PROMPT

LANGCHAIN

MUKESH KUMAR

What is prompt?

A prompt is the input you give to a language model (like ChatGPT) to guide its response. In simple terms:

A prompt is a question, instruction, or piece of text you give to the AI to get a specific output.

Static Vs Dynamic Prompt

Feature	Static Prompt	Dynamic Prompt
Definition	A **fixed** prompt that does not change	A **generated** or **updated** prompt based on inputs/context
Behaviour	Always the same phrasing and structure	Varies based on user input, state, or logic
Use Case	Simple, one-time instructions or fixed templates	Chatbots, search systems, few-shot examples, personalized tasks
Flexibility	Low	High
Example Use	Command-based systems	Conversational agents, dynamic workflows

Example of Static Prompt

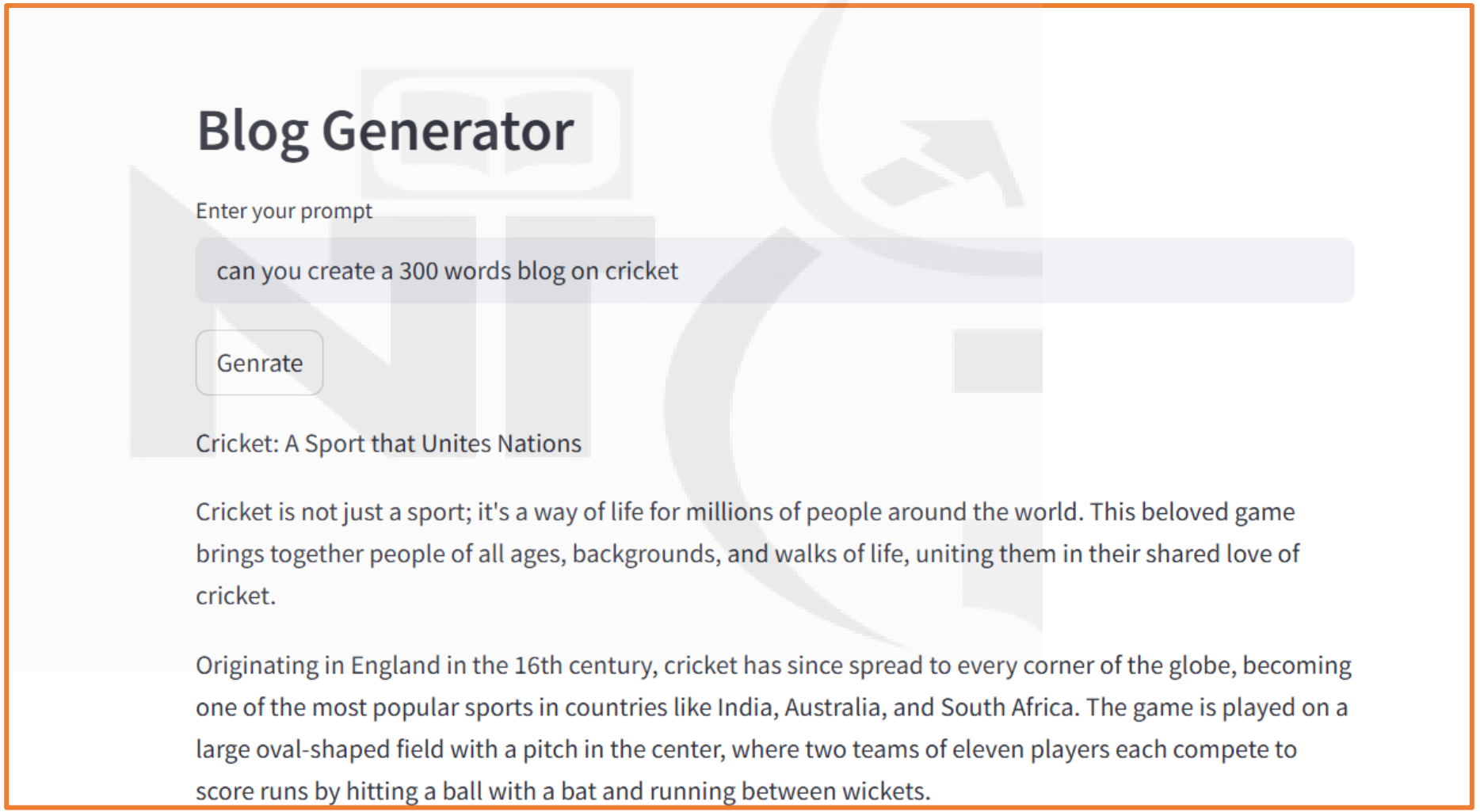
This is a **hardcoded** prompt — it always asks for the same task in the same way

```
prompt = "Translate the following sentence to French: Hello, how are you?"
```

Let's understand this with an example

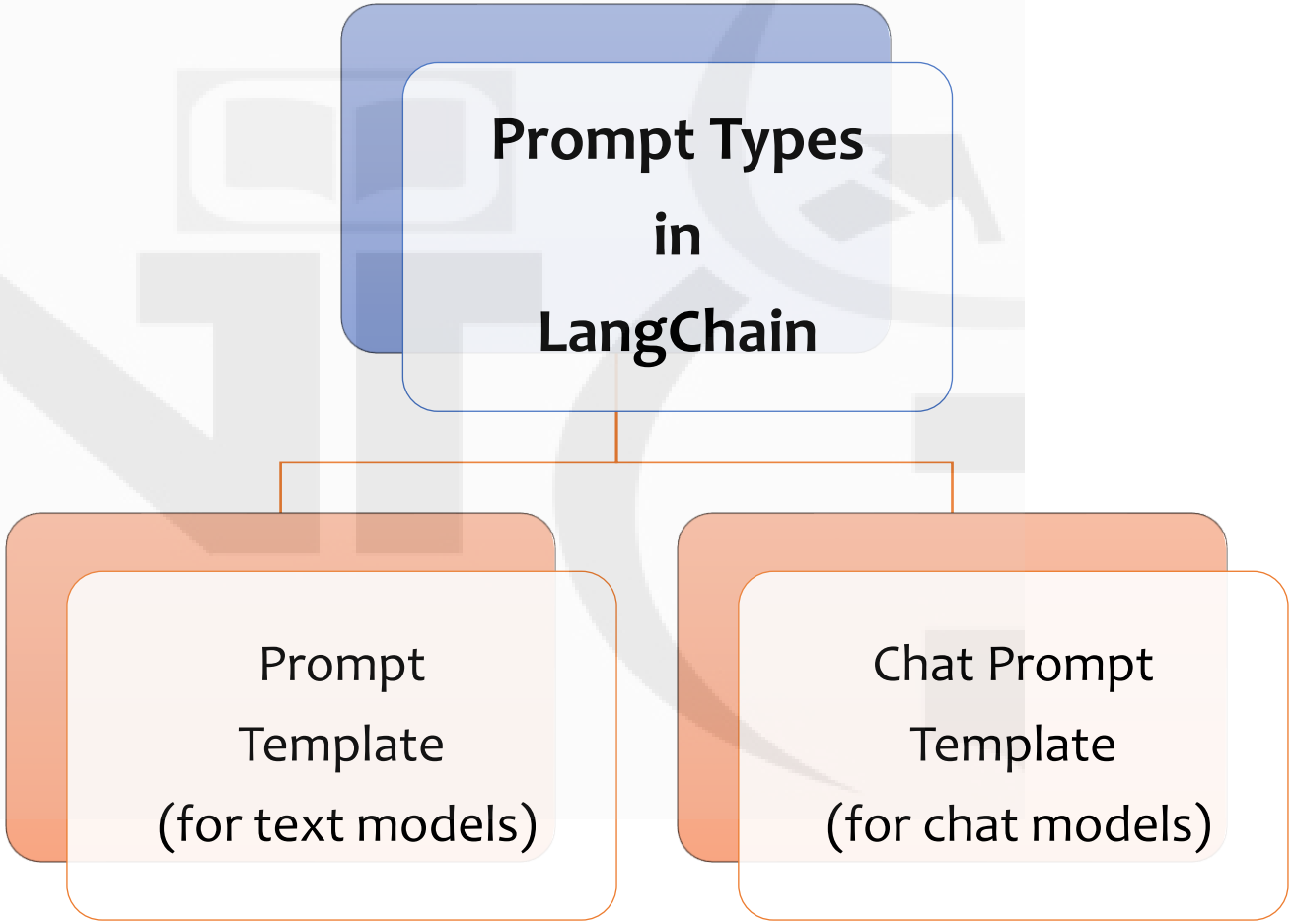
- Let's say you built an app called blog generator
- And user can enter a prompt on your app to generate a blog

Static prompt example



Disadvantages of Static prompt

- User has more control
- If user provides wrong input the ouput of LLM might completely change
- Slight changes in prompts may result in completely different ouput from LLMs


Prompt Types in LangChain
Prompt Template (for text models)
Chat Prompt Template (for chat models)

What is prompt template?

- A PromptTemplate in LangChain is a reusable and structured way to create prompts by filling in variables (like {input} or {context}) into a predefined prompt string.

- It separates the prompt logic from the content, making prompts easier to manage, reuse, and scale.

Prompt Template

```python
from langchain_core.prompts import PromptTemplate

# 1. Define the prompt template
template = PromptTemplate(
    template="Write a short story about a {animal} who learns to {skill}.",
    input_variables=["animal", "skill"],  # These must match the placeholders
    validate_template=True  # Optional but recommended
)
# 2. Fill in the variables using .invoke()
filled_prompt = template.invoke({
    "animal": "cat",
    "skill": "cook"
})
# 3. Print or use the generated prompt string
print(filled_prompt)
```

.invoke() returns a string, ready to send to an LLM.

Final prompt for the Blog Generator App

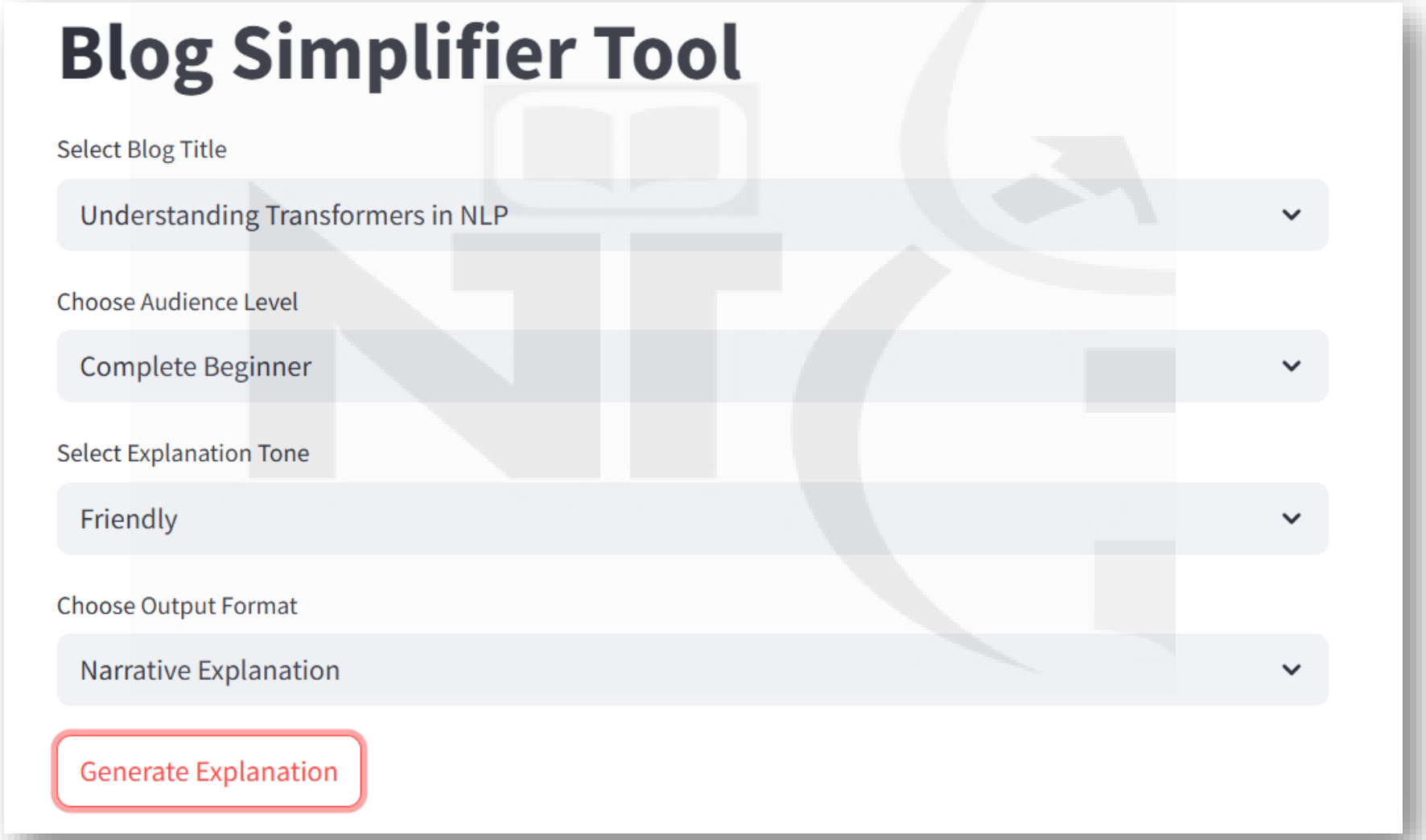
```
# Define the prompt template inline
template = PromptTemplate(
    template="""
You are an expert educator. Convert the blog titled "{blog_title}" into a beginner-friendly explanation.

Audience Level: {audience_level}
Tone: {tone_input}
Target Format: {output_format}

Guidelines:
1. Simplify jargon using everyday language or analogies.
2. Where applicable, include code snippets or diagrams to explain concepts.
3. Highlight real-world applications or use-cases.
4. If certain parts of the blog are ambiguous or missing, clearly state: "Details not provided."

Ensure the explanation is accessible to someone new to the topic and matches the selected tone and format.
""",
    input_variables=["blog_title", "audience_level", "tone_input", "output_format"],
    validate_template=True
)
```

Demo - Blog Generator App



Prompt Template Validation demo

Reusable and readable

```
# Load the saved template
template = load_prompt("explain_blog_template.json")
```

Advantages of PromptTemplate

1. **Reusability**
   – Define once, use with different inputs across multiple tasks.

2. **Clarity & Maintainability**
   – Keeps prompts clean, readable, and separate from code logic.

3. **Dynamic Input Handling**
   – Easily generate custom prompts using user input or data.

4. **Built-in Validation**
   – Catches missing or incorrect variables before runtime errors.

5. **LangChain Ecosystem** – Seamlessly integrates with other LangChain components like Chains, Agents, and Tools.

Demo LangChain Chatbot

Without chat history it becomes difficult for chatbot to track the context about which messages were from which user

MESSAGES

Demo

- To understand Messages , lets build a small chat bot using langchain
- Demo context aware

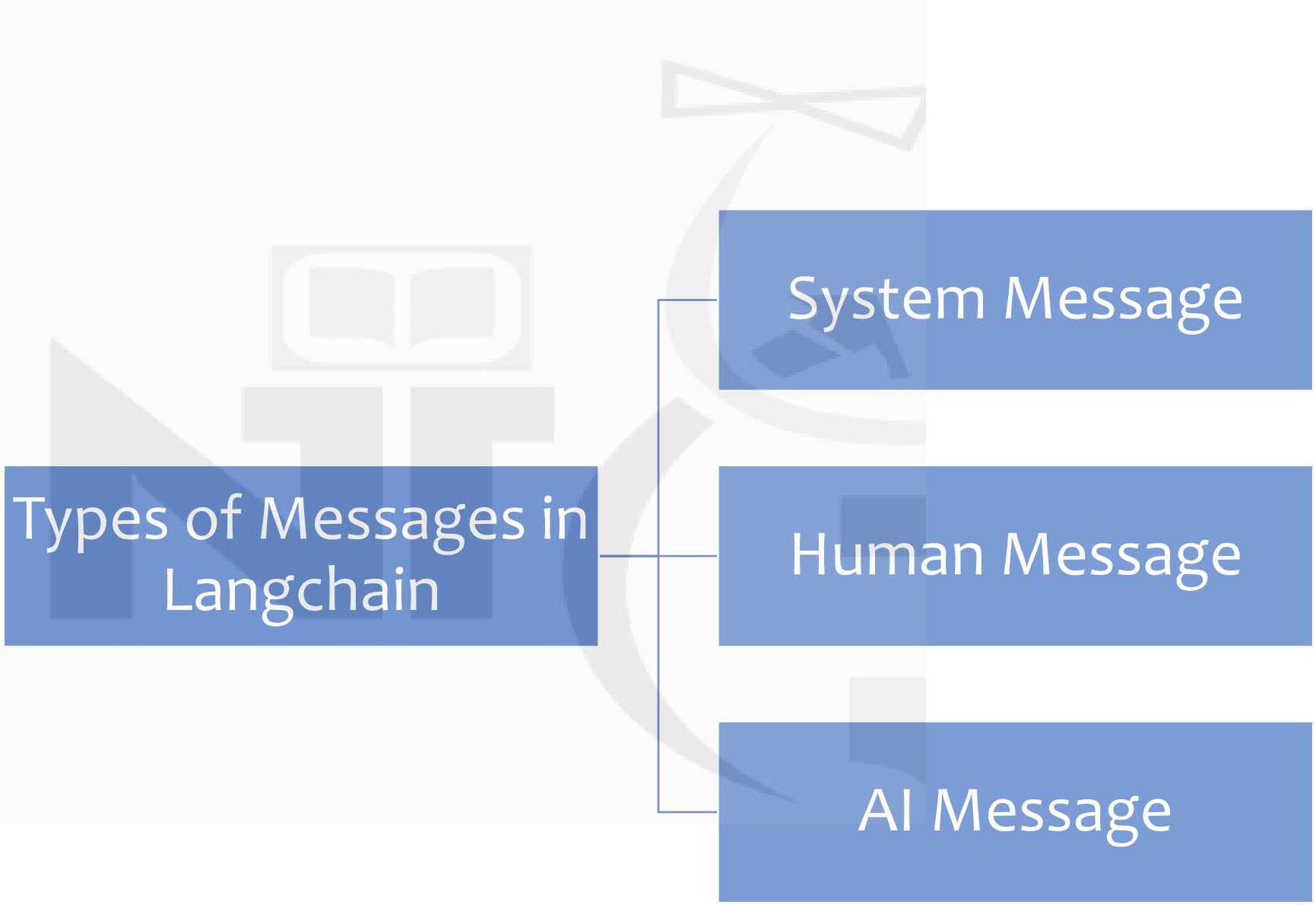
Chatbot history

By default, the chatbot doesn’t tag every message in chat history list.

Without messages tags/types , for longer conversation , it will become difficult for LLM to keep track of which message belong to which user

This is why every message in the chat history must be tagged with message type

LangChain has 3 inbuilt classes for defining different message types


Types of Messages in Langchain
System Message
Human Message
AI Message

```python
from langchain_openai import ChatOpenAI
from langchain_core.messages import SystemMessage, HumanMessage, AIMessage
from dotenv import load_dotenv

load_dotenv()
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)
```

ChatPromptTemplate

- Create dynamic multi-turn messages with (with roles like user, system, AI).
- ChatPromptTemplate lets you build structured prompts for chat models.
- It helps you organize system, user, and assistant messages.
- Use it when working with ChatOpenAI, agents, or anything with chat-based input.

ChatPromptTemplate Examples

ChatPromptTemplate (for chat models)

- Supports structured messages (system, human, AI).

```python
from langchain_core.prompts import ChatPromptTemplate

chat_template = ChatPromptTemplate([
    ('system', 'You are an expert in the field of {field}'),
    ('human', 'Can you break down the concept of {concept} in simple language?')
])

prompt = chat_template.invoke({
    'field': 'machine learning',
    'concept': 'overfitting'
})

print(prompt)
```

```python
from langchain_core.prompts import ChatPromptTemplate

prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a helpful assistant."),
    ("user", "Explain {topic} to a {audience_level} in a {tone} tone.")
])
```

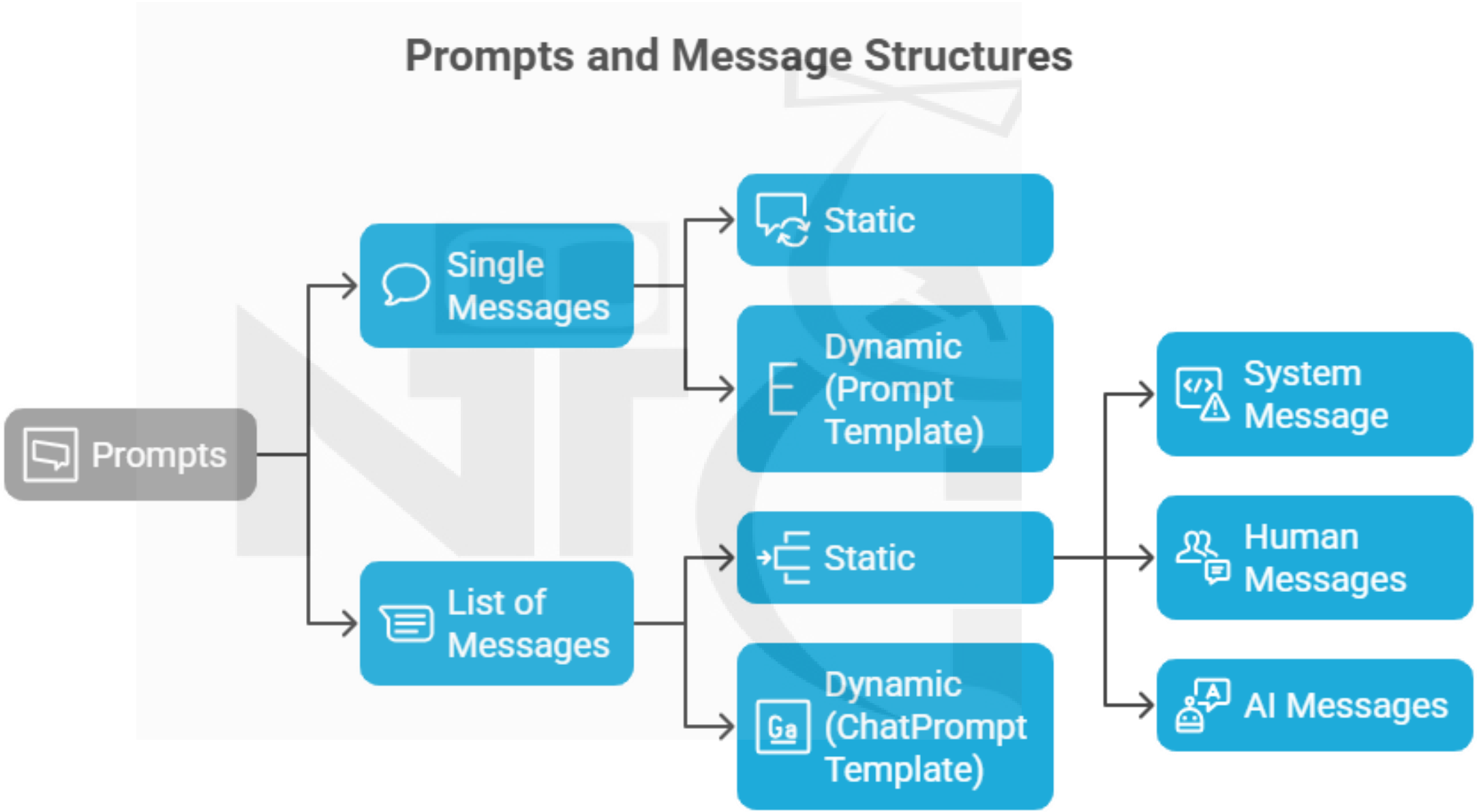
```python
filled_prompt = prompt.invoke({
    "topic": "transformers",
    "audience_level": "beginner",
    "tone": "friendly"
})
```

Message Placeholder

Messages Placeholder is used inside chat prompts to dynamically inject message lists (like chat history).

```python
prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a coding tutor."),
    MessagesPlaceholder("history"),
    ("user", "{question}")
])

# At runtime
formatted = prompt.invoke({
    "history": [
        {"role": "user", "content": "What is a variable?"},
        {"role": "assistant", "content": "A variable stores a value..."}
    ],
    "question": "What is a function?"
})
```


Prompts and Message Structures
Prompts
Single Messages
List of Messages
Static
Dynamic (Prompt Template)
Static
Dynamic (ChatPrompt Template)
System Message
Human Messages
AI Messages