**Building a Simple LLM Application with LangChain**

**1. Overview**

This tutorial covers how to build a simple LLM-based translation application using LangChain. The key concepts include:

* Using Language Models (LLMs)
* Using PromptTemplates and OutputParsers
* Chaining components using LangChain Expression Language (LCEL)
* Debugging and tracing with LangSmith
* Deploying the application using LangServe

**2. Setup**

**Jupyter Notebook**

* Jupyter notebooks are recommended for testing LangChain applications interactively.
* Install Jupyter using:

bash

CopyEdit

pip install notebook

**Installation of LangChain**

* Install LangChain using:

bash

CopyEdit

pip install langchain

* Install LangServe for deploying applications:

bash

CopyEdit

pip install "langserve[all]"

**Setting Up LangSmith for Debugging**

* LangSmith helps in debugging complex chains.
* Set environment variables:

python

CopyEdit

import os

os.environ["LANGCHAIN\_TRACING\_V2"] = "true"

os.environ["LANGCHAIN\_API\_KEY"] = "your\_api\_key"

**3. Using Language Models**

**Supported LLM Providers**

LangChain supports various providers like:

* OpenAI
* Groq
* Cohere
* MistralAI
* NVIDIA AI

**Using Groq's Chat Model**

* Install the required package:

bash

CopyEdit

pip install langchain-groq

* Import and initialize the model:

python

CopyEdit

from langchain\_groq import ChatGroq

model = ChatGroq(model="llama3-8b-8192")

* Example of invoking the model:

python

CopyEdit

from langchain\_core.messages import HumanMessage, SystemMessage

messages = [

SystemMessage(content="Translate the following from English into Italian"),

HumanMessage(content="hi!"),

]

response = model.invoke(messages)

print(response)

**4. Output Parsers**

By default, the model returns an AIMessage containing metadata. To extract only the text output, use StrOutputParser:

python

CopyEdit

from langchain\_core.output\_parsers import StrOutputParser

parser = StrOutputParser()

parsed\_output = parser.invoke(response)

* **Chaining the Model and Parser:**

python

CopyEdit

chain = model | parser

chain.invoke(messages)

**5. Prompt Templates**

**Purpose**

* Format user input into a structured prompt before passing it to the model.
* Example of a prompt template for translation:

python

CopyEdit

from langchain\_core.prompts import ChatPromptTemplate

system\_template = "Translate the following into {language}:"

prompt\_template = ChatPromptTemplate.from\_messages([

('system', system\_template),

('user', '{text}')

])

* Example usage:

python

CopyEdit

prompt\_input = {"language": "Italian", "text": "hi"}

result = prompt\_template.invoke(prompt\_input)

print(result.to\_messages()) # Outputs structured messages

**6. Chaining Components with LangChain Expression Language (LCEL)**

LCEL allows chaining components efficiently using the | operator:

python

CopyEdit

chain = prompt\_template | model | parser

chain.invoke({"language": "Italian", "text": "hi"})

* **Benefits of LCEL**:
  + Optimized execution
  + Better observability via LangSmith

**7. Deploying the Application with LangServe**

**Setting Up the FastAPI Server**

* Install FastAPI and Uvicorn:

bash

CopyEdit

pip install fastapi uvicorn

* Create a serve.py file with the following logic:

python

CopyEdit

from fastapi import FastAPI

from langchain\_core.prompts import ChatPromptTemplate

from langchain\_core.output\_parsers import StrOutputParser

from langchain\_groq import ChatGroq

from langserve import add\_routes

import os

from fastapi.middleware.cors import CORSMiddleware

# Set API Key

os.environ["GROQ\_API\_KEY"] = "your\_api\_key"

# Define Prompt Template

system\_template = "Translate the following into {language}:"

prompt\_template = ChatPromptTemplate.from\_messages([

('system', system\_template),

('user', '{text}')

])

# Define Model

model = ChatGroq(model="llama3-8b-8192")

# Define Parser

parser = StrOutputParser()

# Create Chain

chain = prompt\_template | model | parser

# Initialize FastAPI App

app = FastAPI(

title="LangChain Server",

version="1.0",

description="A simple API server using LangChain's Runnable interfaces"

)

# Enable CORS

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], # Change this for security

allow\_credentials=True,

allow\_methods=["\*"],

allow\_headers=["\*"],

)

# Add Chain Route

add\_routes(app, chain, path="/chain")

if \_\_name\_\_ == "\_\_main\_\_":

import uvicorn

uvicorn.run(app, host="localhost", port=8000)

**Running the Server**

Run the application:

bash

CopyEdit

python serve.py

* Access API at: http://localhost:8000/chain
* Test using a browser or Postman.

**Using the LangServe Playground**

* Open: http://localhost:8000/chain/playground/
* Input:

json

CopyEdit

{

"language": "italian",

"text": "hi"

}

* Expected Output: "Ciao!"

**8. Client-Side API Calls**

A client can call the API using langserve.RemoteRunnable:

python

CopyEdit

from langserve import RemoteRunnable

remote\_chain = RemoteRunnable("http://localhost:8000/chain/")

response = remote\_chain.invoke({"language": "italian", "text": "hi"})

print(response) # Expected Output: "Ciao!"

**9. Key Takeaways**

✅ **LLMs in LangChain:** Supports various providers (Groq, OpenAI, etc.).  
✅ **PromptTemplates:** Help structure user input for models.  
✅ **Output Parsers:** Extract structured text output from model responses.  
✅ **LCEL:** Simplifies chaining components for efficiency.  
✅ **LangServe:** Deploys LangChain applications as REST APIs using FastAPI.  
✅ **LangSmith:** Helps debug and trace execution of chains.