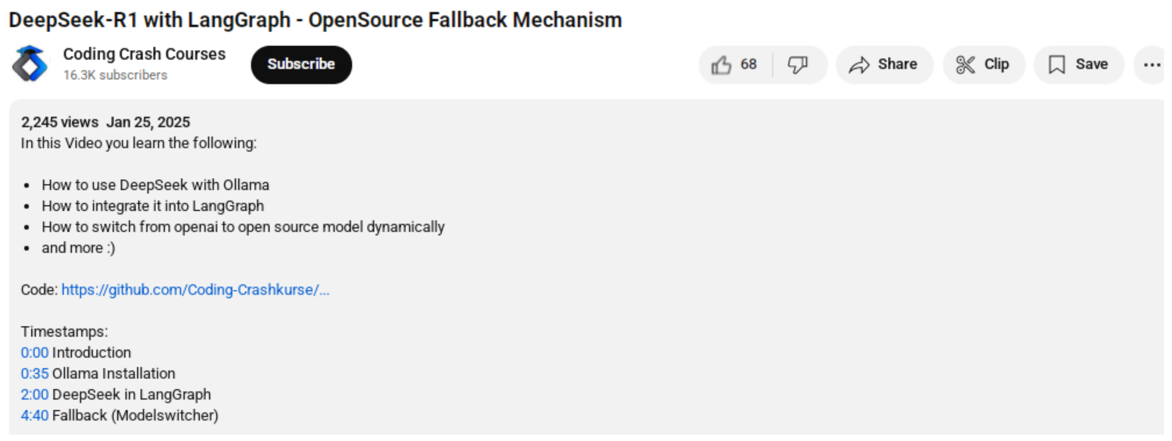


DeepSeek-R1 with LangGraph - OpenSource Fallback Mechanism

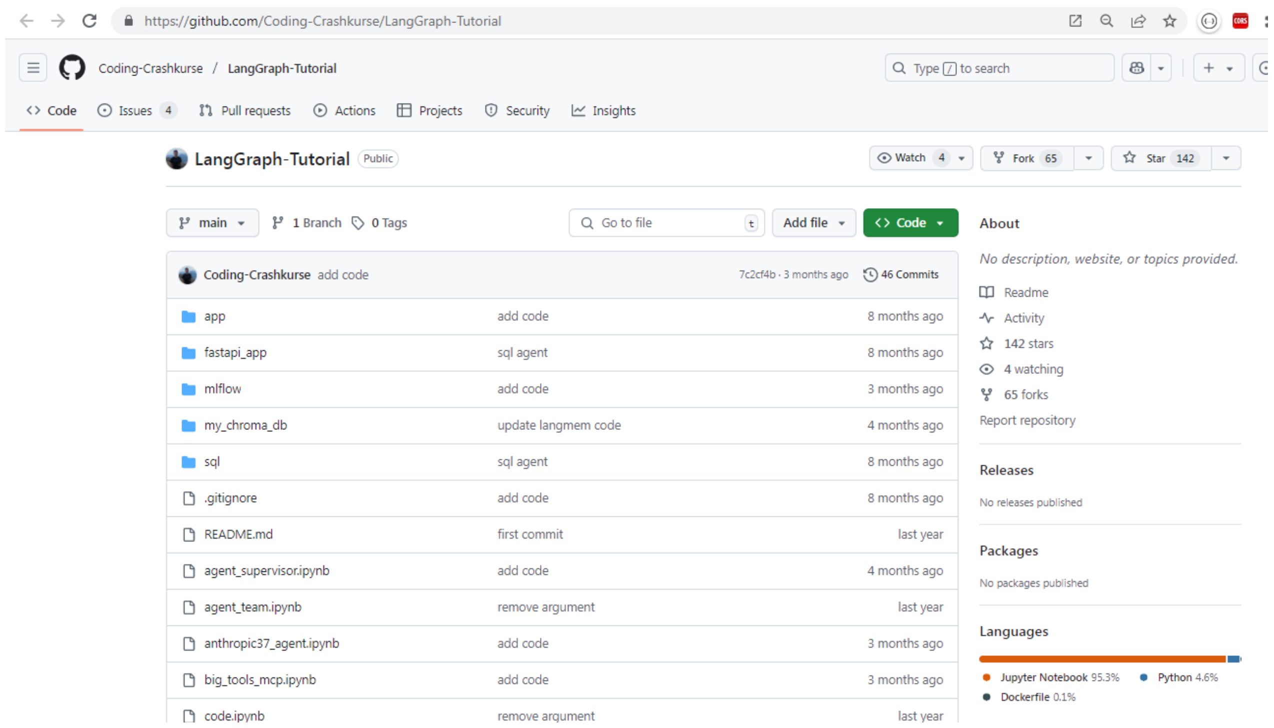
Coding Crash Courses
16.3K subscribers

2,245 views Jan 25, 2025
In this Video you learn the following:

- How to use DeepSeek with Ollama
- How to integrate it into LangGraph
- How to switch from openai to open source model dynamically
- and more :)

Code: https://github.com/Coding-Crashkurse/...

Timestamps:
0:00 Introduction
0:35 Ollama Installation
2:00 DeepSeek in LangGraph
4:40 Fallback (Modelswitcher)



https://github.com/Coding-Crashkurse/LangGraph-Tutorial

Coding-Crashkurse / LangGraph-Tutorial

LangGraph-Tutorial Public

Coding-Crashkurse add code 7c2cf4b · 3 months ago 46 Commits

app	add code	8 months ago
fastapi_app	sql agent	8 months ago
mlflow	add code	3 months ago
my_chroma_db	update langmem code	4 months ago
sql	sql agent	8 months ago
.gitignore	add code	8 months ago
README.md	first commit	last year
agent_supervisor.ipynb	add code	4 months ago
agent_team.ipynb	remove argument	last year
anthropic37_agent.ipynb	add code	3 months ago
big_tools_mcp.ipynb	add code	3 months ago
code.ipynb	remove argument	last year

About

No description, website, or topics provided.

Readme
Activity
142 stars
4 watching
65 forks
Report repository

Releases

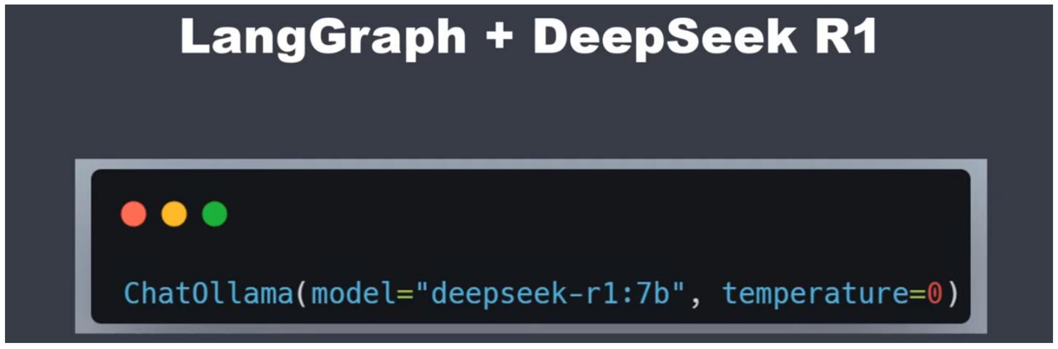
No releases published

Packages

No packages published

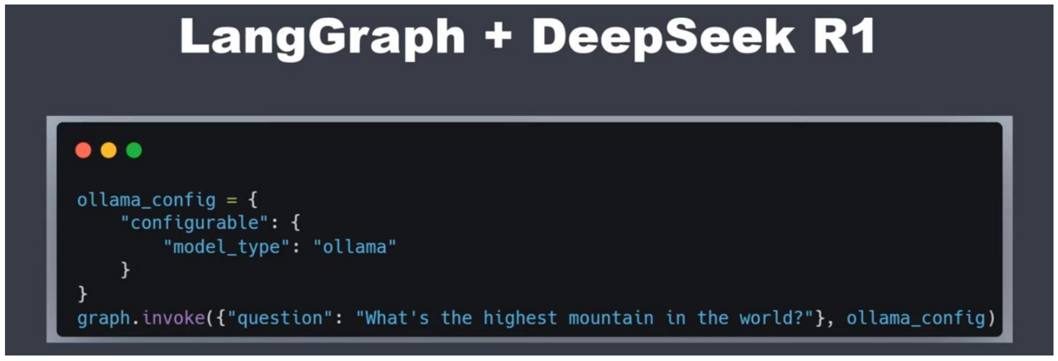
Languages

● Jupyter Notebook 95.3% ● Python 4.6%
● Dockerfile 0.1%



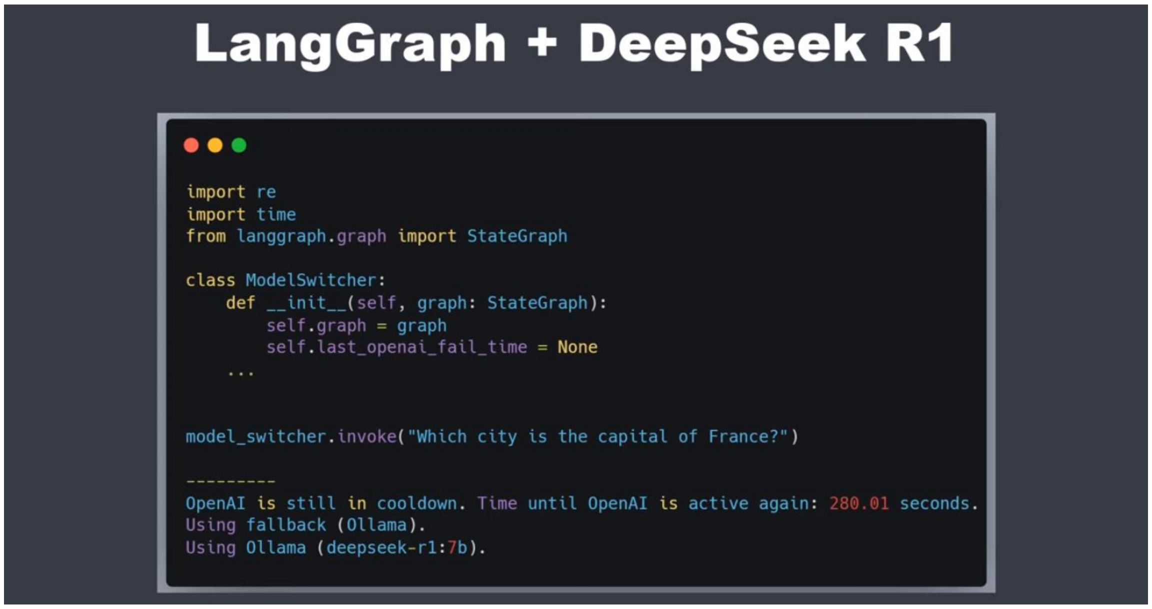
LangGraph + DeepSeek R1

```python
ChatOllama(model="deepseek-r1:7b", temperature=0)
```



LangGraph + DeepSeek R1

```python
ollama_config = {
    "configurable": {
        "model_type": "ollama"
    }
}
graph.invoke({"question": "What's the highest mountain in the world?"}, ollama_config)
```

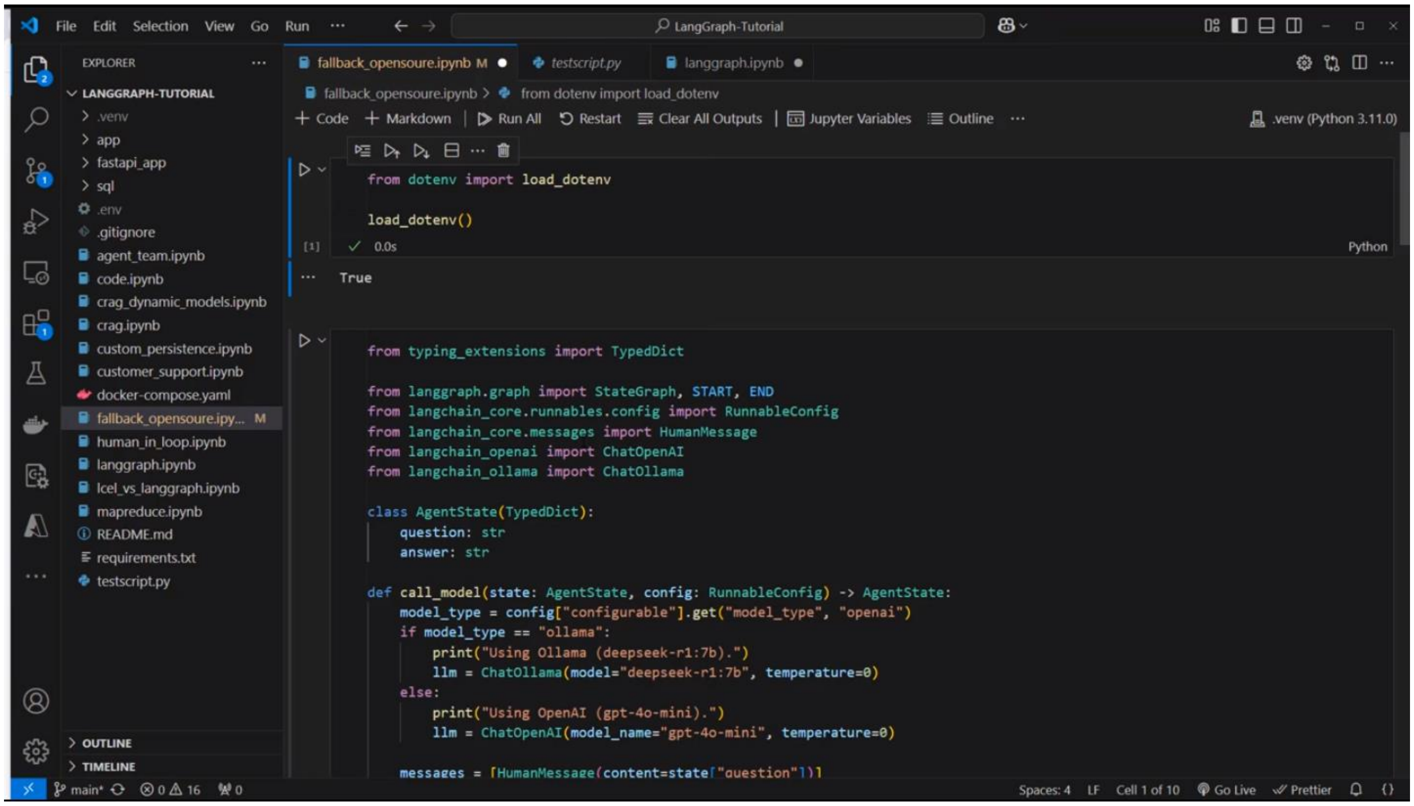


LangGraph + DeepSeek R1

```python
import re
import time
from langgraph.graph import StateGraph

class ModelSwitcher:
    def __init__(self, graph: StateGraph):
        self.graph = graph
        self.last_openai_fail_time = None
    ...


model_switcher.invoke("Which city is the capital of France?")

---------
OpenAI is still in cooldown. Time until OpenAI is active again: 280.01 seconds.
Using fallback (Ollama).
Using Ollama (deepseek-r1:7b).
```

LangGraph + DeepSeek R1

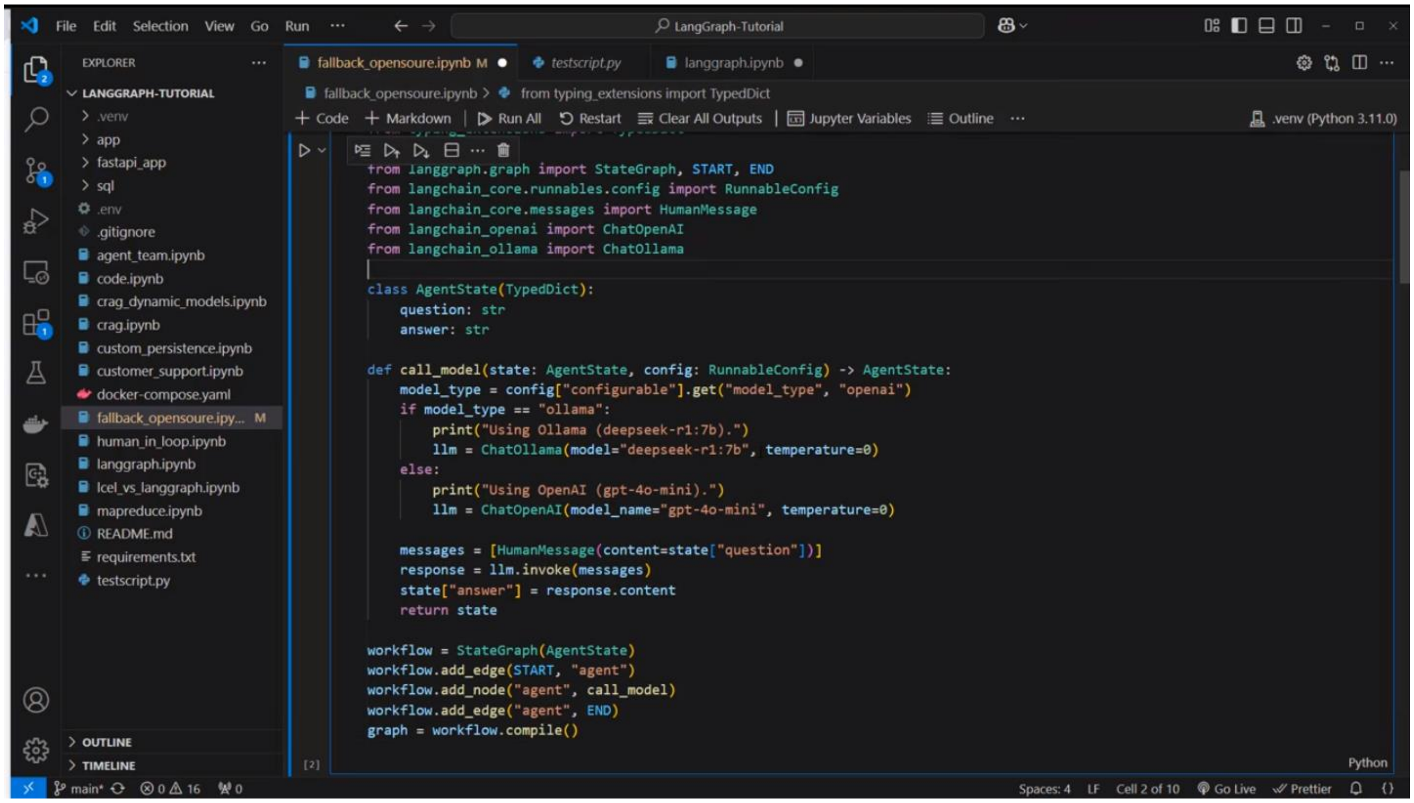
Ollama is the easiest way to setup DeepSeek R1



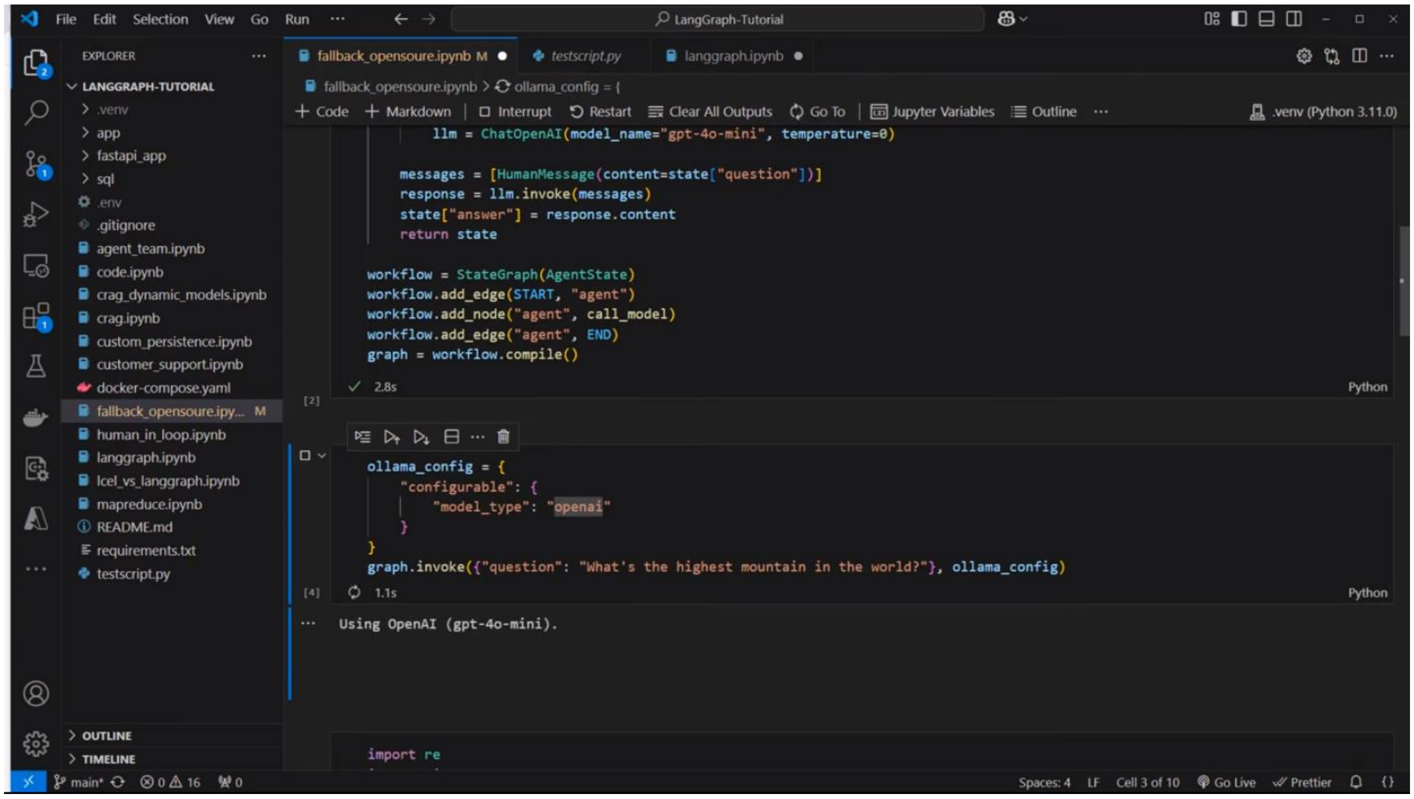
```python
from dotenv import load_dotenv

load_dotenv()
```

```
[1] ✓ 0.0s

True
```

```python
from typing_extensions import TypedDict

from langgraph.graph import StateGraph, START, END
from langchain_core.runnables.config import RunnableConfig
from langchain_core.messages import HumanMessage
from langchain_openai import ChatOpenAI
from langchain_ollama import ChatOllama

class AgentState(TypedDict):
    question: str
    answer: str

def call_model(state: AgentState, config: RunnableConfig) -> AgentState:
    model_type = config["configurable"].get("model_type", "openai")
    if model_type == "ollama":
        print("Using Ollama (deepseek-r1:7b).")
        llm = ChatOllama(model="deepseek-r1:7b", temperature=0)
    else:
        print("Using OpenAI (gpt-4o-mini).")
        llm = ChatOpenAI(model_name="gpt-4o-mini", temperature=0)

    messages = [HumanMessage(content=state["question"])]
```



```python
from langgraph.graph import StateGraph, START, END
from langchain_core.runnables.config import RunnableConfig
from langchain_core.messages import HumanMessage
from langchain_openai import ChatOpenAI
from langchain_ollama import ChatOllama

class AgentState(TypedDict):
    question: str
    answer: str

def call_model(state: AgentState, config: RunnableConfig) -> AgentState:
    model_type = config["configurable"].get("model_type", "openai")
    if model_type == "ollama":
        print("Using Ollama (deepseek-r1:7b).")
        llm = ChatOllama(model="deepseek-r1:7b", temperature=0)
    else:
        print("Using OpenAI (gpt-4o-mini).")
        llm = ChatOpenAI(model_name="gpt-4o-mini", temperature=0)

    messages = [HumanMessage(content=state["question"])]
    response = llm.invoke(messages)
    state["answer"] = response.content
    return state

workflow = StateGraph(AgentState)
workflow.add_edge(START, "agent")
workflow.add_node("agent", call_model)
workflow.add_edge("agent", END)
graph = workflow.compile()
```

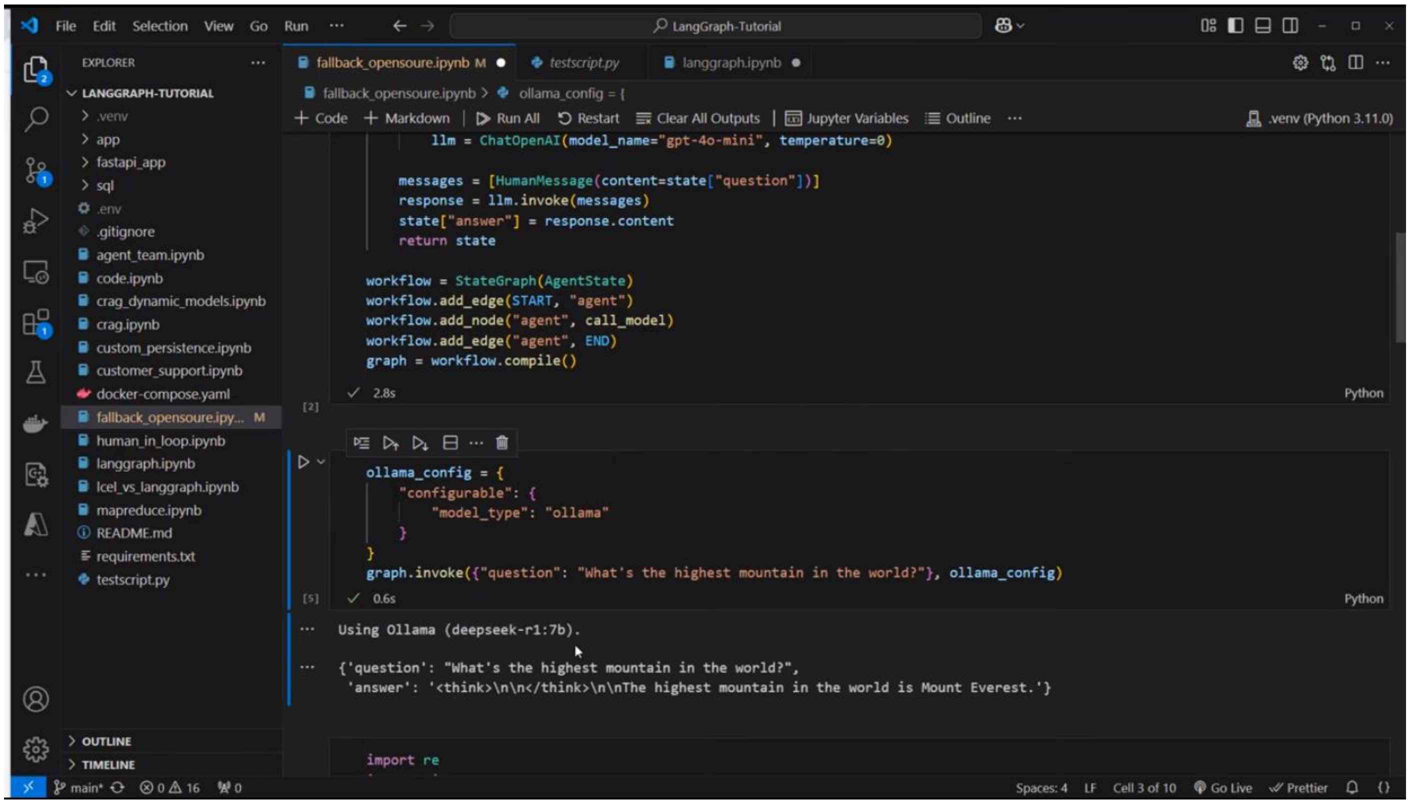


```python
        llm = ChatOpenAI(model_name="gpt-4o-mini", temperature=0)

    messages = [HumanMessage(content=state["question"])]
    response = llm.invoke(messages)
    state["answer"] = response.content
    return state

workflow = StateGraph(AgentState)
workflow.add_edge(START, "agent")
workflow.add_node("agent", call_model)
workflow.add_edge("agent", END)
graph = workflow.compile()
```

```python
ollama_config = {
    "configurable": {
        "model_type": "openai"
    }
}
graph.invoke({"question": "What's the highest mountain in the world?"}, ollama_config)
```

```
Using OpenAI (gpt-4o-mini).
```

```python
import re
```



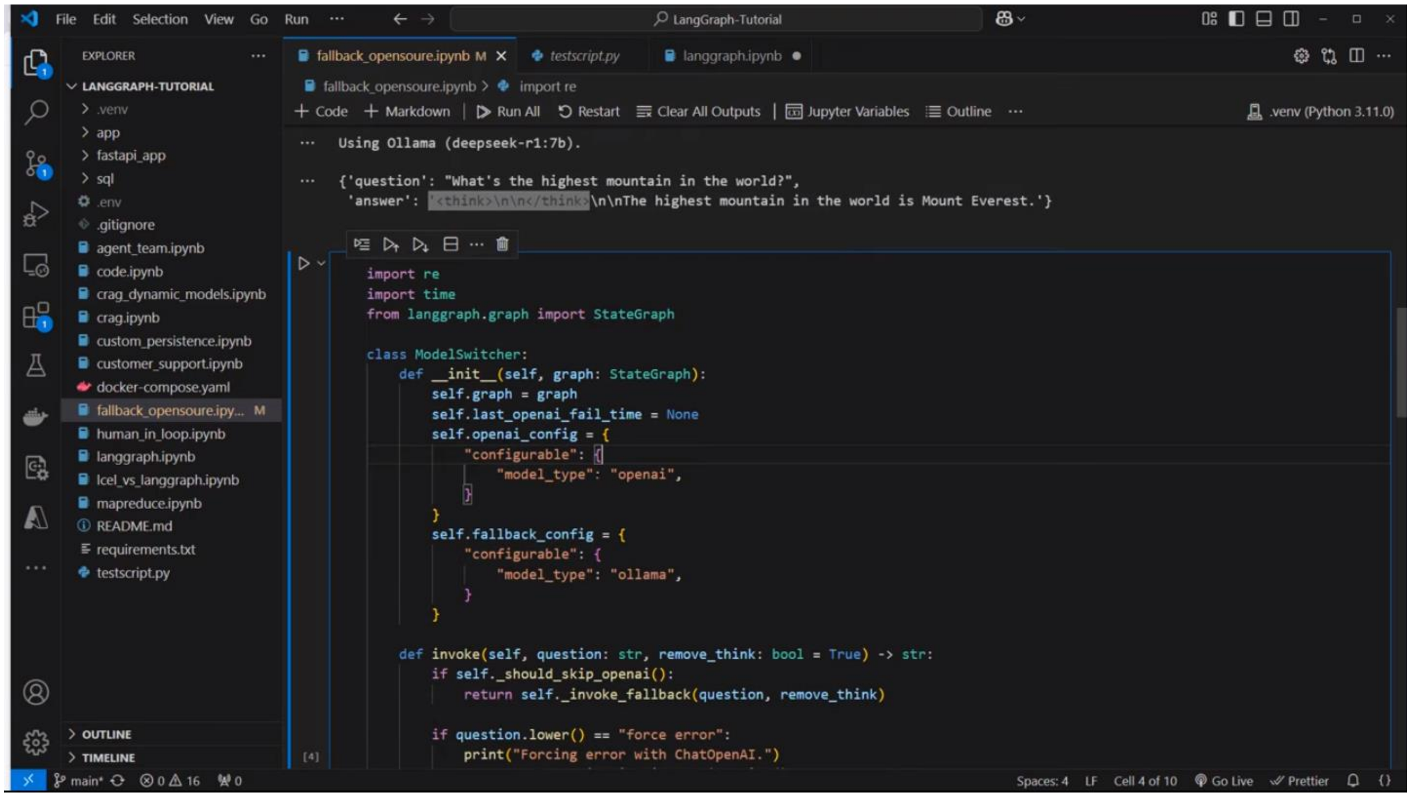
```
        llm = ChatOpenAI(model_name="gpt-4o-mini", temperature=0)

    messages = [HumanMessage(content=state["question"])]
    response = llm.invoke(messages)
    state["answer"] = response.content
    return state

workflow = StateGraph(AgentState)
workflow.add_edge(START, "agent")
workflow.add_node("agent", call_model)
workflow.add_edge("agent", END)
graph = workflow.compile()
```

```
ollama_config = {
    "configurable": {
        "model_type": "ollama"
    }
}
graph.invoke({"question": "What's the highest mountain in the world?"}, ollama_config)
```

```
Using Ollama (deepseek-r1:7b).

{'question': "What's the highest mountain in the world?",
 'answer': '<think>\n\n</think>\n\nThe highest mountain in the world is Mount Everest.'}
```

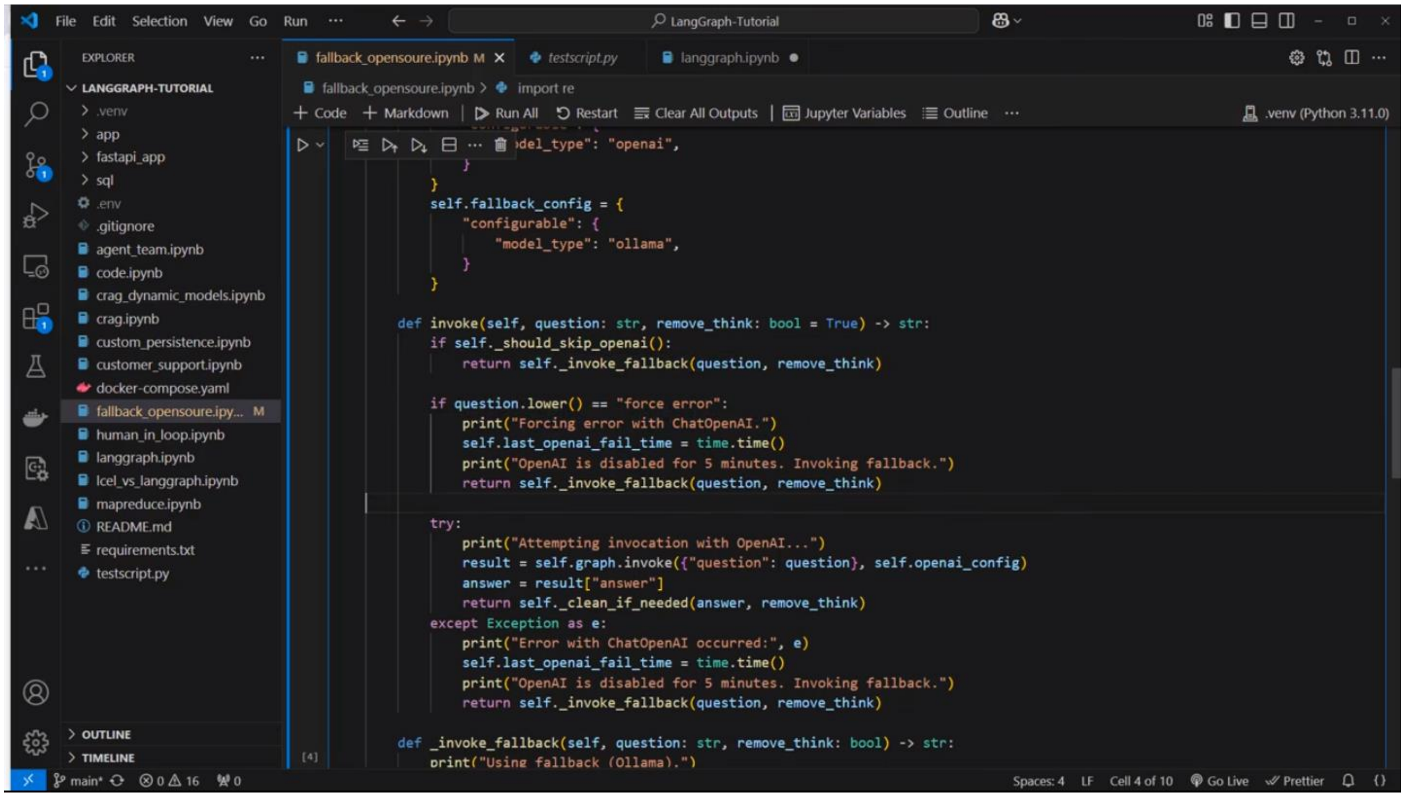
```
import re
```

We can now use OpenAI and DeepSeek as the fallback as below

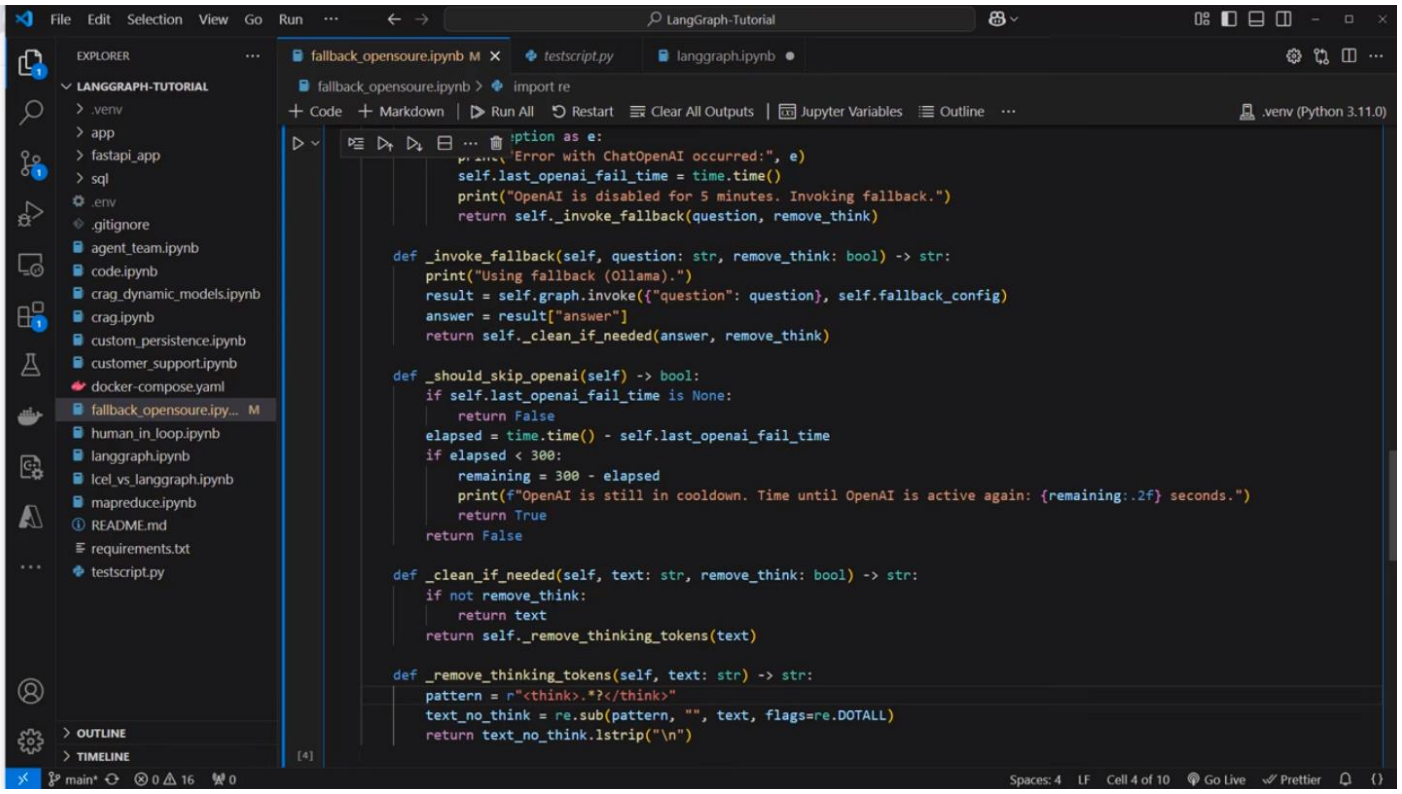


```
Using Ollama (deepseek-r1:7b).

{'question': "What's the highest mountain in the world?",
 'answer': '<think>\n\n</think>\n\nThe highest mountain in the world is Mount Everest.'}
```

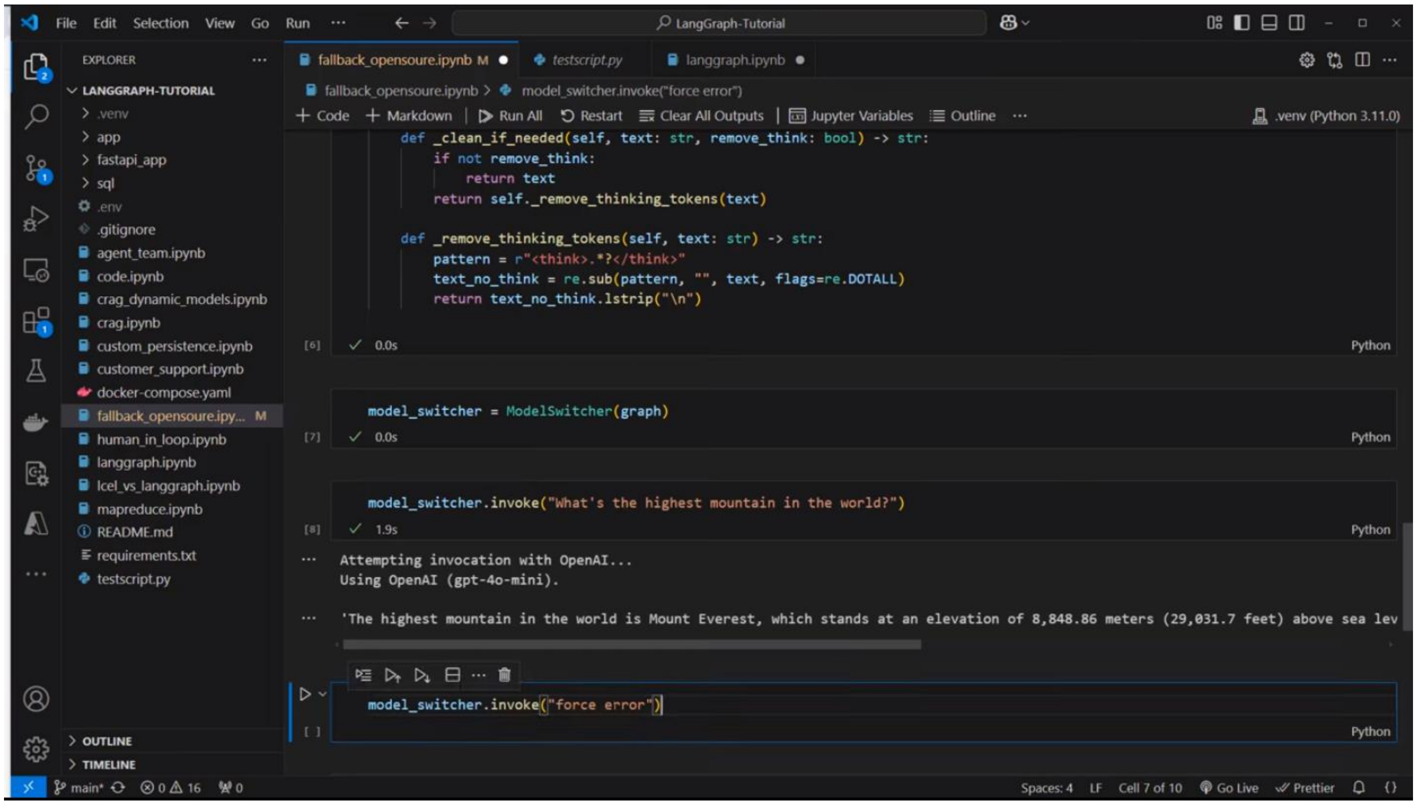
```
import re
import time
from langgraph.graph import StateGraph

class ModelSwitcher:
    def __init__(self, graph: StateGraph):
        self.graph = graph
        self.last_openai_fail_time = None
        self.openai_config = {
            "configurable": {
                "model_type": "openai",
            }
        }
        self.fallback_config = {
            "configurable": {
                "model_type": "ollama",
            }
        }

    def invoke(self, question: str, remove_think: bool = True) -> str:
        if self._should_skip_openai():
            return self._invoke_fallback(question, remove_think)

        if question.lower() == "force error":
            print("Forcing error with ChatOpenAI.")
```



```
                odel_type": "openai",
            }
        }
        self.fallback_config = {
            "configurable": {
                "model_type": "ollama",
            }
        }

    def invoke(self, question: str, remove_think: bool = True) -> str:
        if self._should_skip_openai():
            return self._invoke_fallback(question, remove_think)

        if question.lower() == "force error":
            print("Forcing error with ChatOpenAI.")
            self.last_openai_fail_time = time.time()
            print("OpenAI is disabled for 5 minutes. Invoking fallback.")
            return self._invoke_fallback(question, remove_think)

        try:
            print("Attempting invocation with OpenAI...")
            result = self.graph.invoke({"question": question}, self.openai_config)
            answer = result["answer"]
            return self._clean_if_needed(answer, remove_think)
        except Exception as e:
            print("Error with ChatOpenAI occurred:", e)
            self.last_openai_fail_time = time.time()
            print("OpenAI is disabled for 5 minutes. Invoking fallback.")
            return self._invoke_fallback(question, remove_think)

    def _invoke_fallback(self, question: str, remove_think: bool) -> str:
        print("Using fallback (Ollama).")
```



```
               ption as e:
            print("Error with ChatOpenAI occurred:", e)
            self.last_openai_fail_time = time.time()
            print("OpenAI is disabled for 5 minutes. Invoking fallback.")
            return self._invoke_fallback(question, remove_think)

    def _invoke_fallback(self, question: str, remove_think: bool) -> str:
        print("Using fallback (Ollama).")
        result = self.graph.invoke({"question": question}, self.fallback_config)
        answer = result["answer"]
        return self._clean_if_needed(answer, remove_think)

    def _should_skip_openai(self) -> bool:
        if self.last_openai_fail_time is None:
            return False
        elapsed = time.time() - self.last_openai_fail_time
        if elapsed < 300:
            remaining = 300 - elapsed
            print(f"OpenAI is still in cooldown. Time until OpenAI is active again: {remaining:.2f} seconds.")
            return True
        return False

    def _clean_if_needed(self, text: str, remove_think: bool) -> str:
        if not remove_think:
            return text
        return self._remove_thinking_tokens(text)

    def _remove_thinking_tokens(self, text: str) -> str:
        pattern = r"<think>.*?</think>"
        text_no_think = re.sub(pattern, "", text, flags=re.DOTALL)
        return text_no_think.lstrip("\n")
```



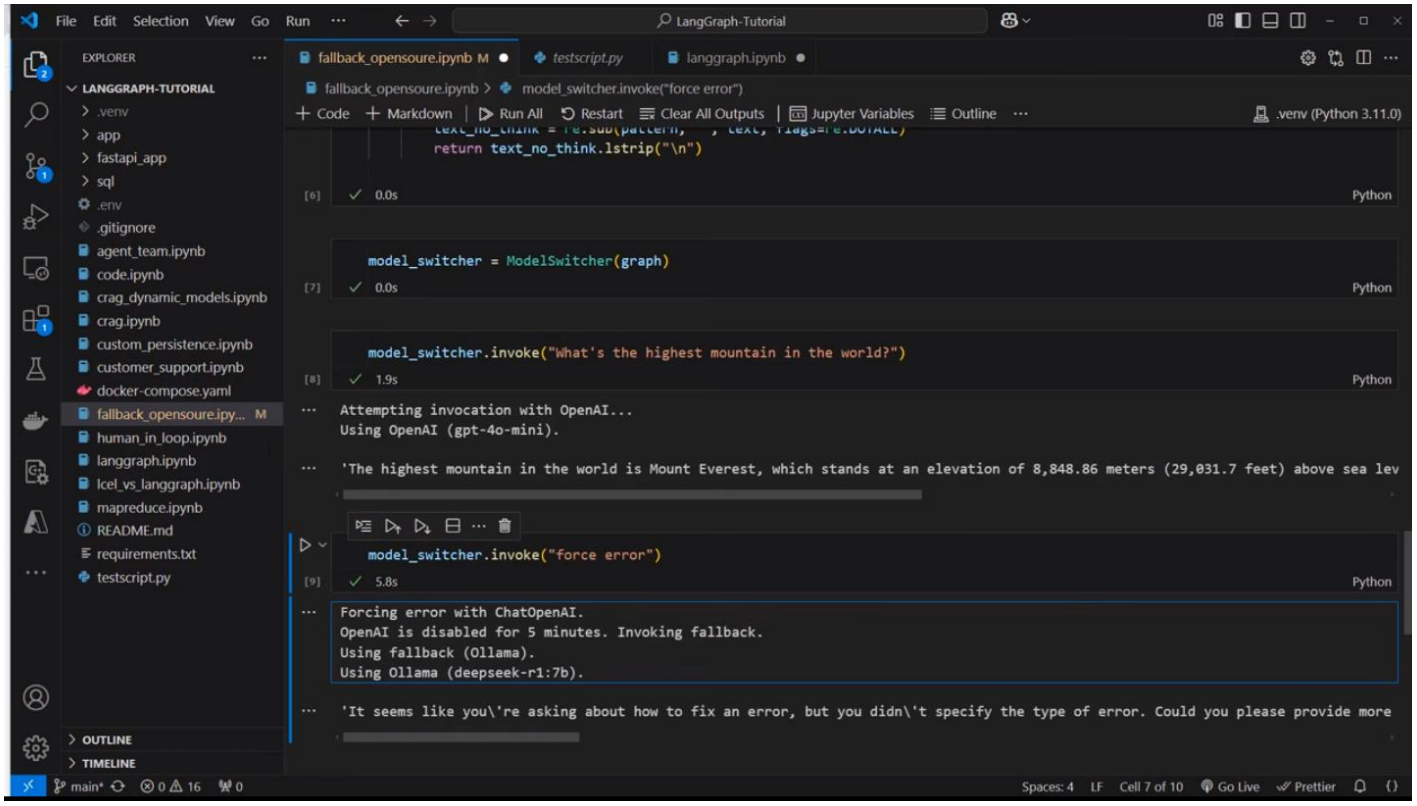
```
def _clean_if_needed(self, text: str, remove_think: bool) -> str:
    if not remove_think:
        return text
    return self._remove_thinking_tokens(text)

def _remove_thinking_tokens(self, text: str) -> str:
    pattern = r"<think>.*?</think>"
    text_no_think = re.sub(pattern, "", text, flags=re.DOTALL)
    return text_no_think.lstrip("\n")
```

```
model_switcher = ModelSwitcher(graph)
```

```
model_switcher.invoke("What's the highest mountain in the world?")
```

```
Attempting invocation with OpenAI...
Using OpenAI (gpt-4o-mini).

'The highest mountain in the world is Mount Everest, which stands at an elevation of 8,848.86 meters (29,031.7 feet) above sea lev
```

```
model_switcher.invoke("force error")
```



```
    return text_no_think.lstrip("\n")
```

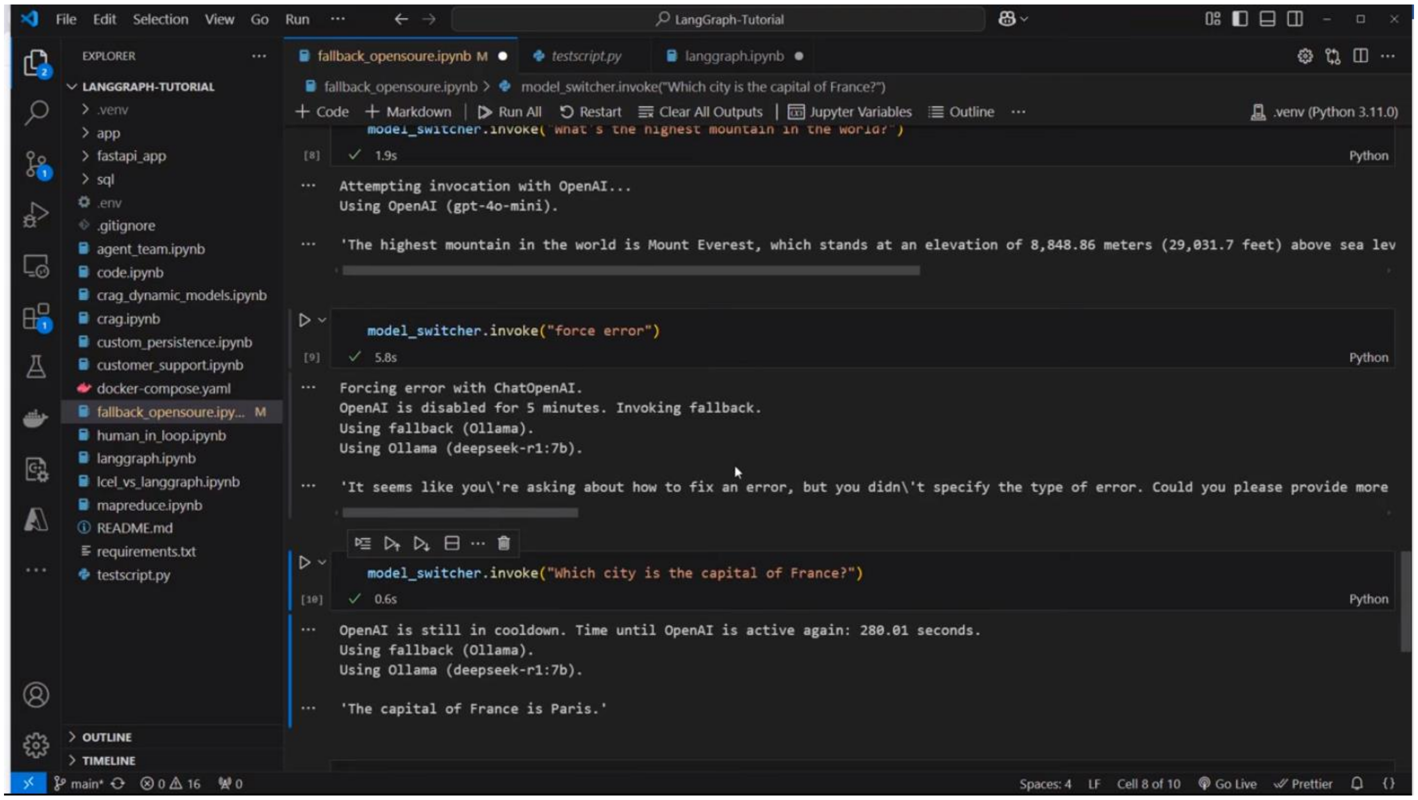
```
model_switcher = ModelSwitcher(graph)
```

```
model_switcher.invoke("What's the highest mountain in the world?")
```

```
Attempting invocation with OpenAI...
Using OpenAI (gpt-4o-mini).

'The highest mountain in the world is Mount Everest, which stands at an elevation of 8,848.86 meters (29,031.7 feet) above sea lev
```

```
model_switcher.invoke("force error")
```

```
Forcing error with ChatOpenAI.
OpenAI is disabled for 5 minutes. Invoking fallback.
Using fallback (Ollama).
Using Ollama (deepseek-r1:7b).

'It seems like you\'re asking about how to fix an error, but you didn\'t specify the type of error. Could you please provide more
```



```
model_switcher.invoke("What's the highest mountain in the world?")
```

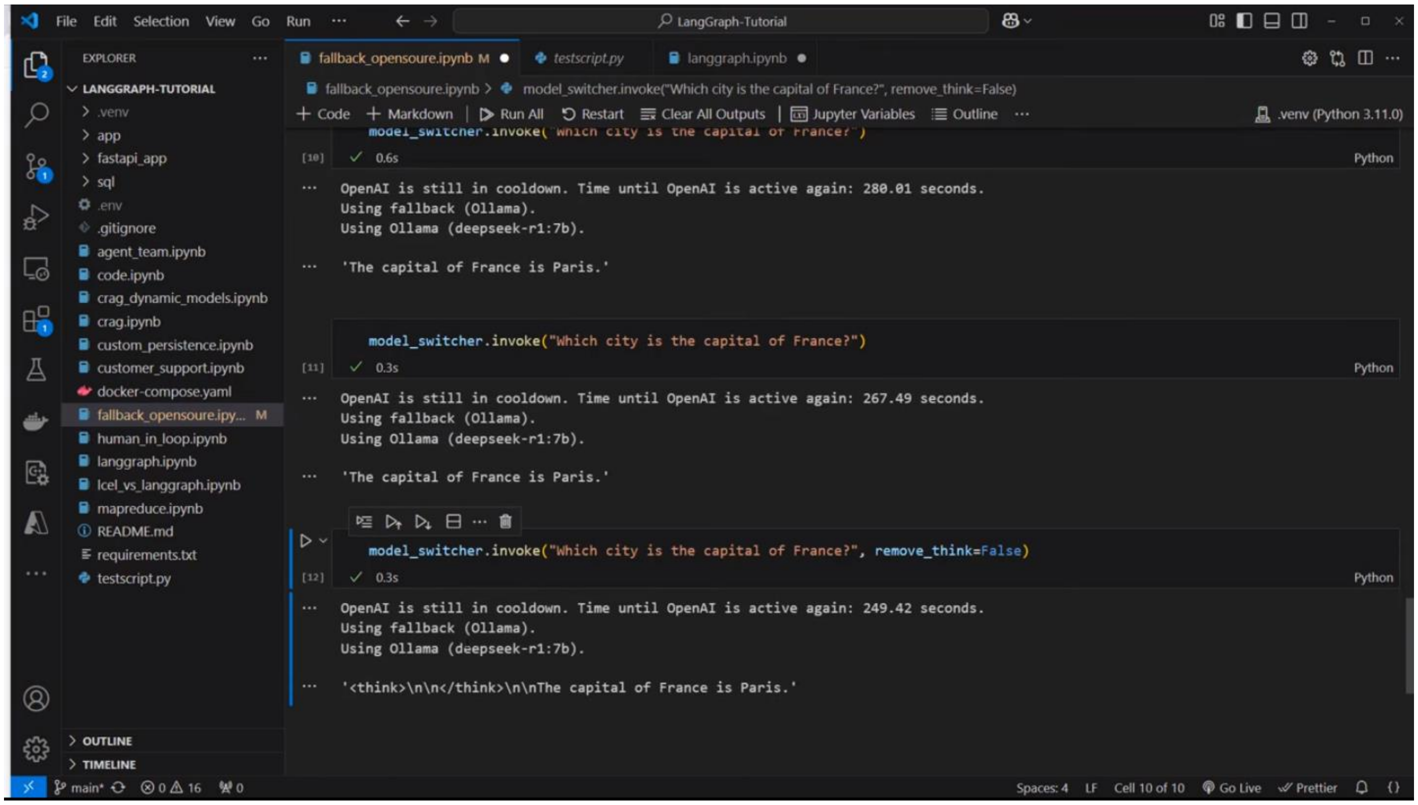
```
Attempting invocation with OpenAI...
Using OpenAI (gpt-4o-mini).

'The highest mountain in the world is Mount Everest, which stands at an elevation of 8,848.86 meters (29,031.7 feet) above sea lev
```

```
model_switcher.invoke("force error")
```

```
Forcing error with ChatOpenAI.
OpenAI is disabled for 5 minutes. Invoking fallback.
Using fallback (Ollama).
Using Ollama (deepseek-r1:7b).

'It seems like you\'re asking about how to fix an error, but you didn\'t specify the type of error. Could you please provide more
```

```
model_switcher.invoke("Which city is the capital of France?")
```

```
OpenAI is still in cooldown. Time until OpenAI is active again: 280.01 seconds.
Using fallback (Ollama).
Using Ollama (deepseek-r1:7b).

'The capital of France is Paris.'
```



```
model_switcher.invoke("Which city is the capital of France?")
```

```
OpenAI is still in cooldown. Time until OpenAI is active again: 280.01 seconds.
Using fallback (Ollama).
Using Ollama (deepseek-r1:7b).

'The capital of France is Paris.'
```

```
model_switcher.invoke("Which city is the capital of France?")
```

```
OpenAI is still in cooldown. Time until OpenAI is active again: 267.49 seconds.
Using fallback (Ollama).
Using Ollama (deepseek-r1:7b).

'The capital of France is Paris.'
```

```
model_switcher.invoke("Which city is the capital of France?", remove_think=False)
```

```
OpenAI is still in cooldown. Time until OpenAI is active again: 249.42 seconds.
Using fallback (Ollama).
Using Ollama (deepseek-r1:7b).

'<think>\n\n</think>\n\nThe capital of France is Paris.'
```