# Prompt Engineering

## What are Prompts

- Prompts are the input instructions or queries given to a model to guide its output.

```
from dotenv import load_dotenv

load_dotenv()

#model = ChatOpenAI(model='gpt-4', temperature=1.5)
model = ChatOpenAI(model='gpt-4', temperature=0)
result = model.invoke("Write a 5 line poem on cricket")

print(result.content)
```

## Streamlit UI based application

```python
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv
import streamlit as st
from langchain_core.prompts import PromptTemplate,load_prompt

load_dotenv()


st.header('Reasearch Tool')
user_input=st.text_input('enter your prompt')

if st.button('Summarize'):
    st.text('some random input')
```

# Userinput based application

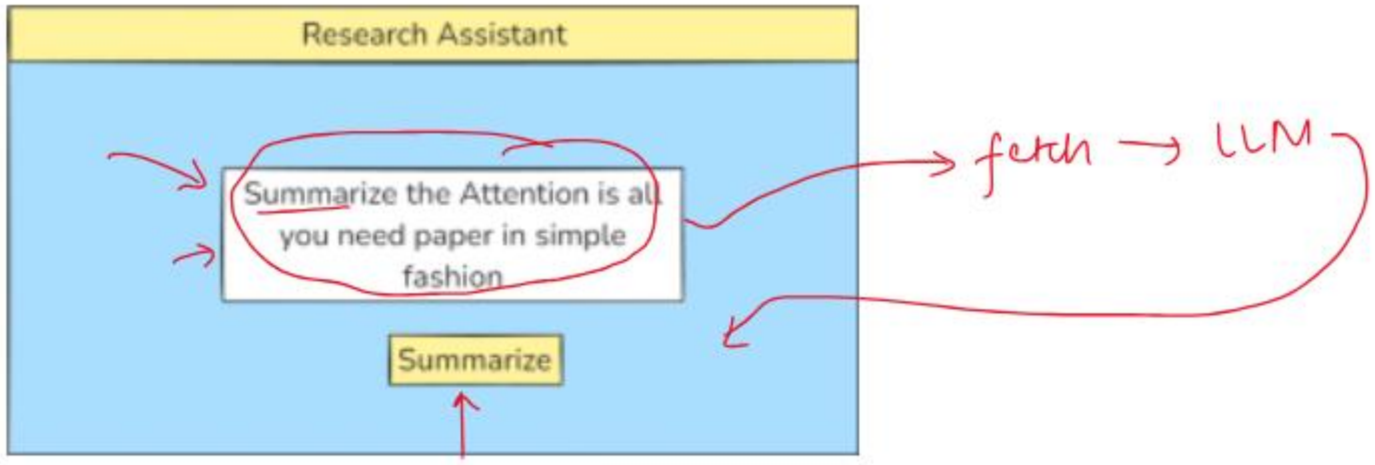
```
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv
import streamlit as st
from langchain_core.prompts import PromptTemplate,load_prompt

load_dotenv()
model = ChatOpenAI()

st.header('Reasearch Tool')
user_input=st.text_input('enter your prompt')

if st.button('Summarize'):
    result = model.invoke(user_input)
    st.write
```

## Static vs Dynamic Prompts


Research Assistant
Summarize the Attention is all you need paper in simple fashion
Summarize
fetch → LLM

A PromptTemplate in LangChain is a structured way to create prompts dynamically by inserting variables into a predefined template. Instead of hardcoding prompts, PromptTemplate allows you to define placeholders that can
be filled in at runtime with different inputs.
This makes it reusable, flexible, and easy to manage, especially when working with dynamic user inputs or automated workflows

# List selection

```
load_dotenv()
model = ChatOpenAI()

st.header('Reasearch Tool')

paper_input = st.selectbox( "Select Research Paper Name",
                           ["Attention Is All You Need",
                            "BERT: Pre-training of Deep Bidirectional Transformers",
                            "GPT-3: Language Models are Few-Shot Learners",
                            "Diffusion Models Beat GANs on Image Synthesis"] )

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

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

if st.button('Summarize'):
    st.write("Hello")
```

## Template based selection

```
load_dotenv()
model = ChatOpenAI()

st.header('Reasearch Tool')
paper_input = st.selectbox( "Select Research Paper Name",
                           ["Attention Is All You Need",
                            "BERT: Pre-training of Deep Bidirectional Transformers",
                            "GPT-3: Language Models are Few-Shot Learners",
                            "Diffusion Models Beat GANs on Image Synthesis"] )

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

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

# Template based selection

```
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 where applicable.

2. Analogies:
  - Use relatable analogies to simplify complex ideas.
  - 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 and length.
""",
input_variables=['paper_input', 'style_input', 'length_input']
)
prompt = template.invoke({
    'paper_input':paper_input,
    'style_input':style_input,
    'length_input':length_input
})

if st.button('Summarize'):
    result = model.invoke(prompt)
    st.write(result.content)
```

## Prompt Generator

A PromptTemplate in LangChain is a structured way to create prompts dynamically by inserting variables into a predefined template. Instead of hardcoding prompts, PromptTemplate allows you to define placeholders that can be filled in at runtime with different inputs.

This makes it reusable, flexible, and easy to manage, especially when working with dynamic user inputs or automated workflows.

## Why use PromptTemplate over f strings?

- Default validation
- Reusable
- LangChain Ecosystem

# Prompt template

```
model = ChatOpenAI()

# detailed way
template2 = PromptTemplate(
    template='Greet this person in 5 languages. The name of the person is {name}',
    input_variables=['name']
)

# fill the values of the placeholders
prompt = template2.invoke({'name':'Perviz hood bahi'})

result = model.invoke(prompt)

print(result.content)
```

## Prompt UI

```python
model = ChatOpenAI()
st.header('Reasearch Tool')
paper_input = st.selectbox( "Select Research Paper Name", ["Attention Is All You Need", "BERT: Pre-training of Deep Bidirectional Transformers", "GPT-3: Language Models are Few-Shot Learners", "Diffusion Models Beat GANs on Image Synthesis"] )
style_input = st.selectbox( "Select Explanation Style", ["Beginner-Friendly", "Technical", "Code-Oriented", "Mathematical"] )
length_input = st.selectbox( "Select Explanation Length", ["Short (1-2 paragraphs)", "Medium (3-5 paragraphs)", "Long (detailed explanation)"] )
template = load_prompt('template.json')
prompt = template.invoke({
    'paper_input':paper_input,
    'style_input':style_input,
    'length_input':length_input
})

if st.button('Summarize'):
    result = model.invoke(prompt)
    st.write(result.content)
```

# Prompt UI

```
model = ChatOpenAI()
st.header('Reasearch Tool')
paper_input = st.selectbox( "Select Research Paper Name", ["Attention Is All You Need",
"BERT: Pre-training of Deep Bidirectional Transformers", "GPT-3: Language Models are Few-Shot
Learners", "Diffusion Models Beat GANs on Image Synthesis"] )
style_input = st.selectbox( "Select Explanation Style", ["Beginner-Friendly", "Technical",
"Code-Oriented", "Mathematical"] )
length_input = st.selectbox( "Select Explanation Length", ["Short (1-2 paragraphs)", "Medium
(3-5 paragraphs)", "Long (detailed explanation)"] )
template = load_prompt('template.json')
if st.button('Summarize'):
    chain = template | model
    result = chain.invoke({
        'paper_input':paper_input,
        'style_input':style_input,
        'length_input':length_input
    })
    st.write(result.content)
```

## Chatbot

```
model = ChatOpenAI()

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

## Chatbot

```python
model = ChatOpenAI()
chat_history = []
while True:
    user_input = input('You: ')
    chat_history.append(user_input)
    if user_input == 'exit':
        break
    result = model.invoke(chat_history)
    chat_history.append(result.content)
    print("AI: ", result.content)

print(chat_history)
```

## Message

- System Message
- Human Message
- AI Message

## Message

```
model = ChatOpenAI()

messages=[
    SystemMessage(content='You are a helpful
assistant'),
    HumanMessage(content='Tell me about
LangChain')
]

result = model.invoke(messages)
messages.append(AIMessage(content=result.conten
t))
print(messages)
```

# Chatbot -message

```python
model = ChatOpenAI()

chat_history = [
    SystemMessage(content='You are a helpful AI assistant')
]

while True:
    user_input = input('You: ')
    chat_history.append(HumanMessage(content=user_input))
    if user_input == 'exit':
        break
    result = model.invoke(chat_history)
    chat_history.append(AIMessage(content=result.content))
    print("AI: ",result.content)

print(chat_history)
```

# Chatprompt template

```
chat_template = ChatPromptTemplate.from_messages([
    SystemMessage(content='You are a helpful {domain} expert'),
    HumanMessage(content='Explain in simple terms, what is
{topic}')
])

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

print(prompt)
```

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

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

print(prompt)
```

# Message-placeholder

```
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)
```