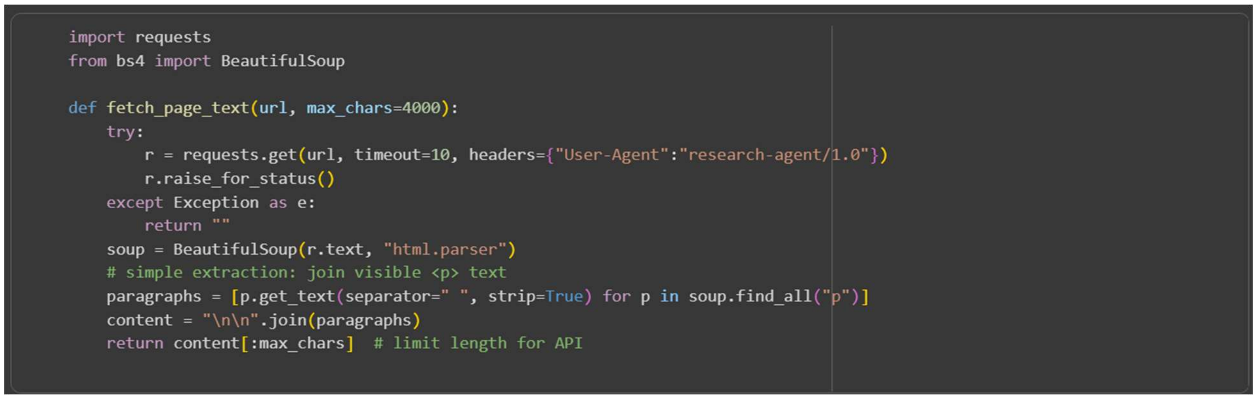
For arXiv API docs and python wrapper examples, follow these links;

https://info.arxiv.org/help/api/basics.html

https://github.com/lukasschwab/arxiv.py

9. **Grab web-page content for non-paper websites**

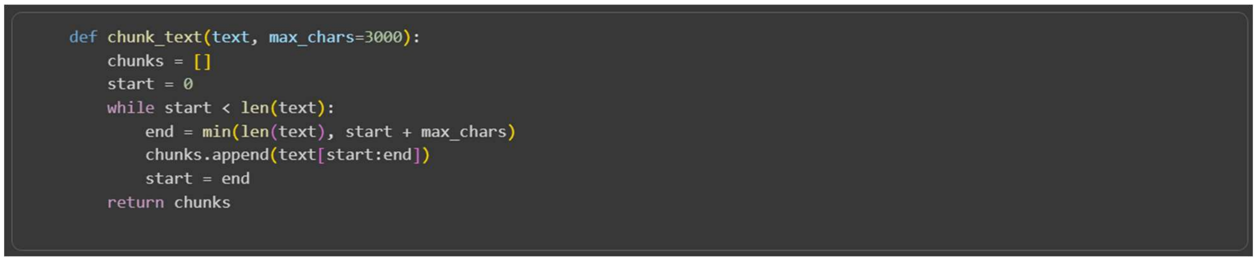
Use requests + BeautifulSoup to extract main paragraphs (simple heuristic):



```python
import requests
from bs4 import BeautifulSoup

def fetch_page_text(url, max_chars=4000):
    try:
        r = requests.get(url, timeout=10, headers={"User-Agent":"research-agent/1.0"})
        r.raise_for_status()
    except Exception as e:
        return ""
    soup = BeautifulSoup(r.text, "html.parser")
    # simple extraction: join visible <p> text
    paragraphs = [p.get_text(separator=" ", strip=True) for p in soup.find_all("p")]
    content = "\n\n".join(paragraphs)
    return content[:max_chars]  # limit length for API
```

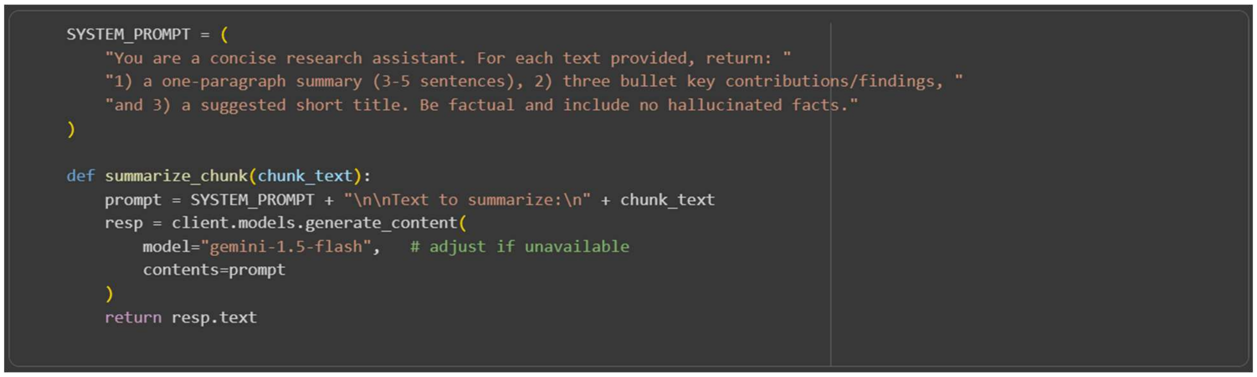
Summarization: chunking + calling Gemini

10. **Create a safe chunking helper** (Gemini has big context but chunking helps reliability):



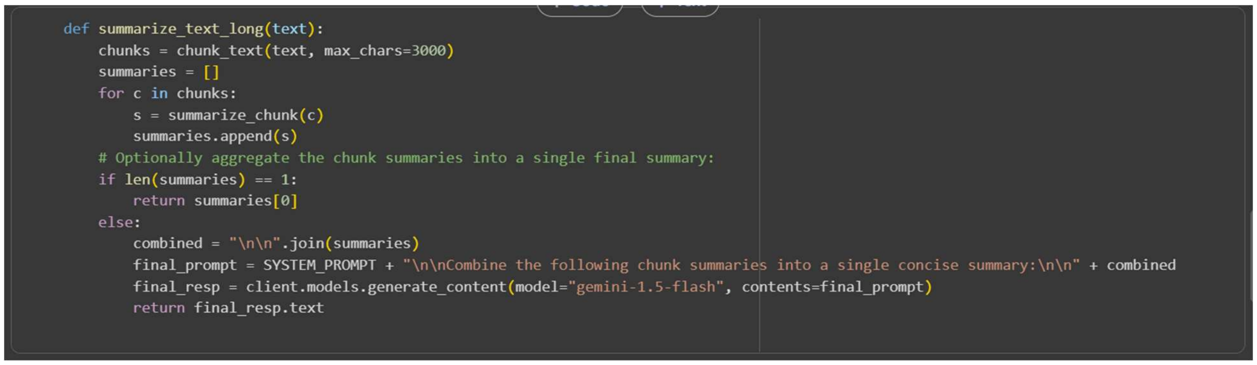
```python
def chunk_text(text, max_chars=3000):
    chunks = []
    start = 0
    while start < len(text):
        end = min(len(text), start + max_chars)
        chunks.append(text[start:end])
        start = end
    return chunks
```

11. **Create a function that asks Gemini to summarize a text chunk.**
Keep a short "system" instruction at the top to set the assistant behavior:



```python
SYSTEM_PROMPT = (
    "You are a concise research assistant. For each text provided, return: "
    "1) a one-paragraph summary (3-5 sentences), 2) three bullet key contributions/findings, "
    "and 3) a suggested short title. Be factual and include no hallucinated facts."
)

def summarize_chunk(chunk_text):
    prompt = SYSTEM_PROMPT + "\n\nText to summarize:\n" + chunk_text
    resp = client.models.generate_content(
        model="gemini-1.5-flash",   # adjust if unavailable
        contents=prompt
    )
    return resp.text
```

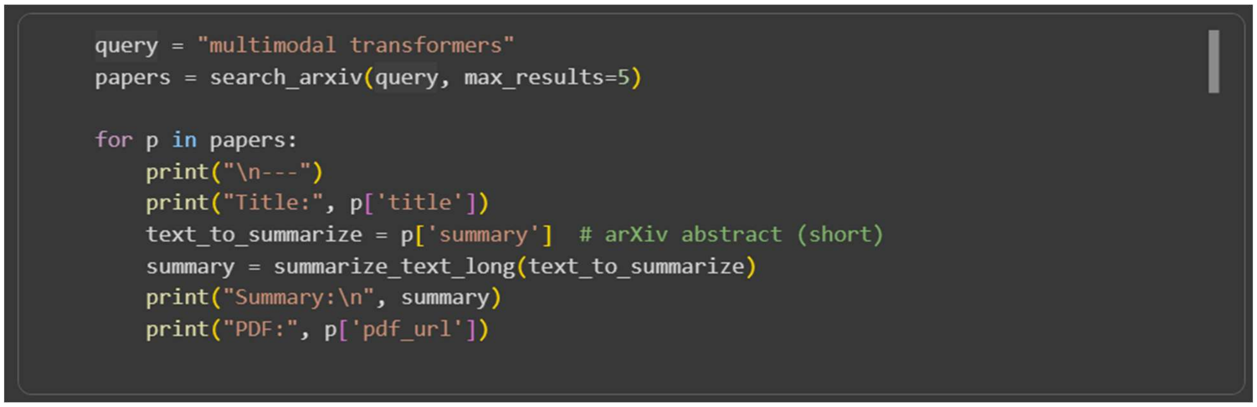
12. Summarize a long document by summarizing chunks + combining.



```python
def summarize_text_long(text):
    chunks = chunk_text(text, max_chars=3000)
    summaries = []
    for c in chunks:
        s = summarize_chunk(c)
        summaries.append(s)
    # Optionally aggregate the chunk summaries into a single final summary:
    if len(summaries) == 1:
        return summaries[0]
    else:
        combined = "\n\n".join(summaries)
        final_prompt = SYSTEM_PROMPT + "\n\nCombine the following chunk summaries into a single concise summary:\n\n" + combined
        final_resp = client.models.generate_content(model="gemini-1.5-flash", contents=final_prompt)
        return final_resp.text
```

Example: search arXiv + summarize abstracts

13. **Putting it together — search and summarize top N arXiv abstracts:**

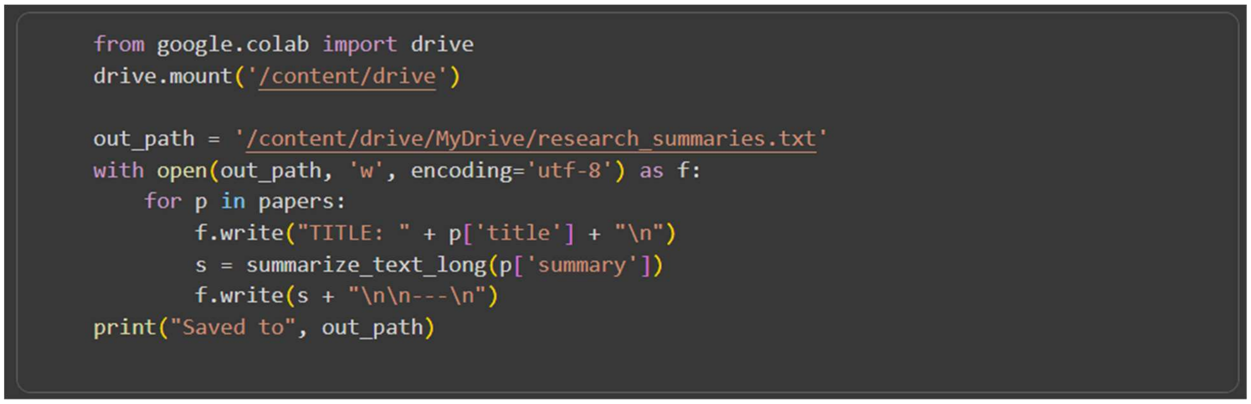


```python
query = "multimodal transformers"
papers = search_arxiv(query, max_results=5)

for p in papers:
    print("\n---")
    print("Title:", p['title'])
    text_to_summarize = p['summary']  # arXiv abstract (short)
    summary = summarize_text_long(text_to_summarize)
    print("Summary:\n", summary)
    print("PDF:", p['pdf_url'])
```

This gives you an immediate research-digest workflow: query -> fetch metadata -> summarize -> save.

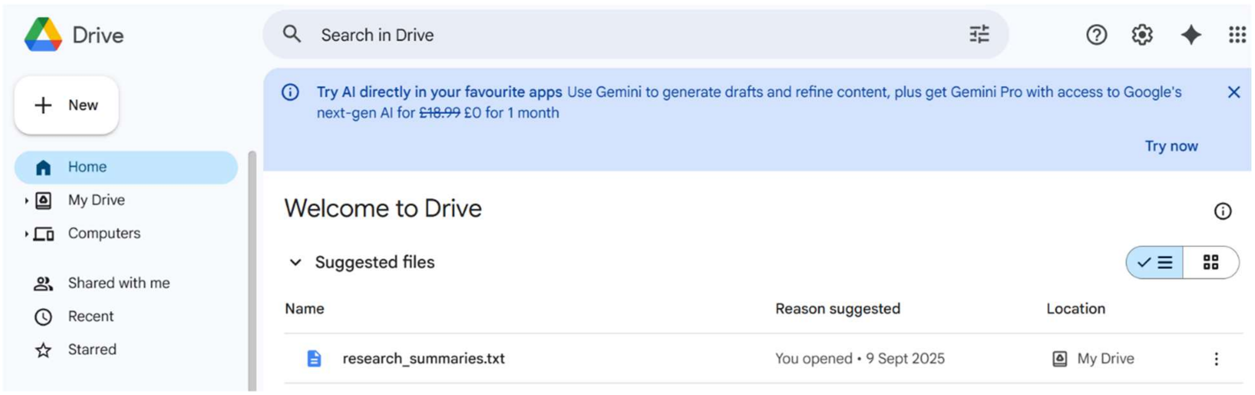
Save results (Drive)

14. **(Optional) Mount Google Drive and save summaries for later.**



```python
from google.colab import drive
drive.mount('/content/drive')

out_path = '/content/drive/MyDrive/research_summaries.txt'
with open(out_path, 'w', encoding='utf-8') as f:
    for p in papers:
        f.write("TITLE: " + p['title'] + "\n")
        s = summarize_text_long(p['summary'])
        f.write(s + "\n\n---\n")
print("Saved to", out_path)
```

You will be asked to give permissions to access the google drive on executing the above code. One the code successfully runs, a file with our provided name research_summaries.txt will be created on drive as shown in the snapshot below.



Try AI directly in your favourite apps Use Gemini to generate drafts and refine content, plus get Gemini Pro with access to Google's next-gen AI for ~~£18.99~~ £0 for 1 month

Welcome to Drive

Suggested files

Name	Reason suggested	Location
research_summaries.txt	You opened • 9 Sept 2025	My Drive

How to get help

- Please speak with your Practical Group tutor if you have any questions during the practical session.
- If you have any questions about your studies, not relating to this course, please contact your Year Tutor