LANGCHAIN -Fundamentals

LangChain is an open-source framework for developing applications powered by LLMs.

Why do we need LangChain?

- **Concept of Chains** – Easily connect different components of a project.
- The output of one component becomes the input of the next.
- **Types of chains**: sequential, parallel, and conditional chains.
- **Model-agnostic development** – We only have to change one line of code to use different models (OpenAI, Google, AWS, GCP).
- **Complete ecosystem** – Includes all types of document loaders, various text splitters, embedding models (basically provides support for all components).
- **Memory and state handling** – Provides in-conversation memory.

What can we build using LangChain?

1. Conversational chatbots
2. AI knowledge assistants (chatbots trained on personal or specific data)
3. AI agents (chatbots that can respond and also perform tasks)
4. Workflow automation
5. Summarization and research helpers (helps overcome ChatGPT's content length limitations and allows companies to build their own secure systems)

Other frameworks:

- LlamaIndex
- Haystack

Components of LangChain:

1. Models

2. Prompts
3. Chains
4. Memory
5. Indexes
6. Agents

The Purpose of LangChain

Everyone wanted to create chatbots, but there were two major problems:

1. The system needed to understand language (NLU) and process it (NLP) to generate a context-aware response.
   → LLMs solved this.
2. But LLMs were trained using data from the internet and had billions of parameters (>100 GB), making them too large to run or store on local/company servers.

→ So LLM providers hosted them on the cloud and offered APIs to access them remotely.

However, each provider built different APIs and methods to implement their models. This meant:

- Different code was needed for different LLMs
- Switching or combining LLMs in the same project became difficult
- Each provider had different response formats

This is where LangChain comes in:
It provides a **standardized way to use LLMs**, making it easier to switch between or use multiple LLMs in a single project.

---

1. Models

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

LangChain supports two types of models:

- **Language models** (text input → text output)
- **Embedding models** (text input → embedding output)

The **models component** standardizes how we interact with different LLMs.

---

2. Prompts

Prompts are the inputs provided to an LLM.

LangChain supports:

1. Writing **dynamic & reusable prompts**

2. **Role-based prompts**

3. **Few-shot prompting**

Example:
For a customer support chatbot:

- We show the LLM a few examples of support messages in the format: (input, output)
- Then we create an **example template**
- Finally, we build a **few-shot prompt template** that includes those examples
- When we send a new prompt, we ask:
  “Based on previous examples, tell me what category this new example falls under”

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"],
)
```

This is what the LLM receives in the backend when a new prompt is sent.

```
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:
```

---

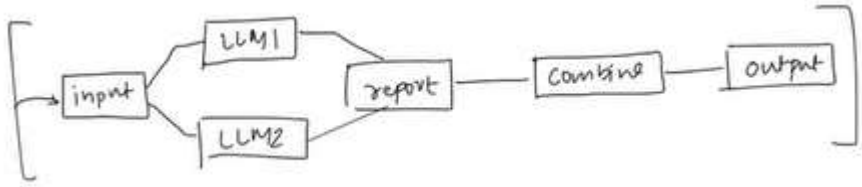
3. Chains

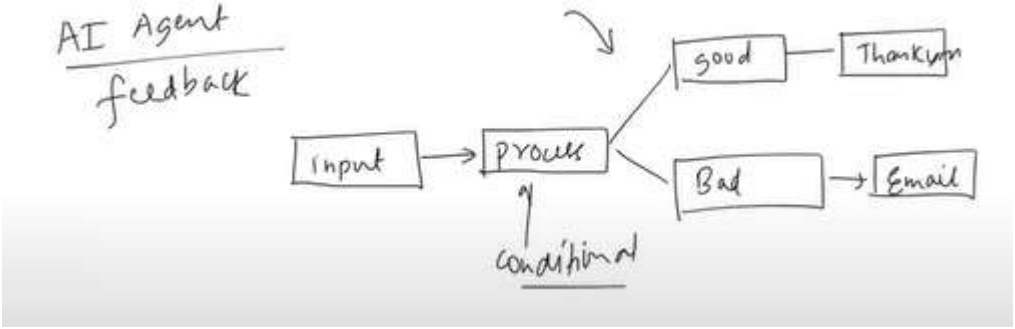
Chains help us create **automated pipelines** where the output of one stage becomes the input of the next.

They allow us to build **complex workflows** easily.

Types of chains:

- **Parallel chain** – e.g., two LLMs generate separate reports and then their outputs are combined

- 

- **Conditional chain** – e.g., customer feedback
    - If feedback is positive → say "Thank you"
    - If feedback is negative → send an email to the support team
    - 

---

4. Indexes

Indexes connect your application to **external knowledge** (PDFs, websites, databases).

This helps us build LLM applications that can access:

- Company policy documents
- Personal data
- Internal knowledge bases

---

5. Memory

LLM API calls are **stateless**—they don't remember previous interactions.

LangChain solves this with **memory components** that maintain context.

Types of memory:

- **ConversationBufferMemory** – Stores recent messages; good for short chats but grows quickly
- **ConversationBufferWindowMemory** – Stores only the last *N* messages to reduce token usage

- **Summarizer-Based Memory** – Summarizes older conversations to keep memory light
- **Custom Memory** – Store specific state like user preferences or facts about them

---

6. AI Agents

Agents are **evolved chatbots that can perform tasks**.

They differ from simple chatbots in two ways:

1. **Reasoning capabilities** (e.g., chain of thought)
2. **Access to tools/APIs** they can use to perform tasks