System, AI, and Human Messages in LangChain

Understanding Message Types for Structured Conversations

1 Introduction

LangChain facilitates multi-turn conversational AI by structuring interactions between the system, the user, and the model through three distinct message types:

- **SystemMessage** – defines the behavior, tone, and constraints of the AI.
- **HumanMessage** – represents the user's input or query.
- **AIMessage** – represents the model's response to the human input.

2 System Messages — The Rule Setter

2.1 Definition

A SystemMessage sets the context and behavior for the AI. It defines how the model should think, respond, or behave in the conversation.

```
from langchain_core.messages import SystemMessage

SystemMessage(content="You are a concise, formal academic assistant.")
```

3 Human Messages — The User's Voice

3.1 Definition

A HumanMessage captures what the user says or requests.

```
from langchain_core.messages import HumanMessage

HumanMessage(content="Translate 'Bonjour' to English.")
```

4 AI Messages — The Model's Response

4.1 Definition

An `AIMessage` stores what the model generates.

```
from langchain_core.messages import AIMessage

AIMessage(content="The translation of 'Bonjour' is 'Hello'.")
```

5 Multi-Turn Example

```
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain_core.messages import SystemMessage, HumanMessage, AIMessage

model = ChatGoogleGenerativeAI(model="gemini-2.5-flash")

messages = [
    SystemMessage(content="You are an NLP research assistant."),
    HumanMessage(content="Explain the Transformer architecture.")
]

first_reply = model.invoke(messages)
print(first_reply.content)

messages.append(AIMessage(content=first_reply.content))
messages.append(HumanMessage(content="Summarize it simply."))

second_reply = model.invoke(messages)
print(second_reply.content)
```

6 Messages Placeholder — Dynamic Message Injection

6.1 Definition

A `MessagesPlaceholder` in LangChain is a special placeholder used inside a `ChatPromptTemplate` to dynamically insert chat history or a list of messages at runtime. This enables seamless multi-turn conversation handling without manually managing message lists.

```
from langchain_core.prompts import ChatPromptTemplate, MessagesPlaceholder
from langchain_google_genai import ChatGoogleGenerativeAI

model = ChatGoogleGenerativeAI(model="gemini-2.5-flash")

prompt = ChatPromptTemplate.from_messages([
    ("system", "You are a helpful AI assistant."),
    MessagesPlaceholder(variable_name="chat_history"),
    ("human", "{input}")
])

# Example conversation
chat_history = [
    ("human", "What is LangChain?"),
    ("ai", "LangChain is a framework for building LLM-powered applications
    .")
]

response = prompt | model
result = response.invoke({"chat_history": chat_history, "input": "Summarize
     that briefly."})

print(result.content)
```

6.2 Key Benefits

- **Dynamic History Injection:** Enables runtime insertion of conversation history.
- **Simplified Context Handling:** No need to manually concatenate previous messages.
- **Flexible Integration:** Works seamlessly with conversational chains and memory modules.