Prompts

```
from langchain_core.prompts import PromptTemplate,load_prompt
```

Inputs:

```
paper_input = st.selectbox( "Select Research Paper Name", ["What Is It Like t
o Be a Bat? (Nagel)","Facing Up to the Problem of Consciousness (Chalmer
s)","On Sense and Reference (Frege)", "On Denoting (Russell)", "Attention Is Al
l You Need", "BERT: Pre-training of Deep Bidirectional Transformers", "GPT-3:
Language Models are Few-Shot Learners", "Diffusion Models Beat GANs on I
mage Synthesis"] )

style_input = st.selectbox( "Select Explanation Style", ["Beginner-Friendly", "T
echnical", "Code-Oriented", "Mathematical"] )

length_input = st.selectbox( "Select Explanation Length", ["Short (1-2 paragra
phs)", "Medium (3-5 paragraphs)", "Long (detailed explanation)"] )
```

Template:

```
template= PromptT emplate(template="""
Please summarize the research paper titled "{paper_input}" with the following
specifications:
Explanation Style: {style_input}
Explanation Length: {length_input}
1. Mathematical Details:
- Include relevant mathematical equations if present in the paper.
- Explain the mathematical concepts using simple, intuitive code snippets whe
re
```

```
applicable.
2. Analogies:
- Use relatable analogies to simplify complex ideas.
If certain information is not available in the paper, respond with: "Insufficient information available" instead of guessing.
Ensure the summary is clear, accurate, and aligned with the provided style an d length.
"""
input_variables= ['paper_input', 'style_input', 'length_input']
)
```

Pass the Input Variable inside **PromptTemplate**

- We have 3 Input variables

Invoke

```
template.invoke({'paper_input':paper_input,
        'style_input':style_input,
        'length_input':length_input})
```

💡 **Remember: Pass the input variables in a dictionary.**

Structure:

```
template= PromptTemplate(template= """
{input1} {input2}
""",
input_variables=['input1', 'input2']
)
```

```
prompt= template.invoke( {
'input1': input1
'input2': input1
}
)
```

```
result = model.invoke(prompt)
```

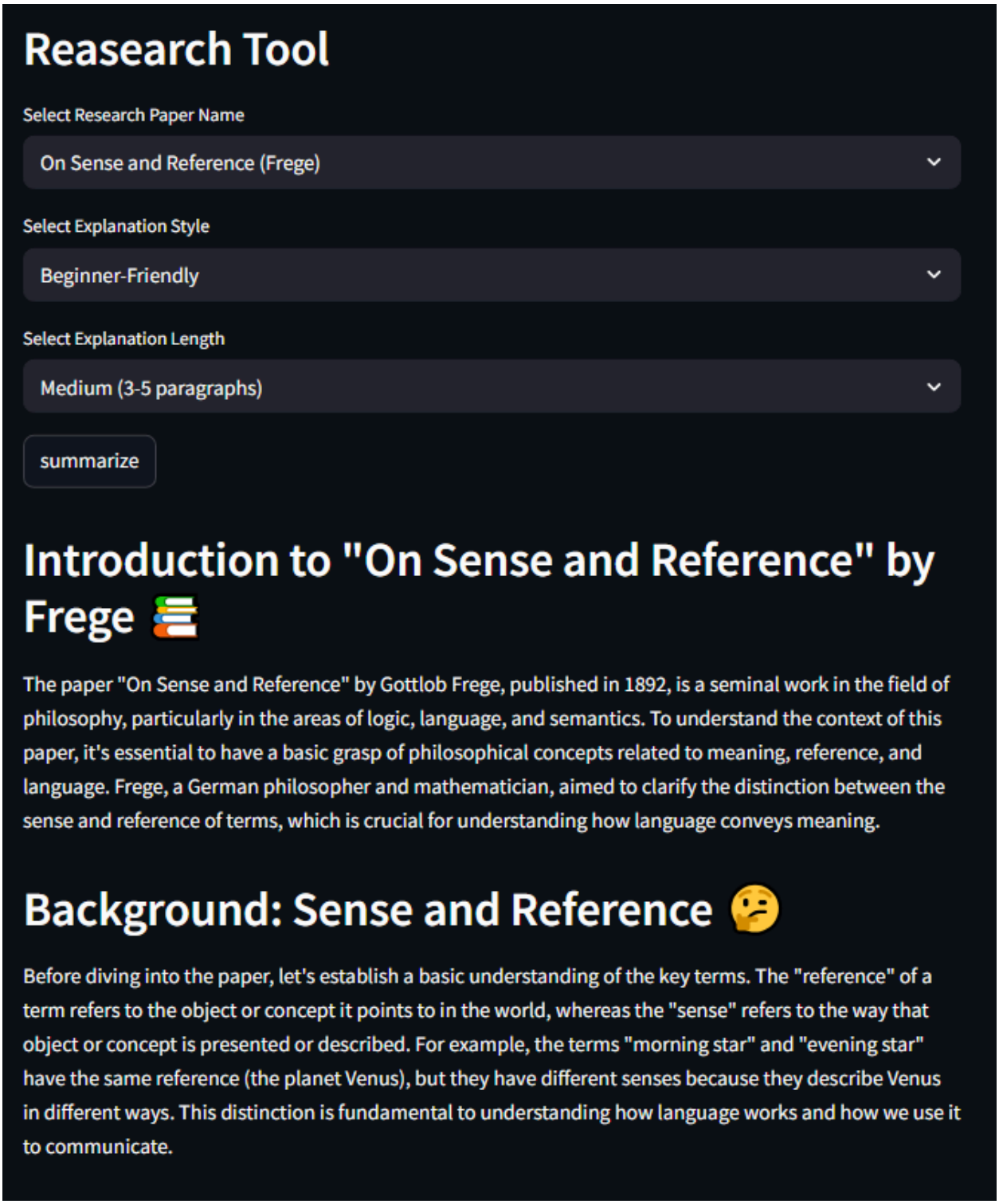
Code for Sreamlit app:

Research_Tool.py

```
from langchain_groq import ChatGroq
from dotenv import load_dotenv
import streamlit as st
from langchain_core.prompts import PromptTemplate,load_prompt

load_dotenv() #for running the code in local environment


st.header('✨Reasearch Tool')

api_key= st.text_input("Enter Groq API Key", type="password")

if not api_key:
    st.error("Please enter the Groq API Key")
```

```
model = ChatGroq(model="llama-3.3-70b-versatile", api_key=api_key, streami
ng=True)

paper_input = st.selectbox( "Select Research Paper Name", ["What Is It Like t
o Be a Bat? (Nagel)","Facing Up to the Problem of Consciousness (Chalmer
s)","On Sense and Reference (Frege)", "On Denoting (Russell)", "Attention Is Al
l You Need", "BERT: Pre-training of Deep Bidirectional Transformers", "GPT-3:
Language Models are Few-Shot Learners", "Diffusion Models Beat GANs on I
mage Synthesis"] )

style_input = st.selectbox( "Select Explanation Style", ["Beginner-Friendly", "T
echnical", "Code-Oriented", "Mathematical"] )

length_input = st.selectbox( "Select Explanation Length", ["Short (1-2 paragra
phs)", "Medium (3-5 paragraphs)", "Long (detailed explanation)"] )


template = PromptTemplate(template="""
Please summarize the research paper titled "{paper_input}" with the following
specifications:
Explanation Style: {style_input}
Explanation Length: {length_input}
1. Mathematical Details:
- Include relevant mathematical equations if present in the paper.
- Explain the mathematical concepts using simple, intuitive code snippets whe
re
applicable.
2. Analogies:
- Use relatable analogies to simplify complex ideas.
If certain information is not available in the paper, respond with: "Insufficient
information available" instead of guessing.
Use headings, subheadings, emojis when necessary.
Give a background to better understand the topic.
Ensure the summary is clear, accurate, and aligned with the provided style an
d length.
""",
```

```
input_variables= ['paper_input', 'style_input', 'length_input']
)


prompt = template.invoke({'paper_input':paper_input,
        'style_input':style_input,
        'length_input':length_input})

summarize= st.button("summarize")

if summarize:
    result = model.invoke(prompt)

    st.write(result.content)
```

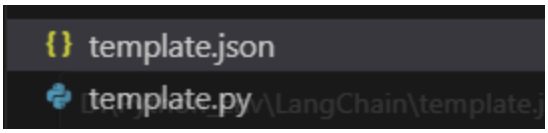


Save Template as JSON File

- Paste the template in another .py file

- `template.save("template.json")`

```
from langchain_core.prompts import PromptTemplate

template = PromptTemplate(template="""
Please summarize the research paper titled "{paper_input}" with the following
specifications:
Explanation Style: {style_input}
Explanation Length: {length_input}
1. Mathematical Details:
- Include relevant mathematical equations if present in the paper.
- Explain the mathematical concepts using simple, intuitive code snippets whe
re
applicable.
2. Analogies:
- Use relatable analogies to simplify complex ideas.
If certain information is not available in the paper, respond with: "Insufficient
information available" instead of guessing.
Use headings, subheadings, emojis when necessary.
Give a background to better understand the topic.
Ensure the summary is clear, accurate, and aligned with the provided style an
d length.
""",
input_variables= ['paper_input', 'style_input', 'length_input']
)

template.save("template.json")
```



Newly created file:

```
{
    "name": null,
    "input_variables": [
        "length_input",
        "paper_input",
        "style_input"
    ],
    "optional_variables": [],
    "output_parser": null,
    "partial_variables": {},
    "metadata": null,
    "tags": null,
    "template": "\nPlease summarize the research paper titled \"{paper_input}\" with the following specifications:\nExplanation Style: {style_input}\nExplanation Length: {length_input}\n1. Mathematical Details:\n- Include relevant mathematical equations if present in the paper.\n- Explain the mathematical concepts using simple, intuitive code snippets where\napplicable.\n2. Analogies:\n- Use relatable analogies to simplify complex ideas.\nIf certain information is not available in the paper, respond with: \"Insufficient\ninformation available\" instead of guessing.\nUse headings, subheadings, emojis when necessary.\nGive a background to better understand the topic.\nEnsure the summary is clear, accurate, and aligned with the provided style and length.\n",
    "template_format": "f-string",
    "validate_template": false,
    "_type": "prompt"
}
```


Load Prompt

```
from langchain_core.prompts import PromptTemplate,load_prompt

template = load_prompt('template.json')
```

🔗Chain:

Remove:

```
prompt = template.invoke({'paper_input':paper_input,
        'style_input':style_input,
        'length_input':length_input})
```

Write:

```
chain = template | model

result = chain.invoke({'paper_input':paper_input,
        'style_input':style_input,
        'length_input':length_input})

result.content
```

💡 Here, we call `.invoke()` **only once**

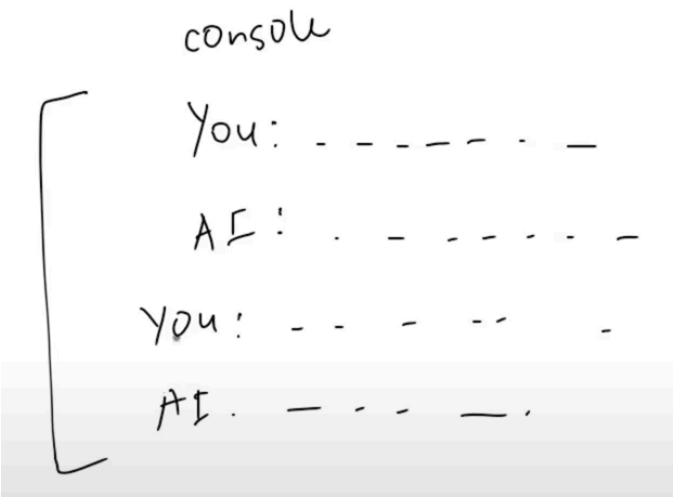
Updated Code with chain:

```
from langchain_groq import ChatGroq
from dotenv import load_dotenv
import streamlit as st
from langchain_core.prompts import PromptTemplate,load_prompt

load_dotenv() #for running the code in local environment


st.header('✨Reasearch Tool')
```

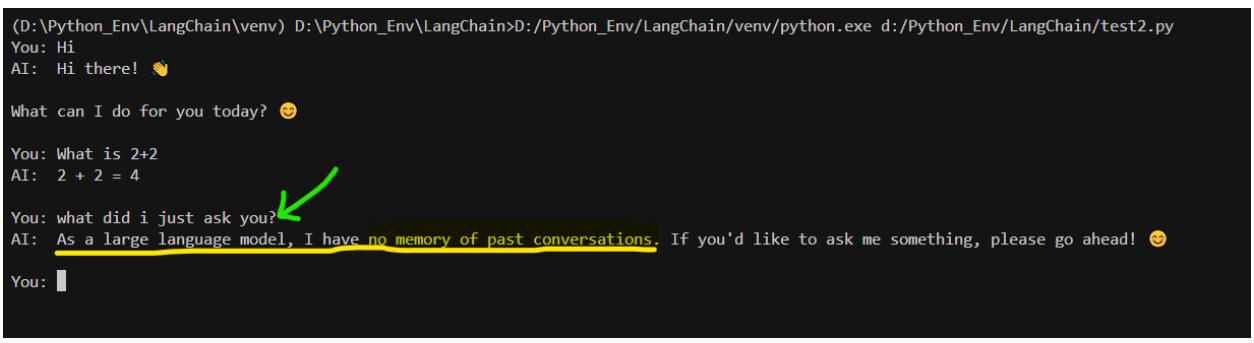
```
api_key= st.text_input("Enter Groq API Key", type="password")

if not api_key:
    st.error("Please enter the Groq API Key")

model = ChatGroq(model="llama-3.3-70b-versatile", api_key=api_key, streami
ng=True)

paper_input = st.selectbox( "Select Research Paper Name", ["What Is It Like t
o Be a Bat? (Nagel)","Facing Up to the Problem of Consciousness (Chalmer
s)","On Sense and Reference (Frege)", "On Denoting (Russell)", "Attention Is Al
l You Need", "BERT: Pre-training of Deep Bidirectional Transformers", "GPT-3:
Language Models are Few-Shot Learners", "Diffusion Models Beat GANs on I
mage Synthesis"] )

style_input = st.selectbox( "Select Explanation Style", ["Beginner-Friendly", "T
echnical", "Code-Oriented", "Mathematical"] )

length_input = st.selectbox( "Select Explanation Length", ["Short (1-2 paragra
phs)", "Medium (3-5 paragraphs)", "Long (detailed explanation)"] )


template = PromptTemplate(template="""
Please summarize the research paper titled "{paper_input}" with the following
specifications:
Explanation Style: {style_input}
Explanation Length: {length_input}
1. Mathematical Details:
- Include relevant mathematical equations if present in the paper.
- Explain the mathematical concepts using simple, intuitive code snippets whe
re
applicable.
2. Analogies:
- Use relatable analogies to simplify complex ideas.
If certain information is not available in the paper, respond with: "Insufficient
information available" instead of guessing.
```

```
Use headings, subheadings, emojis when necessary.
Give a background to better understand the topic.
Ensure the summary is clear, accurate, and aligned with the provided style an
d length.
""",
input_variables= ['paper_input', 'style_input', 'length_input']
)



summarize= st.button("summarize")

if summarize:
    chain = template | model
    result = chain.invoke({'paper_input':paper_input,
        'style_input':style_input,
        'length_input':length_input})

    st.write(result.content)
```

💬Chatbot



```
from langchain_groq import ChatGroq
from dotenv import load_dotenv

load_dotenv()

model = ChatGroq(model= "gemma2-9b-it")

while True:
    user_input = input("You: ").lower()
    if user_input == "exit":
        break
    result = model.invoke(user_input)
    print("AI: ", result.content)
```



- The chatbot does not retain history

Add History to the chatbot:

- Just initiate a blank list and append the new messages (Both You & AI) to the list

```
history = []

while True:
    user_input = input("You: ").lower()
```

```
    history.append(user_input)
    if user_input == "exit":
        break
    result = model.invoke(history)
    history.append(result.content)

    print("AI: ", result.content)
print(history)
```

Entire code:

▼

```
from langchain_groq import ChatGroq
from dotenv import load_dotenv

load_dotenv()

model = ChatGroq(model= "gemma2-9b-it")

history = []

while True:
    user_input = input("You: ").lower()
    history.append(user_input)
    if user_input == "exit":
        break
    result = model.invoke(history)
    history.append(result.content)

    print("AI: ", result.content)
print(history)
```

```
You: hi
AI:  Hello! 👋

How can I help you today? 😊

You: who is PM of india?
AI:  The Prime Minister of India is **Narendra Modi**.

You: what is his height?
AI:  Narendra Modi's height is reported to be around **5 feet 8 inches** (173 cm).


Let me know if you have any other questions! 😊

You:
```

Now it remembers the history

On exit:

```
You: exit
['hey', 'Hey there! What can I do for you? 😊\n', 'what did i ask u', 'You asked "Hey there! What can I do for you? 😊".  \n\nIs there something I can help you with?  \n',
 'good', "I'm glad to hear that! \n\nIs there anything specific I can help you with today?  Perhaps you'd like to:\n\n* **Chat about a topic that interests you?**\n* **Get
 help with writing something?**\n* **Brainstorm ideas?**\n* **Play a text-based game?**\n\nLet me know how I can be of service! 😊  \n", 'exit']
```

> Here, we don't know which message is sent by whom.

Langchain helps you in this.

Types of Messages in Langchain

1. System Message
2. Human Message
3. AI Message

```
from langchain_core.messages import SystemMessage, HumanMessage, AIM
essage
```

- System Message is the one at the very beginning

- Eg. Act as an experienced data scientist

```
messages= [
   SystemMessage(content="You are a genius expert in everything who explai
ns things in simplest way"),
   HumanMessage(content="Tell me about latest langchain version").

result = model.invoke(messages)
```

💡 We are passing these into a list.

- We'll get an AI message as a result

Let's append the AI message in the messages list:

```
messages.append(AIMessage(content=result.content))

print(messages)
```

```
[SystemMessage(content='You are a genius expert in everything who explain
s things in simplest way', additional_kwargs={}, response_metadata={}),

HumanMessage(content='Tell me about latest langchain version', additional_k
wargs={}, response_metadata={}),

AIMessage(content='Imagine LangChain as a toolbox for building AI applicatio
ns.  It helps you connect different AI "tools" – like text generators, search engi
nes, and databases – and put them together in cool and useful ways.\n\nThe l
atest version of LangChain (version 0.0.100) is like getting a whole bunch of n
```

```
ew, shiny tools and upgrades for your toolbox! \n\nHere are some highlights:\n
\n* **Easier to use:** They\'ve made things simpler and more intuitive, so eve
n if you\'re new to AI, you can start building things faster.\n\n* **More powerf
ul connectors:**  LangChain now connects to even more AI services and tool
s, like new chatbots, specialized search engines, and ways to control your ow
n code.\n\n* **Better organization:** They\'ve reorganized the toolbox, makin
g it easier to find what you need and understand how everything works togeth
er.\n\n* **Faster and more efficient:** They\'ve made LangChain run faster an
d use less resources, so you can build bigger and more complex application
s.\n\nEssentially, the latest LangChain is like a supercharged toolbox that mak
es it easier and more powerful to build amazing things with AI!\n', additional_k
wargs={}, response_metadata={})]
```

Integrate this in the chatbot:

```
from langchain_core.messages import SystemMessage, HumanMessage, AIM
essage
from langchain_groq import ChatGroq
from dotenv import load_dotenv

load_dotenv()

model = ChatGroq(model= "gemma2-9b-it")

history= [
   SystemMessage(content="You are a genius expert in everything who explai
ns things in simplest way"),
   ] #👈

while True:
   user_input = input("You: ").lower()
   history.append(HumanMessage(content=user_input)) #👈
   if user_input == "exit":
      break
```

```
    result = model.invoke(history)
    history.append(AIMessage(content=result.content)) #👈

    print("AI: ", result.content)

print(history)
```

- We **appended** the messages as **Human & AIMessage**

💡 **Now every message is labeled.**

Dynamic Messages (*ChatPromptTemplate*)

```
from langchain_core.prompts import ChatPromptTemplate
```

```
chat_template = ChatPromptTemplate([
    ('system', 'You are a helpful {domain} expert'),
    ('human', 'Explain in simple terms, what is {topic}')
])
```

💡 You pass **system** & **human** message in **ChatPromptTemplate.**

> You also add placeholders.

Provide the values of placeholders by invoking the prompt:

```
prompt = chat_template.invoke({'domain':'cricket','topic':'Dusra'})
```

```
print(prompt)
```

Output:

```
messages=[SystemMessage(content='You are a helpful cricket expert', additional_kwargs={}, response_metadata={}),

HumanMessage(content='Explain in simple terms, what is Dusra', additional_kwargs={}, response_metadata={})]
```

💡 Older version code: `ChatPromptTemplate.from_messages`

- It gives the same output

Single message → `PromptTemplate`

Multiple messages → `ChatPromptTemplate`

Generate response:

```
result = model.invoke(prompt)
print(result.content)
```

```
Imagine a cricket bowler throwing a regular off-spinner, curving away from a right-handed batsman. Now, imagine t
 the batsman instead! That's a Dusra!

It's a special type of delivery, a deceptive variation of the off-spinner, where the ball spins in the opposite d

Think of it like a magic trick for bowlers. They make the ball do something unexpected, confusing the batsman and

**Here's the tricky part:** It's really hard to bowl a Dusra well. It takes lots of practice and skill to make th

That's why when a bowler successfully bowls a Dusra, it's a major moment in the game!
```

Message Placeholder

> **Inserts history at runtime.**

- You create a placeholder for your history called message placeholder

Flow:

```
chat tamplate ⟶ Load chat history ⟶ Create prompt
```

Chat Template:

```
chat_template = ChatPromptTemplate([
    ('system','You are a helpful customer support agent'),
    MessagesPlaceholder(variable_name='chat_history'),
    ('human','{query}')
])
```

- To make sense of the previous chat, we'll insert the `Messageplaceholder` in **between system & human message.**

💡 `chat_history` **is a .txt file with all the chat messages** .

Fetch/Load the history from .txt:

```
chat_history = []

# load chat history
with open('chat_history.txt') as f:
    chat_history.extend(f.readlines())
```

Create prompt:

```
prompt = chat_template.invoke({'chat_history':chat_history, 'query':'Where is
my refund'})
```

Entire Code:

```
from langchain_core.prompts import ChatPromptTemplate, MessagesPlacehol
der

# chat template
chat_template = ChatPromptTemplate([
    ('system','You are a helpful customer support agent'),
    MessagesPlaceholder(variable_name='chat_history'),
    ('human','{query}')
])

chat_history = []
# load chat history
with open('chat_history.txt') as f:
    chat_history.extend(f.readlines())

print(chat_history)

# create prompt
prompt = chat_template.invoke({'chat_history':chat_history, 'query':'Where is
my refund'})

print(prompt)
```