Models - OneNote

Search

Buy Microsoft 365

File Home Insert Draw History Review View Help

Sticky Notes Share

Add Pen Insert Space Shapes Automatic Shapes Full Page View Ink Help

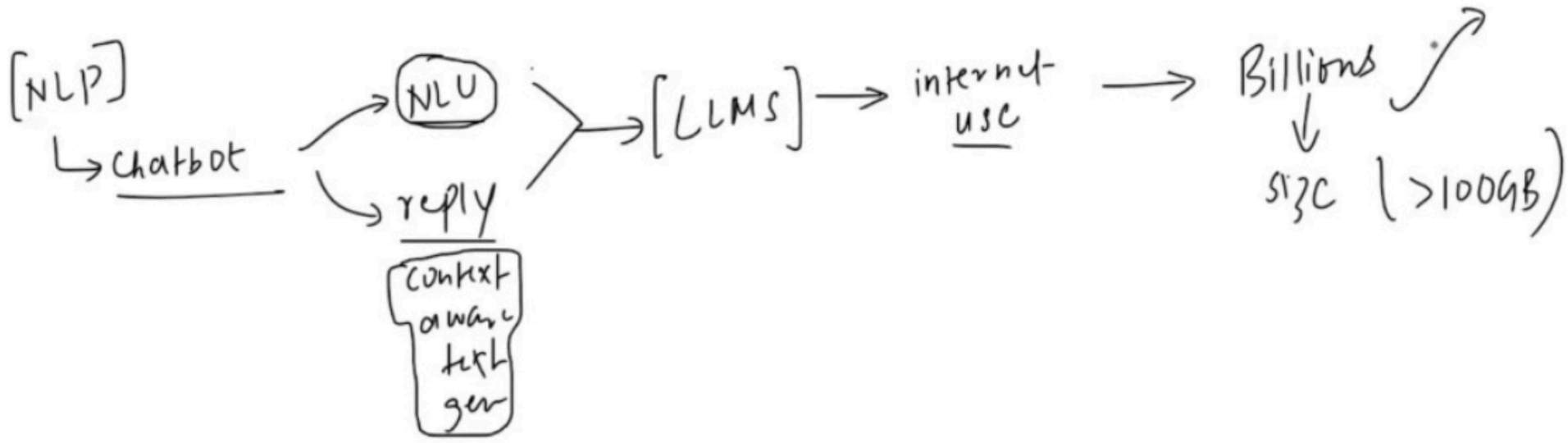
Search Notebooks

Recap
LangChain Comp...
Models
Prompts
Chain
Indexes
Memory
Agents

Models

23 January 2025 10:30

In LangChain, "models" are the core interfaces through which you interact with AI models.



Create a human-like response to a prompt

```python
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
```

claude_quickstart.py

```python
import anthropic

client = anthropic.Anthropic()

message = client.messages.create(
```

text gen

OpenAI

Create a human-like response to a prompt

```python
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
    model="gpt-4o-mini",
    messages=[
```

Claude api

claude_quickstart.py

```python
import anthropic

client = anthropic.Anthropic()

message = client.messages.create(
```

Models - OneNote

Search

Buy Microsoft 365

File Home Insert Draw History Review View Help

Sticky Notes Share

Add Pen Insert Space Shapes Automatic Shapes Full Page View Ink Help

Search Notebooks

Recap
LangChain Comp...
Models
Prompts
Chain
Indexes
Memory
Agents

openAI.

langchain

Claude

```python
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv

load_dotenv()

model = ChatOpenAI(model='gpt-4', temperature=0)

result = model.invoke("Now divide the result by 1.5")

print(result.content)
```

```python
from langchain_anthropic import ChatAnthropic
from dotenv import load_dotenv

load_dotenv()

model = ChatAnthropic(model='claude-3-opus-20240229')

result = model.invoke("Hi who are you")

print(result.content)
```

Prompts - OneNote

Search

Buy Microsoft 365

File Home Insert Draw History Review View Help

Sticky Notes Share

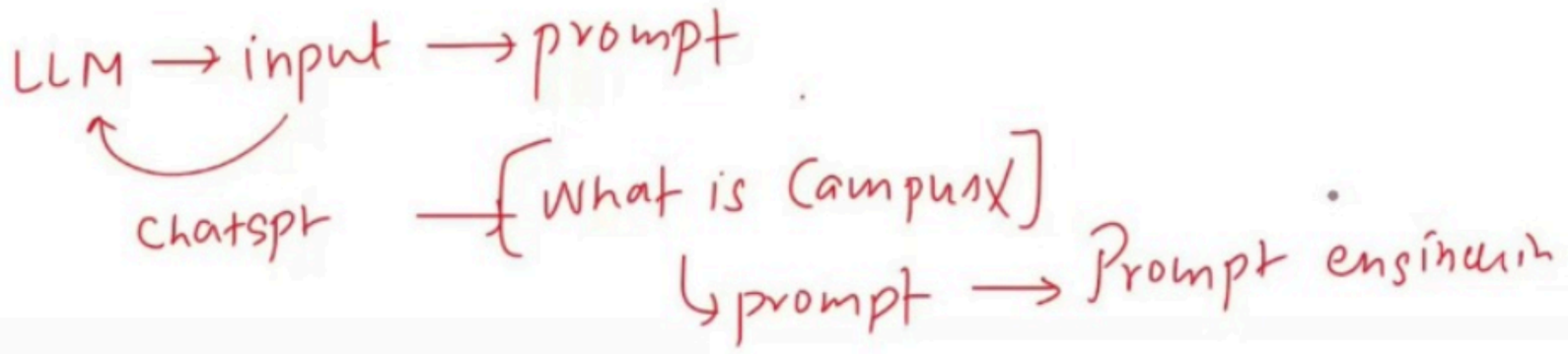
Add Pen Insert Space Shapes Automatic Shapes Full Page View Ink Help

Search Notebooks

Recap
LangChain Comp...
Models
Prompts
Chains
Indexes
Memory
Agents

Prompts

23 January 2025 10:30



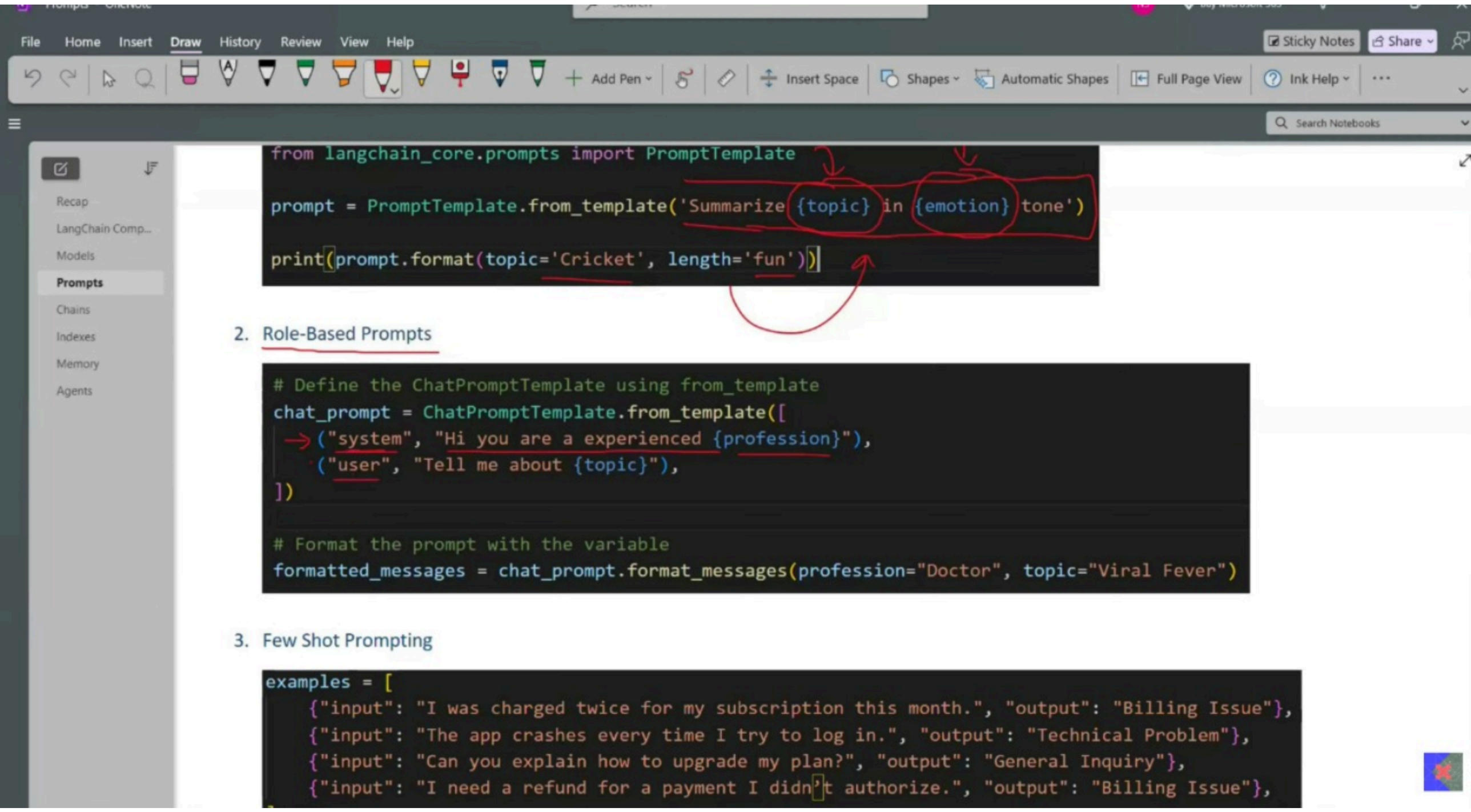
1. Dynamic & Reusable Prompts

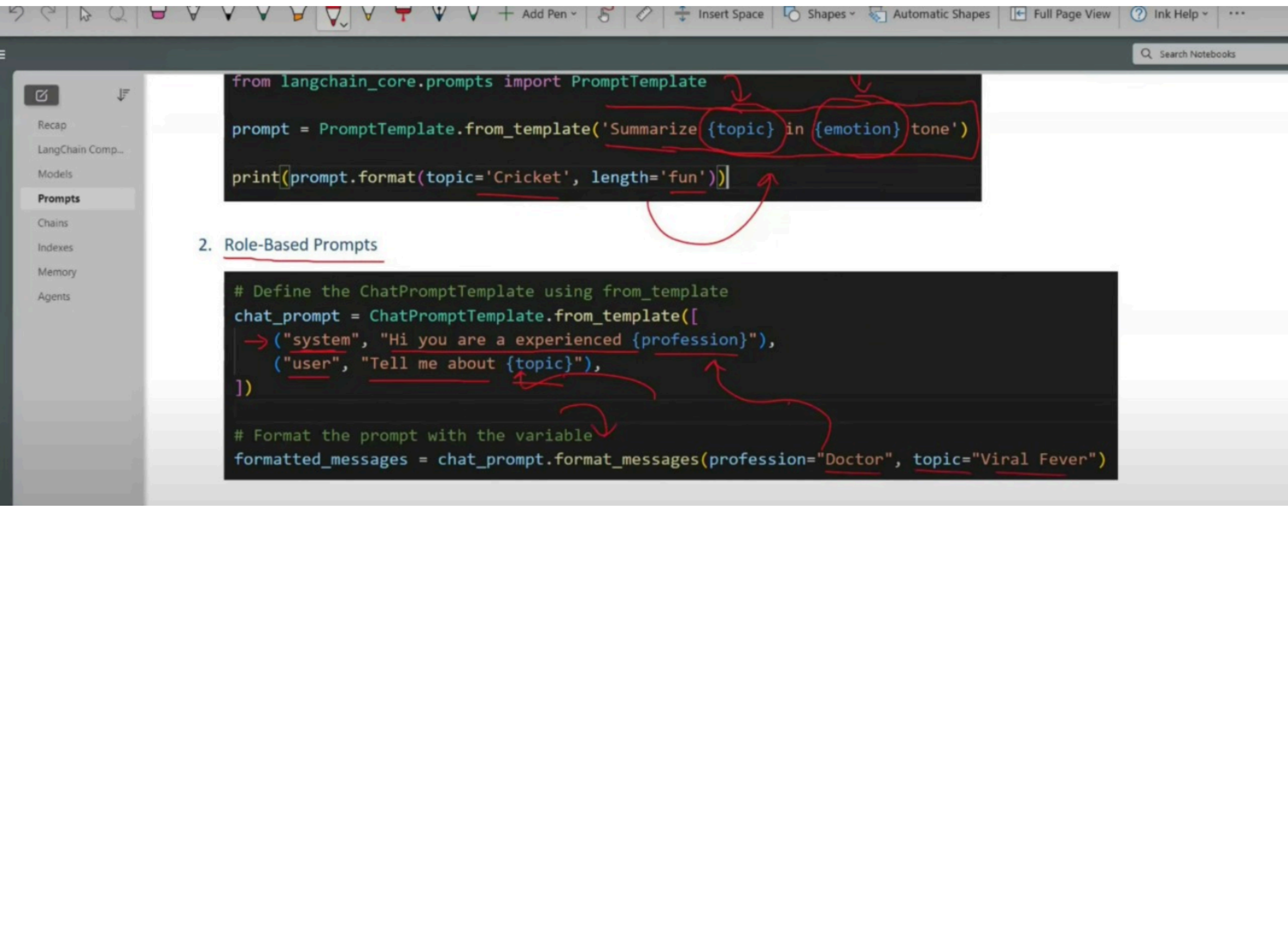
```
from langchain_core.prompts import PromptTemplate

prompt = PromptTemplate.from_template('Summarize {topic} in {emotion} tone')

print(prompt.format(topic='Cricket', length='fun'))
```

2. Role-Based Prompts


File Home Insert Draw History Review View Help
Sticky Notes
Share
Add Pen
Insert Space
Shapes
Automatic Shapes
Full Page View
Ink Help
Search Notebooks
Recap
LangChain Comp...
Models
Prompts
Chains
Indexes
Memory
Agents
from langchain_core.prompts import PromptTemplate
prompt = PromptTemplate.from_template('Summarize {topic} in {emotion} tone')
print(prompt.format(topic='Cricket', length='fun'))
2. Role-Based Prompts
# Define the ChatPromptTemplate using from_template
chat_prompt = ChatPromptTemplate.from_template([
("system", "Hi you are a experienced {profession}"),
("user", "Tell me about {topic}"),
])
# Format the prompt with the variable
formatted_messages = chat_prompt.format_messages(profession="Doctor", topic="Viral Fever")
3. Few Shot Prompting
examples = [
{"input": "I was charged twice for my subscription this month.", "output": "Billing Issue"},
{"input": "The app crashes every time I try to log in.", "output": "Technical Problem"},
{"input": "Can you explain how to upgrade my plan?", "output": "General Inquiry"},
{"input": "I need a refund for a payment I didn't authorize.", "output": "Billing Issue"},


Add Pen
Insert Space
Shapes
Automatic Shapes
Full Page View
Ink Help
Search Notebooks
Recap
LangChain Comp...
Models
Prompts
Chains
Indexes
Memory
Agents
from langchain_core.prompts import PromptTemplate
prompt = PromptTemplate.from_template('Summarize {topic} in {emotion} tone')
print(prompt.format(topic='Cricket', length='fun'))
2. Role-Based Prompts
# Define the ChatPromptTemplate using from_template
chat_prompt = ChatPromptTemplate.from_template([
("system", "Hi you are a experienced {profession}"),
("user", "Tell me about {topic}"),
])
# Format the prompt with the variable
formatted_messages = chat_prompt.format_messages(profession="Doctor", topic="Viral Fever")

Search Notebooks

Recap
LangChain Comp...
Models
Prompts
Chains
Indexes
Memory
Agents

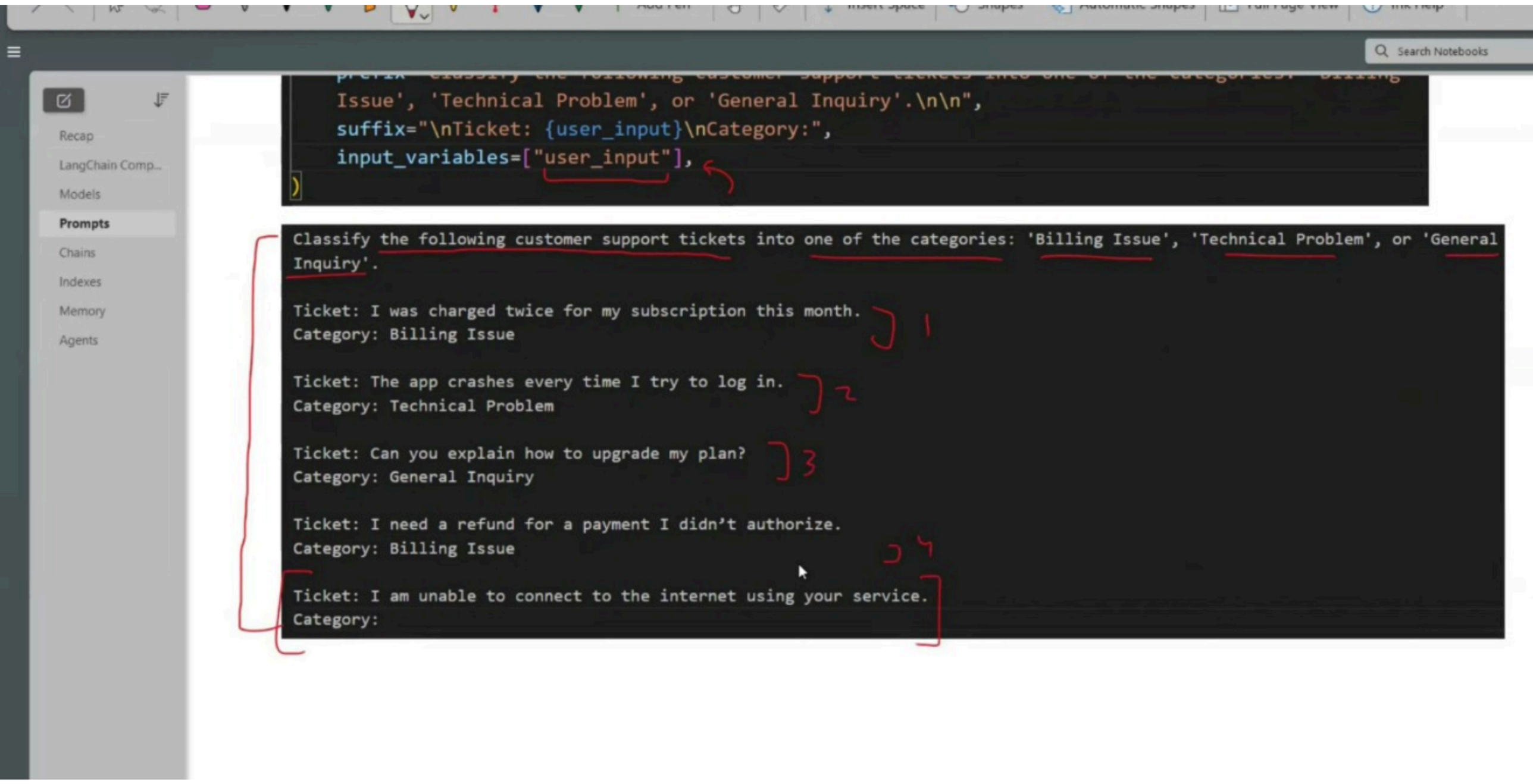
```
formatted_messages = chat_prompt.format_messages(profession="Doctor", topic="Viral Fever")
```

3. Few Shot Prompting

```
examples = [
    {"input": "I was charged twice for my subscription this month.", "output": "Billing Issue"},
    {"input": "The app crashes every time I try to log in.", "output": "Technical Problem"},
    {"input": "Can you explain how to upgrade my plan?", "output": "General Inquiry"},
    {"input": "I need a refund for a payment I didn't authorize.", "output": "Billing Issue"},
]
```

```
# Step 2: Create an example template
example_template = """
Ticket: {input}
Category: {output}
"""
```

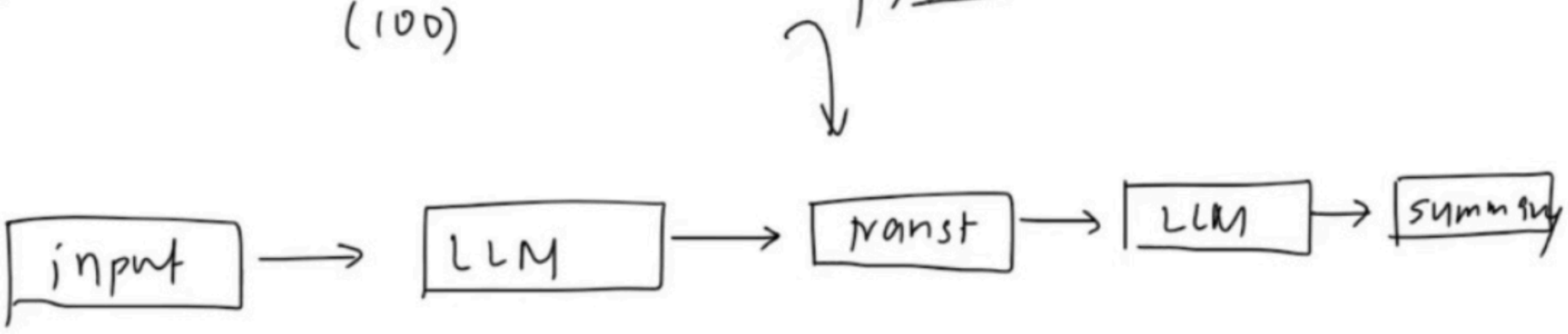
```
# Step 3: Build the few-shot prompt template
few_shot_prompt = FewShotPromptTemplate(
    examples=examples,
    example_prompt=PromptTemplate(input_variables=["input", "output"], template=example_template),
    prefix="Classify the following customer support tickets into one of the categories: 'Billing
    Issue', 'Technical Problem', or 'General Inquiry'.\n\n",
    suffix="\nTicket: {user_input}\nCategory:",
    input_variables=["user_input"],
)
```

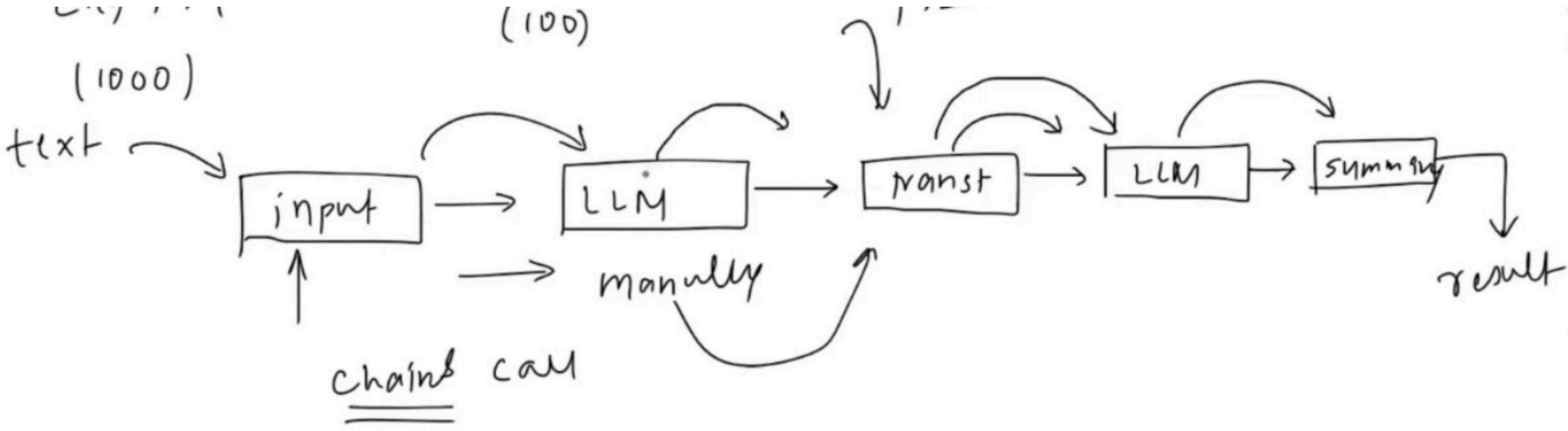

Search Notebooks
Recap
LangChain Comp...
Models
Prompts
Chains
Indexes
Memory
Agents
Issue', 'Technical Problem', or 'General Inquiry'.\n\n",
suffix="\nTicket: {user_input}\nCategory:",
input_variables=["user_input"],
)
Classify the following customer support tickets into one of the categories: 'Billing Issue', 'Technical Problem', or 'General Inquiry'.
Ticket: I was charged twice for my subscription this month.
Category: Billing Issue
Ticket: The app crashes every time I try to log in.
Category: Technical Problem
Ticket: Can you explain how to upgrade my plan?
Category: General Inquiry
Ticket: I need a refund for a payment I didn't authorize.
Category: Billing Issue
Ticket: I am unable to connect to the internet using your service.
Category:

Pipelines → LLM + Pipeline

input
Eng text
(1000)
→ Hindi summary
(100)

pipeline

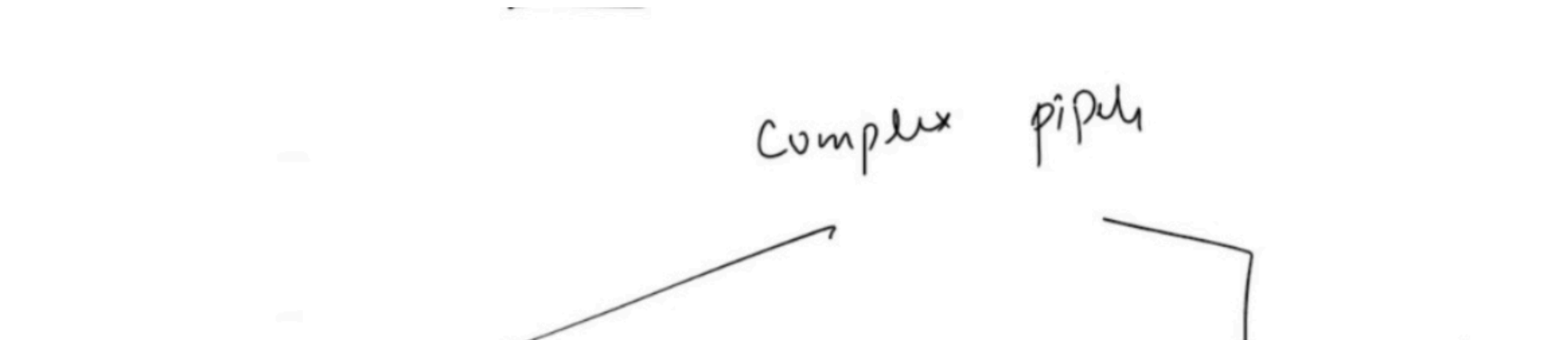



(100)
(1000)
text
input
LLM
manully
chains call
LLM
LLM
result

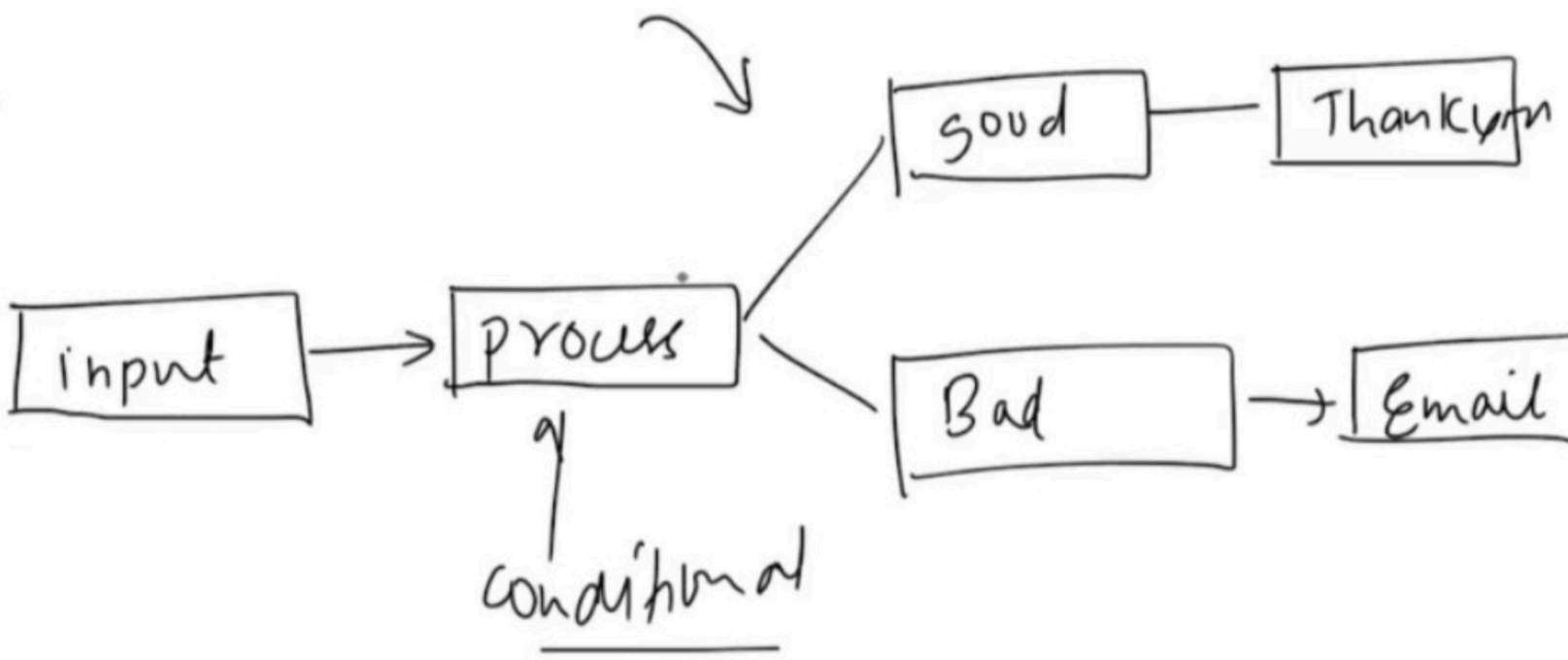
Complex pipeline

Conditional chains

Parallel chain



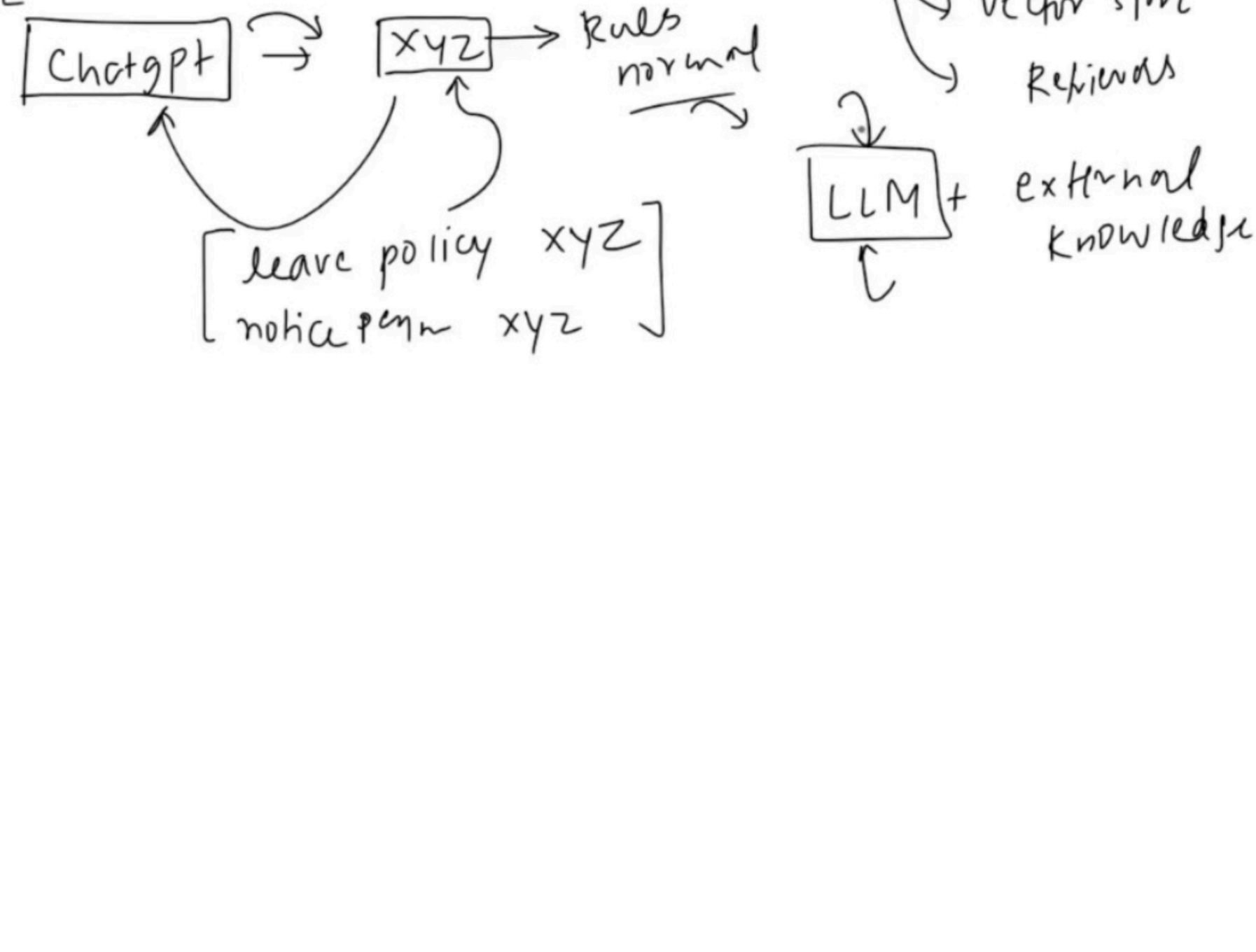
AI Agent feedback



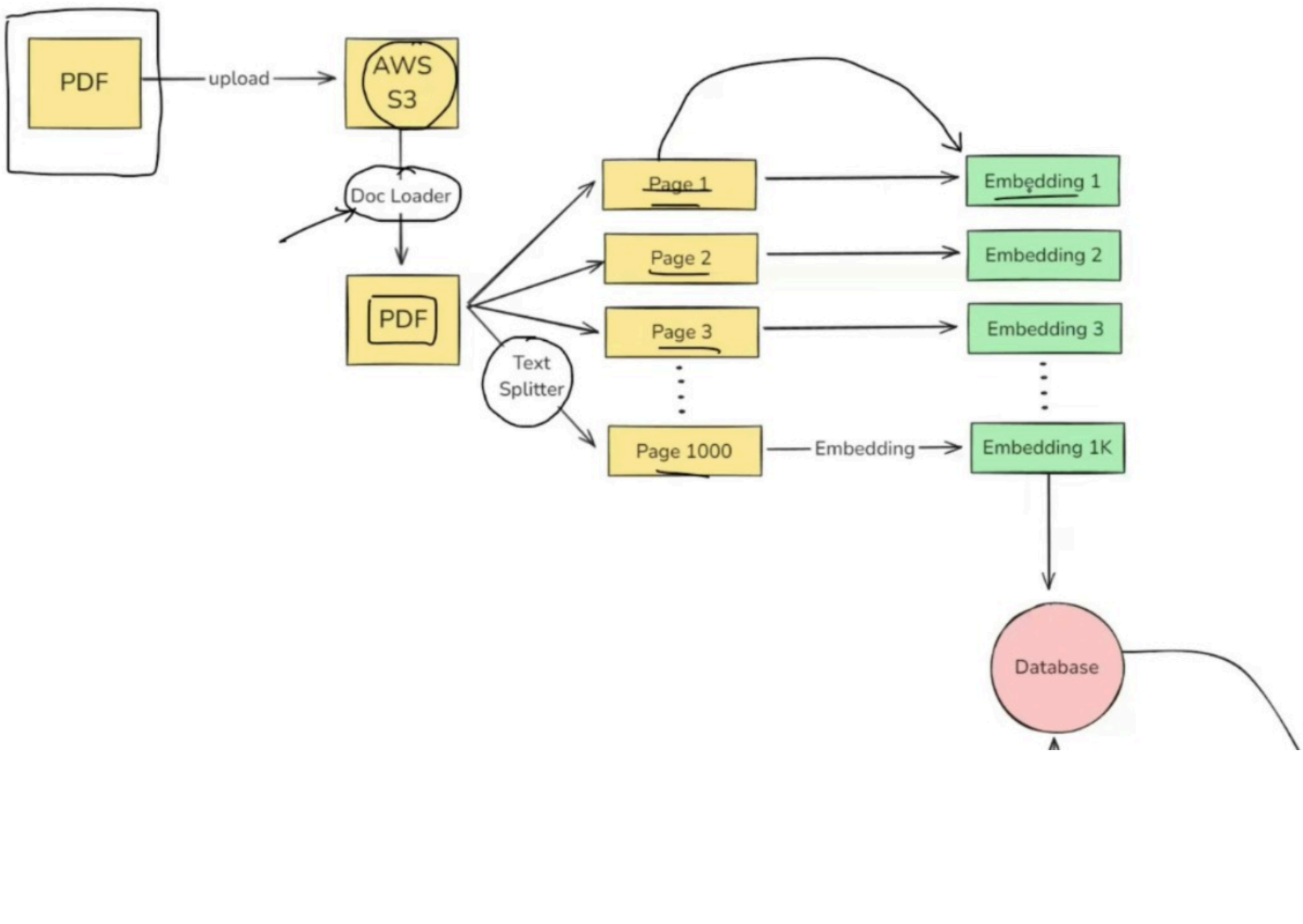
Indexes

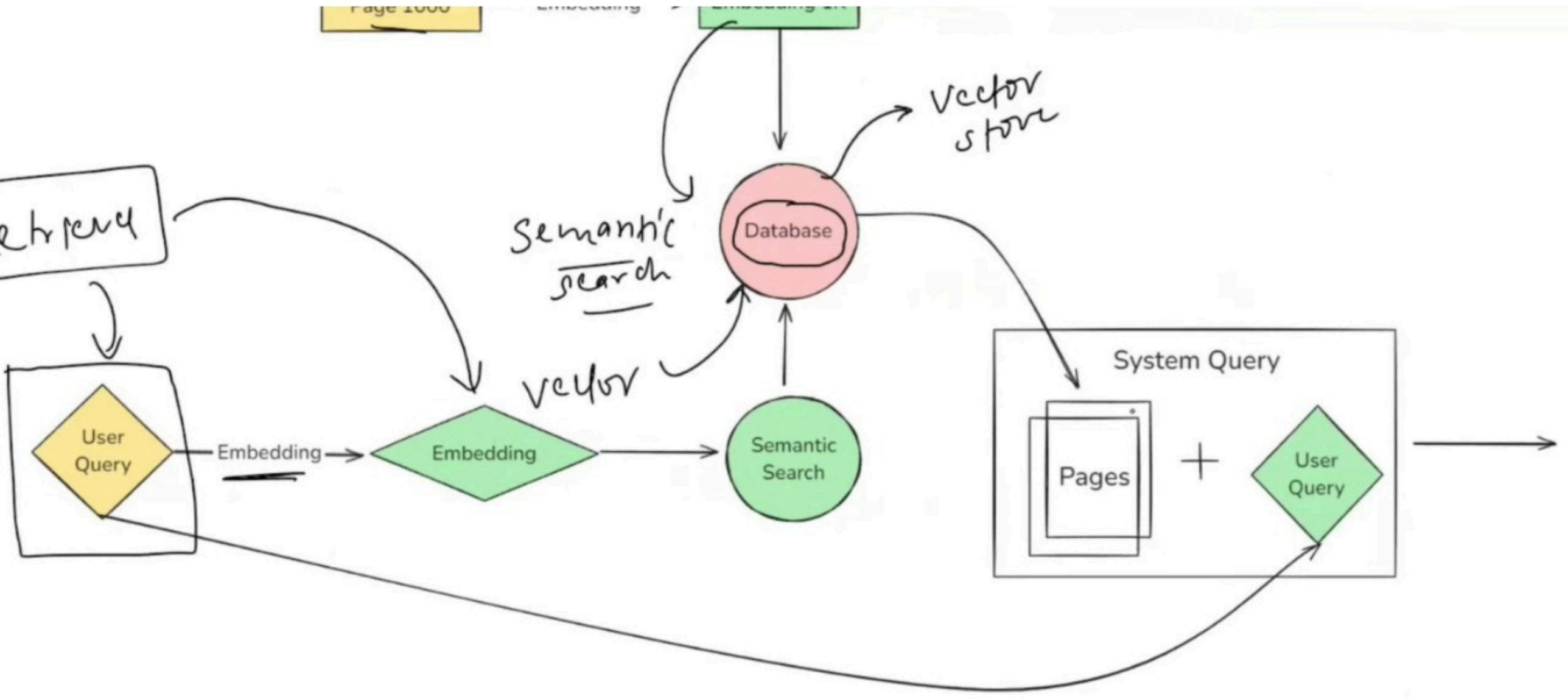
23 January 2025 10:31

Indexes connect your application to external knowledge—such as PDFs, websites or databases



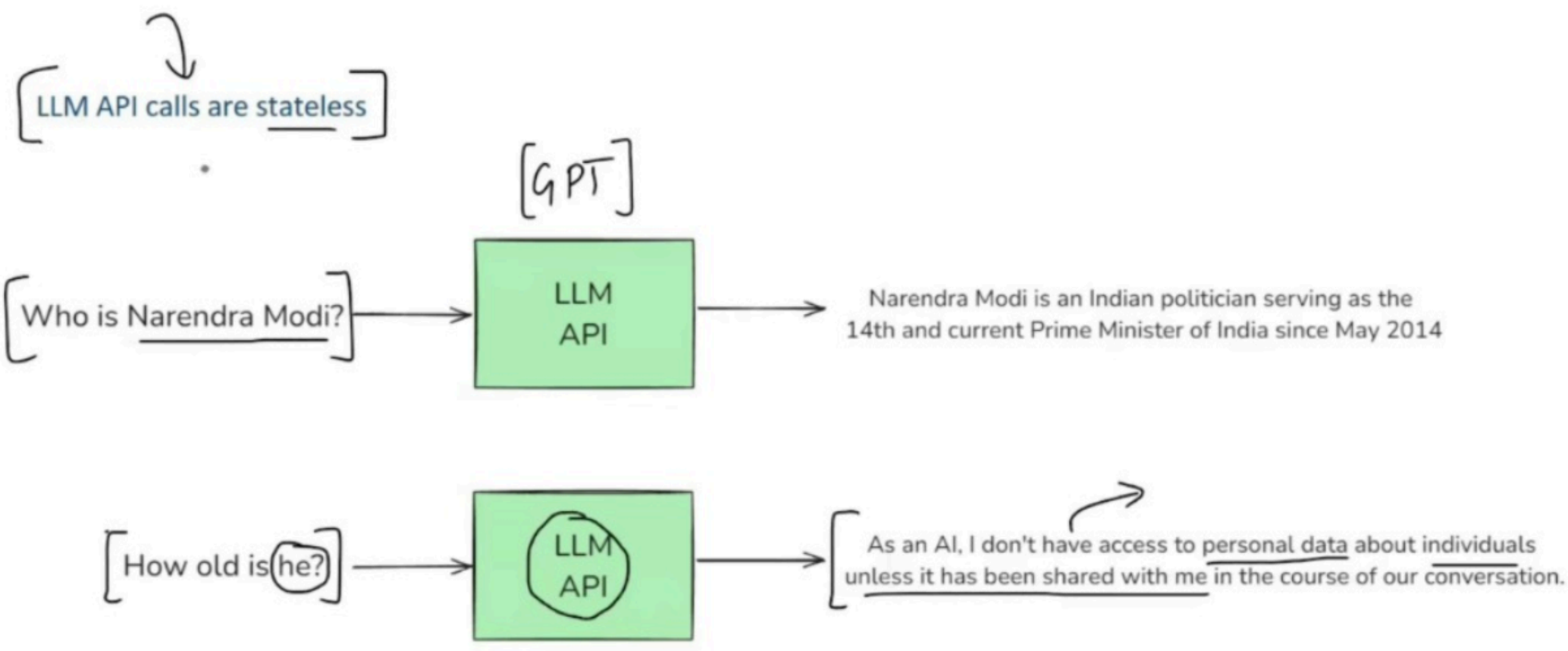
Rulebook 1000 pages




Page 1000
Embedding
Embedding API
Vector store
Database
Semantic search
vector
retrieval
User Query
Embedding
Embedding
Semantic Search
System Query
Pages
+
User Query

Memory

23 January 2025 10:31



- ConversationBufferMemory: Stores a transcript of recent messages. Great for short chats but can grow large quickly.
- ConversationBufferWindowMemory: Only keeps the last N interactions to avoid excessive token usage.
- Summarizer-Based Memory: Periodically summarizes older chat segments to keep a condensed memory footprint.
- Custom Memory: For advanced use cases, you can store specialized state (e.g., the user's preferences or key facts about them) in a custom memory class.