LANGCHAIN

MUKESH KUMAR

WHAT PROBLEM DOES LLM SOLVE?

OPEN AI Code

```
import openai

openai.api_key = "your-openai-key"

response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": "Summarize what LangChain is in one sentence."}]
)

print(response['choices'][0]['message']['content'])
```

CLAUDE AI Code

```python
import anthropic

client = anthropic.Anthropic(api_key="your-anthropic-key")

response = client.messages.create(
    model="claude-3-haiku-20240307",
    max_tokens=100,
    messages=[{"role": "user", "content": "Summarize what LangChain is in one sentence."}]
)

print(response.content[0].text)
```

Existing Challenges

- Developer will need to write different codes for different models

- It becomes a challenge when:
  - We use multiple LLMs in same application
  - When we want to switch from one LLM provider to another, we will need to re-write the code

What if we could talk to all the LLMs with same code without re-writing the code ????

With LangChain

```python
from langchain.chat_models import ChatOpenAI, ChatAnthropic
from langchain.schema import HumanMessage

prompt = [HumanMessage(content="Summarize what LangChain is in one sentence.")]

# OpenAI model
chat_openai = ChatOpenAI(model_name="gpt-3.5-turbo", api_key="your-openai-key")
print("OpenAI:", chat_openai(prompt).content)

# Anthropic model
chat_claude = ChatAnthropic(model="claude-3-haiku-20240307", api_key="your-anthropic-key")
print("Anthropic:", chat_claude(prompt).content)
```

Benefits:

- Unified message format.
- Easily interchangeable models.
- Works seamlessly with memory, agents, tools.
- Clean, minimal code.

What is LangChain?

- LangChain is an open-source framework for developing applications powered by LLMs.
- Designed to integrate with external data sources and tools.
- Focus on chaining multiple components to enable advanced reasoning.

Why Use LangChain?

- Simplifies development of LLM-based apps.
- Modular and composable architecture.
- Integrates with memory, tools, databases, APIs, and more.
- Encourages production-ready use cases.

Multi-Turn Conversation Handling

Problem:

Maintaining context across messages (e.g., remembering previous turns) is messy with raw API calls.

Solution:

- Chat models make it easy to handle **conversational history** using message lists. LangChain also supports **Memory** modules that plug into chat chains.

Model-Agnostic Code

Problem:

Switching providers (e.g., from OpenAI to Anthropic) requires changing APIs and payload formats.

Solution:

- LangChain's chat interface lets you **swap models easily** by changing one line:

Seamless Integration with Agents and Chains

Problem:

Agents and Chains need structured interaction with LLMs, not just raw strings.

Solution:

- Chat interfaces support structured messaging, enabling them to plug into tools, memory, and logic flows more easily than plain LLMs.

LangChain Core Components

Models : LLMs

Prompts: Generate text responses.

Chains: Sequence LLM calls and logic.

Agents: Use tools with decision-making.

Memory: Store conversation history or knowledge.

Tools: External APIs, search engines, Python REPL, etc.

Popular Use Cases

- Chatbots and Virtual Assistants.
- Document Q&A (PDFs, Notion, Google Docs).
- Code generation and debugging.
- Autonomous agents (e.g., BabyAGI).
- Data Augmented Generation (RAG).

Agents and Tools

- Agents make decisions on what actions to take.
- Use tools like:
  - Google Search
  - Wolfram Alpha
  - Calculator
- Demo code snippet (optional).

Integrations

- LangChain supports:
  - OpenAI, Anthropic, Hugging Face, etc.
  - Vector databases: Pinecone, FAISS, Weaviate.
  - Document loaders: PDFs, websites, APIs.
  - UIs: Streamlit, Gradio, LangServe.

Real-World Applications

- Autonomous Research Agents
- AI Coding Assistants
- Legal Document Analysis
- Personalized Learning Tools