```
import os
from dotenv import load_dotenv
from openai import OpenAI
import anthropic
from IPython.display import Markdown, display, update_display
import google.generativeai
```

Thư viện	Chức năng
os	Cung cấp các chức năng làm việc với hệ điều hành, như quản lý tệp và biến môi trường.
dotenv.load_dotenv	Tải các biến môi trường từ tệp .env, giúp bảo mật thông tin nhạy cảm.
openai.OpenAI	Tương tác với API của OpenAI để tạo văn bản, hình ảnh, hoặc thực hiện các tác vụ AI khác.
anthropic	Thư viện của Anthropic dùng để làm việc với mô hình AI của họ, như Claude.
IPython.display.Markdown	Hiển thị nội dung Markdown trong Jupyter Notebook.
IPython.display.display	Hiển thị nội dung trong Jupyter Notebook.
IPython.display.update_display	Cập nhật nội dung hiển thị trong Jupyter Notebook.
google.generativeai	Làm việc với mô hình AI tạo sinh của Google (Gemini AI).

```
load_dotenv(override=True)
openai_api_key = os.getenv('OPENAI_API_KEY')
anthropic_api_key = os.getenv('ANTHROPIC_API_KEY')
google_api_key = os.getenv('GOOGLE_API_KEY')

if openai_api_key:
    print(f"OpenAI API Key exists and begins {openai_api_key[:8]}")
else:
    print("OpenAI API Key not set")

if anthropic_api_key:
    print(f"Anthropic API Key exists and begins
{anthropic_api_key[:7]}")
else:
    print("Anthropic API Key not set")

if google_api_key:
    print(f"Google API Key exists and begins {google_api_key[:8]}")
else:
    print("Google API Key not set")
```

Dòng lệnh	Chức năng
load_dotenv(override=True)	Tải biến môi trường từ tệp .env, ghi đè nếu đã tồn tại.
openai_api_key = os.getenv('OPENAI_API_KEY')	Lấy giá trị API key của OpenAI từ biến môi trường.
anthropic_api_key = os.getenv('ANTHROPIC_API_KEY')	Lấy giá trị API key của Anthropic từ biến môi trường.

google_api_key = os.getenv('GOOGLE_API_KEY')	Lấy giá trị API key của Google từ biến môi trường.
if openai_api_key: print(f"OpenAI API Key exists and begins {openai_api_key[:8]}")	Kiểm tra xem API key của OpenAI có tồn tại không và in ra 8 ký tự đầu tiên nếu có.
else: print("OpenAI API Key not set")	Thông báo nếu API key của OpenAI không được thiết lập.
if anthropic_api_key: print(f"Anthropic API Key exists and begins {anthropic_api_key[:7]}")	Kiểm tra và in ra 7 ký tự đầu tiên của API key Anthropic nếu có.
else: print("Anthropic API Key not set")	Thông báo nếu API key của Anthropic không được thiết lập.
if google_api_key: print(f"Google API Key exists and begins {google_api_key[:8]}")	Kiểm tra và in ra 8 ký tự đầu tiên của API key Google nếu có.
else: print("Google API Key not set")	Thông báo nếu API key của Google không được thiết lập.

```
openai = OpenAI()

claude = anthropic.Anthropic()
google.generativeai.configure()
```

Dòng lệnh	Chức năng
openai = OpenAI()	Khởi tạo đối tượng OpenAI để tương tác với API của OpenAI.
claude = anthropic.Anthropic()	Khởi tạo đối tượng Anthropic để làm việc với mô hình Claude.
google.generativeai.configure()	Cấu hình thư viện Generative AI của Google (Gemini AI).

```
system_message = "You are an assistant that is great at telling jokes"
user_prompt = "Tell a light-hearted joke for an audience of Data Scientists"

prompts = [
  {"role": "system", "content": system_message},
  {"role": "user", "content": user_prompt}
 ]
```

Dòng lệnh	Chức năng
system_message = "You are an assistant that is great at telling jokes"	Thiết lập thông điệp hệ thống để hướng dẫn AI đóng vai trò là một trợ lý chuyên kể chuyện cười.
user_prompt = "Tell a light-hearted joke for an audience of Data Scientists"	Thiết lập lời nhắc của người dùng để yêu cầu AI kể một câu chuyện cười dành cho các nhà khoa học dữ liệu.

prompts = [{"role": "system", "content": system_message}, {"role": "user", "content": user_prompt}]	Tạo danh sách lời nhắc (prompt) với vai trò hệ thống và người dùng để gửi đến AI.

```
completion = openai.chat.completions.create(model='gpt-3.5-turbo', messages=prompts)
print(completion.choices[0].message.content)
```

Dòng lệnh	Chức năng
completion = openai.chat.completions.create(model='gpt-3.5-turbo', messages=prompts)	Gửi danh sách lời nhắc (prompts) đến mô hình gpt-3.5-turbo của OpenAI để nhận phản hồi.
print(completion.choices[0].message.content)	In nội dung phản hồi đầu tiên từ AI ra màn hình.

```
completion = openai.chat.completions.create(
    model='gpt-4o-mini',
    messages=prompts,
    temperature=0.7
)
print(completion.choices[0].message.content)

completion = openai.chat.completions.create(
    model='gpt-4o',
    messages=prompts,
    temperature=0.4
)
print(completion.choices[0].message.content)
```

Dòng lệnh	Chức năng
completion = openai.chat.completions.create(model='gpt-4o-mini', messages=prompts, temperature=0.7)	Tạo một yêu cầu hoàn thành (completion) với mô hình gpt-4o-mini, truyền đầu vào là prompts, và thiết lập temperature là 0.7 để điều chỉnh độ sáng tạo của mô hình.
print(completion.choices[0].message.content)	In ra nội dung của phản hồi đầu tiên từ mô hình hoàn thành (completion).
completion = openai.chat.completions.create(model='gpt-4o', messages=prompts, temperature=0.4)	Tạo một yêu cầu hoàn thành (completion) với mô hình gpt-4o, truyền đầu vào là prompts, và thiết lập temperature là 0.4 để giảm độ sáng tạo của mô hình.
print(completion.choices[0].message.content)	In ra nội dung của phản hồi đầu tiên từ mô hình hoàn thành (completion).

```
message = claude.messages.create(
    model="claude-3-5-sonnet-latest",
    max_tokens=200,
    temperature=0.7,
    system=system_message,
    messages=[
        {"role": "user", "content": user_prompt},
    ],
)

print(message.content[0].text)

result = claude.messages.stream(
    model="claude-3-5-sonnet-latest",
    max_tokens=200,
    temperature=0.7,
    system=system_message,
    messages=[
        {"role": "user", "content": user_prompt},
    ],
)

with result as stream:
    for text in stream.text_stream:
            print(text, end="", flush=True)
```

Dòng lệnh	Chức năng
message = claude.messages.create(model= "claude-3-5-sonnet-latest", max_tokens=200, temperature=0.7, system=system_message, messages=[{"role": "user", "content": user_prompt}])	Tạo một yêu cầu hoàn thành (completion) với mô hình claude-3-5-sonnet-latest, thiết lập tối đa 200 token và nhiệt độ temperature là 0.7, sử dụng thông điệp hệ thống và yêu cầu từ người dùng (user_prompt).
print(message.content[0].text)	In ra nội dung của phản hồi từ Claude, lấy phần đầu tiên trong danh sách content.
result = claude.messages.stream(model ="claude-3-5-sonnet-latest", max_tokens=200, temperature=0.7, system=system_message, messages=[{"role": "user", "content": user_prompt}])	Tạo một yêu cầu stream hoàn thành với mô hình claude-3-5-sonnet-latest, thiết lập tối đa 200 token và nhiệt độ temperature là 0.7, sử dụng thông điệp hệ thống và yêu cầu từ người dùng.
with result as stream: for text in stream.text_stream: print(text, end="", flush=True)	Duyệt qua từng phần văn bản trong luồng (stream.text_stream) và in ra nội dung từng phần khi nhận được từ Claude.

```
gemini = google.generativeai.GenerativeModel(
    model_name='gemini-2.0-flash-exp',
    system_instruction=system_message
)
response = gemini.generate_content(user_prompt)
print(response.text)

gemini_via_openai_client = OpenAI(
    api_key=google_api_key,
    base_url="https://generativelanguage.googleapis.com/v1beta/openai/"
)

response = gemini_via_openai_client.chat.completions.create(
    model="gemini-2.0-flash-exp",
    messages=prompts
)
print(response.choices[0].message.content)
```

Dòng lệnh	Chức năng
gemini = google.generativeai.GenerativeModel(model_name='gemini-2.0-flash-exp', system_instruction=system_message)	Khởi tạo mô hình generative AI từ Google với tên mô hình là gemini-2.0-flash-exp và thiết lập thông điệp hệ thống.
response = gemini.generate_content(user_prompt)	Tạo nội dung dựa trên yêu cầu từ người dùng (user_prompt) bằng mô hình Gemini.
print(response.text)	In ra nội dung trả về từ mô hình Gemini.
gemini_via_openai_client = OpenAI(api_key=google_api_key, base_url="https://generativelanguage.googleapis.com/v1beta/openai/")	Khởi tạo client OpenAI với khóa API của Google và URL cơ sở của API Generative Language.
response = gemini_via_openai_client.chat.completions.create(model="gemini-2.0-flash-exp", messages=prompts)	Gửi yêu cầu tạo hoàn thành (completion) cho mô hình Gemini thông qua client OpenAI, với các thông điệp (prompts) đã cung cấp.
print(response.choices[0].message.content)	In ra nội dung của phản hồi đầu tiên từ mô hình Gemini thông qua client OpenAI.

```
stream = deepseek_via_openai_client.chat.completions.create(
    model="deepseek-chat",
    messages=challenge,
    stream=True
)

reply = ""
display_handle = display(Markdown(""), display_id=True)
for chunk in stream:
    reply += chunk.choices[0].delta.content or ''
    reply = reply.replace("```","").replace("markdown","")
    update_display(Markdown(reply), display_id=display_handle.display_id)

print("Number of words:", len(reply.split(" ")))
```

Dòng lệnh	Chức năng
stream = deepseek_via_openai_client.chat.completions.create(model="deepseek-chat", messages=challenge, stream=True)	Gửi yêu cầu tạo hoàn thành (completion) cho mô hình deepseek-chat với các thông điệp (challenge), bật chế độ stream để nhận kết quả từng phần.
reply = ""	Khởi tạo biến reply để lưu trữ nội dung trả về từ stream.
display_handle = display(Markdown(""), display_id=True)	Hiển thị một đối tượng Markdown trống và tạo display_id để cập nhật nội dung trong quá trình stream.
for chunk in stream:	Duyệt qua từng phần (chunk) dữ liệu nhận được từ stream.
reply += chunk.choices[0].delta.content or "	Thêm nội dung nhận được từ mỗi phần dữ liệu vào biến reply.
reply = reply.replace("```","").replace("markdown","")	Xử lý chuỗi reply bằng cách loại bỏ các phần tử không cần thiết như markdown và ``` từ nội dung trả về.
update_display(Markdown(reply), display_id=display_handle.display_id)	Cập nhật nội dung Markdown đã xử lý lên giao diện người dùng trong quá trình stream.
print("Number of words:", len(reply.split(" ")))	In ra số từ trong nội dung reply sau khi hoàn thành stream.

```
deepseek_api_key = os.getenv('DEEPSEEK_API_KEY')

if deepseek_api_key:
    print(f"DeepSeek API Key exists and begins {deepseek_api_key[:3]}")
else:
    print("DeepSeek API Key not set - please skip to the next section if
you don't wish to try the DeepSeek API")
```

Dòng lệnh	Chức năng
deepseek_api_key = os.getenv('DEEPSEEK_API_KEY')	Lấy giá trị của biến môi trường DEEPSEEK_API_KEY từ hệ thống.
if deepseek_api_key:	Kiểm tra xem khóa API DEEPSEEK_API_KEY có tồn tại (không rỗng) trong biến môi trường.
print(f"DeepSeek API Key exists and begins {deepseek_api_key[:3]}")	Nếu khóa API tồn tại, in ra thông báo xác nhận và hiển thị ba ký tự đầu tiên của khóa API.
else:	Nếu khóa API không tồn tại, thực thi phần sau của khối điều kiện.

```
deepseek_via_openai_client = OpenAI(
    api_key=deepseek_api_key,
    base_url="https://api.deepseek.com"
)

response = deepseek_via_openai_client.chat.completions.create(
    model="deepseek-chat",
    messages=prompts,
)

print(response.choices[0].message.content)
```

Dòng lệnh	Chức năng
deepseek_via_openai_client = OpenAI(api_key=deepseek_api_key, base_url="https://api.deepseek.com")	Khởi tạo client OpenAI với khóa API deepseek_api_key và URL cơ sở của API DeepSeek (https://api.deepseek.com).
response = deepseek_via_openai_client.chat.completions.create(model="deepseek-chat", messages=prompts)	Gửi yêu cầu tạo hoàn thành (completion) cho mô hình deepseek-chat với các thông điệp (prompts).
print(response.choices[0].message.content)	In ra nội dung của phản hồi đầu tiên từ mô hình deepseek-chat.

```
challenge = [{"role": "system", "content": "You are a helpful
assistant"},
             {"role": "user", "content": "How many words are there in
your answer to this prompt"}]
```

Dòng lệnh	Chức năng
challenge = [{"role": "system", "content": "You are a helpful assistant"}, {"role": "user", "content": "How many words are there in your answer to this prompt"}]	Khởi tạo biến challenge dưới dạng một danh sách chứa hai từ điển: một từ điển cho vai trò system với nội dung mô tả trợ lý, và một từ điển cho vai trò user với câu hỏi yêu cầu đếm số từ trong câu trả lời của trợ lý.

```
response = deepseek_via_openai_client.chat.completions.create(
    model="deepseek-reasoner",
    messages=challenge
)

reasoning_content = response.choices[0].message.reasoning_content
content = response.choices[0].message.content

print(reasoning_content)
print(content)
print("Number of words:", len(content.split(" ")))
```

response = deepseek_via_openai_client.chat.completions.create(model="deepseek-reasoner", messages=challenge)	Gửi yêu cầu tạo hoàn thành (completion) cho mô hình deepseek-reasoner với các thông điệp trong biến challenge.
reasoning_content = response.choices[0].message.reasoning_content	Lấy nội dung lý luận (reasoning_content) từ phản hồi của mô hình, nếu có.
content = response.choices[0].message.content	Lấy nội dung câu trả lời (content) từ phản hồi của mô hình.
print(reasoning_content)	In ra nội dung lý luận (reasoning_content).
print(content)	In ra nội dung câu trả lời (content).
print("Number of words:", len(content.split(" ")))	In ra số từ trong nội dung câu trả lời (content) bằng cách chia chuỗi thành các từ và đếm số lượng từ.

```
response = deepseek_via_openai_client.chat.completions.create(
    model="deepseek-reasoner",
    messages=challenge
)

stream = openai.chat.completions.create(
    model='gpt-4o',
    messages=prompts,
    temperature=0.7,
    stream=True
)

reply = ""
display_handle = display(Markdown(""), display_id=True)
for chunk in stream:
    reply += chunk.choices[0].delta.content or ''
    reply = reply.replace("```","").replace("markdown","")
    update_display(Markdown(reply), display_id=display_handle.display_id)
reasoning_content = response.choices[0].message.reasoning_content
content = response.choices[0].message.content

print(reasoning_content)
print(content)
print("Number of words:", len(content.split(" ")))
```

Dòng lệnh	Chức năng
prompts = [{"role": "system", "content": "You are a helpful assistant that responds in Markdown"}, {"role": "user", "content": "How do I decide if a business problem is suitable for an LLM solution? Please respond in Markdown."}]	Khởi tạo biến prompts với hai thông điệp: một thông điệp hệ thống mô tả trợ lý sẽ phản hồi bằng Markdown, và một câu hỏi từ người dùng yêu cầu trợ lý trả lời về việc xác định tính phù hợp của một vấn đề kinh doanh cho giải pháp LLM.
stream = openai.chat.completions.create(model='gpt-4o', messages=prompts, temperature=0.7, stream=True)	Gửi yêu cầu tạo hoàn thành (completion) cho mô hình gpt-4o, với các thông điệp trong prompts, thiết lập nhiệt độ temperature là 0.7, và bật chế độ stream để nhận kết quả từng phần.
reply = ""	Khởi tạo biến reply để lưu trữ nội dung trả về từ stream.
display_handle = display(Markdown(""), display_id=True)	Hiển thị một đối tượng Markdown trống và tạo display_id để cập nhật nội dung trong quá trình stream.
for chunk in stream:	Duyệt qua từng phần (chunk) dữ liệu nhận được từ stream.
reply += chunk.choices[0].delta.content or ''	Thêm nội dung nhận được từ mỗi phần dữ liệu vào biến reply.
reply = reply.replace("```","").replace("markdown","")	Xử lý chuỗi reply bằng cách loại bỏ các phần tử không cần thiết như markdown và ``` từ nội dung trả về.
update_display(Markdown(reply),	Cập nhật nội dung Markdown đã xử lý lên giao diện người dùng trong quá trình stream.

display_id=display_handle.display_id)	

```
stream = openai.chat.completions.create(
    model='gpt-4o',
    messages=prompts,
    temperature=0.7,
    stream=True
)

reply = ""
display_handle = display(Markdown(""), display_id=True)
for chunk in stream:
    reply += chunk.choices[0].delta.content or ''
    reply = reply.replace("```","").replace("markdown","")
    update_display(Markdown(reply), display_id=display_handle.display_id)
```

Dòng lệnh	Chức năng
gpt_model = "gpt-4o-mini"	Khởi tạo biến gpt_model với giá trị tên mô hình GPT là gpt-4o-mini.
claude_model = "claude-3-haiku-20240307"	Khởi tạo biến claude_model với giá trị tên mô hình Claude là claude-3-haiku-20240307.
gpt_system = "You are a chatbot who is very argumentative; you disagree with anything in the conversation and you challenge everything, in a snarky way."	Khởi tạo biến gpt_system với thông điệp mô tả chatbot GPT sẽ luôn phản đối và thử thách mọi thứ trong cuộc trò chuyện một cách khiêu khích.
claude_system = "You are a very polite, courteous chatbot. You try to agree with everything the other person says, or find common ground. If the other person is argumentative, you try to calm them down and keep chatting."	Khởi tạo biến claude_system với thông điệp mô tả chatbot Claude sẽ luôn lịch sự, đồng ý hoặc tìm điểm chung, và cố gắng làm dịu cuộc trò chuyện khi người đối diện trở nên tranh cãi.
gpt_messages = ["Hi there"]	Khởi tạo biến gpt_messages chứa một thông điệp "Hi there" cho chatbot GPT.
claude_messages = ["Hi"]	Khởi tạo biến claude_messages chứa một thông điệp "Hi" cho chatbot Claude.

```
def call_claude():
    messages = []
    for gpt, claude_message in zip(gpt_messages, claude_messages):
        messages.append({"role": "user", "content": gpt})
        messages.append({"role": "assistant", "content": claude_message})
    messages.append({"role": "user", "content": gpt_messages[-1]})
    message = claude.messages.create(
        model=claude_model,
        system=claude_system,
        messages=messages,
        max_tokens=500
    )
    return message.content[0].text
```

Dòng lệnh	Chức năng
def call_claude():	Định nghĩa hàm call_claude để gọi mô hình Claude và xử lý các thông điệp.
messages = []	Khởi tạo danh sách rỗng messages để chứa các thông điệp trong cuộc trò chuyện.
for gpt, claude_message in zip(gpt_messages, claude_messages):	Duyệt qua các cặp thông điệp từ gpt_messages và claude_messages bằng cách sử dụng zip.
messages.append({"role": "user", "content": gpt})	Thêm thông điệp của người dùng (từ gpt_messages) vào danh sách messages.
messages.append({"role": "assistant", "content": claude_message})	Thêm thông điệp của trợ lý (từ claude_messages) vào danh sách messages.
messages.append({"role": "user", "content": gpt_messages[-1]})	Thêm thông điệp cuối cùng của người dùng vào danh sách messages.
message = claude.messages.create(model= claude_model, system=claude_system, messages=messages, max_tokens=500)	Gọi API để tạo phản hồi từ mô hình Claude với các thông điệp trong messages và cấu hình mô hình và thông điệp hệ thống.
return message.content[0].text	Trả về nội dung văn bản của phản hồi đầu tiên từ Claude.

```
call_claude()
call_gpt()
gpt_messages = ["Hi there"]
claude_messages = ["Hi"]

print(f"GPT:\n{gpt_messages[0]}\n")
print(f"Claude:\n{claude_messages[0]}\n")

for i in range(5):
    gpt_next = call_gpt()
    print(f"GPT:\n{gpt_next}\n")
    gpt_messages.append(gpt_next)

    claude_next = call_claude()
    print(f"Claude:\n{claude_next}\n")
    claude_messages.append(claude_next)
```

Dòng lệnh	Chức năng
call_claude()	Gọi hàm call_claude() để tạo phản hồi từ mô hình Claude và nhận kết quả.
call_gpt()	Gọi hàm call_gpt() để tạo phản hồi từ mô hình GPT và nhận kết quả.
gpt_messages = ["Hi there"]	Khởi tạo danh sách gpt_messages với thông điệp ban đầu "Hi there" cho GPT.
claude_messages = ["Hi"]	Khởi tạo danh sách claude_messages với thông điệp ban đầu "Hi" cho Claude.
print(f"GPT:\ n{gpt_messages[0]}\n")	In ra thông điệp đầu tiên trong gpt_messages (là "Hi there") cho GPT.
print(f"Claude:\ n{claude_messages[0]}\n")	In ra thông điệp đầu tiên trong claude_messages (là "Hi") cho Claude.
for i in range(5):	Lặp qua 5 vòng lặp để tạo ra các phản hồi tiếp theo cho GPT và Claude.
gpt_next = call_gpt()	Gọi hàm call_gpt() để nhận phản hồi tiếp theo từ GPT và lưu vào gpt_next.
print(f"GPT:\n{gpt_next}\n")	In ra phản hồi tiếp theo từ GPT.
gpt_messages.append(gpt_next)	Thêm phản hồi từ GPT vào danh sách gpt_messages.
claude_next = call_claude()	Gọi hàm call_claude() để nhận phản hồi tiếp theo từ Claude và lưu vào claude_next.
print(f"Claude:\ n{claude_next}\n")	In ra phản hồi tiếp theo từ Claude.
claude_messages.append(claud e_next)	Thêm phản hồi từ Claude vào danh sách claude_messages.