# LANG-CHAIN 

A comprehensive learning repository for **LangChain** - the framework for building applications with Large Language Models (LLMs). This repository contains practical examples, tutorials, and projects demonstrating various LangChain components and integrations.

## Table of Contents



## ✨ Features

- **Multiple LLM Integrations**: OpenAI, Anthropic Claude, Google Gemini, Hugging Face, Groq
- **RAG Implementation**: Complete Retrieval-Augmented Generation pipeline
- **Document Processing**: Support for PDFs, CSVs, text files, and web scraping
- **Output Parsing**: Structured output handling with JSON parsers
- **Prompt Engineering**: Advanced prompt templates and techniques
- **AI Agents**: Intelligent agent implementations
- **Real-world Projects**: Practical applications including chatbots and email generators

## Repository Structure

```
LANG-CHAIN/
├── 1.Models-LLMS/              # Basic LLM implementations
├── 2.Models-ChatModel/         # Chat model integrations
│   ├── 1_openai.py            # OpenAI GPT models
│   ├── 2.antropic_claude.py   # Anthropic Claude
│   ├── 3_gemini.py            # Google Gemini
│   ├── 4_chat_hf.py           # Hugging Face models
│   ├── 5_hf_loca.py           # Local HF models
│   └── 6_groq-api.py          # Groq API integration
├── 3.Models-EmbeddingModel/    # Text embedding models
├── 4.Prompts/                  # Prompt engineering examples
```

```
├── 5.Structure_ouput/          # Structured output handling
├── 6.OutputParser/             # Output parsing techniques
├── 7.Chain/                    # LangChain chain implementations
├── 8.Runnable/                 # Runnable interface examples
├── AI_AGENT/                   # AI agent implementations
├── Projects/                   # Real-world applications
│   ├── email_generator.ipynb  # Cold email generation tool
│   └── 1_Building_Chatbot/    # Chatbot implementation
├── RAG-1.DocumentLoader/       # Document loading utilities
├── RAG-2.TextSplitter/         # Text splitting strategies
├── RAG-3.VectorStore/          # Vector database integration
├── RAG-4.Retriever/            # Document retrieval systems
├── RAG-BuildingSystem/         # Complete RAG pipeline
├── Document_Similarlity.py     # Document similarity calculator
├── requirements.txt            # Python dependencies
└── .env                        # Environment variables
```

## Installation

### Prerequisites

- Python 3.8+
- Virtual environment (recommended)

### Setup

1. **Clone the repository**

```
git clone https://github.com/yashjhota/LANG-CHAIN.git
cd LANG-CHAIN
```

2. **Create and activate virtual environment**

```
python -m venv LC
# Windows
LC\Scripts\Activate
# macOS/Linux
source LC/bin/activate
```

3. **Install dependencies**

```
pip install -r requirements.txt
```

4. **Configure environment variables**
   - Copy `.env` file and add your API keys:

```
OPENAI_API_KEY="your_openai_api_key"
ANTHROPIC_API_KEY="your_anthropic_api_key"
GOOGLE_API_KEY="your_google_api_key"
HUGGINGFACEHUB_ACCESS_TOKEN="your_hf_token"
```

```
GROQ_API_KEY="your_groq_api_key"
PINECONE_API_KEY="your_pinecone_api_key"
```

## Getting Started

### Quick Example: Document Similarity

```
from langchain_huggingface import HuggingFaceEmbeddings
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

# Initialize embedding model
embedding = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2"
)

# Sample documents
documents = [
    "Virat Kohli is an Indian cricketer known for his aggressive batting.",
    "MS Dhoni is a former Indian captain famous for his calm demeanor.",
    "Sachin Tendulkar holds many batting records."
]

query = "tell me about MS Dhoni"

# Calculate similarities
doc_embeddings = embedding.embed_documents(documents)
query_embedding = embedding.embed_query(query)
scores = cosine_similarity([query_embedding], doc_embeddings)[0]

# Find most similar document
best_match_idx = np.argmax(scores)
print(f"Most similar: {documents[best_match_idx]}")
```

### Basic Chat Model Usage

```
from langchain_openai import ChatOpenAI
from dotenv import load_dotenv

load_dotenv()

model = ChatOpenAI(model='gpt-4')
response = model.invoke("What is the capital of France?")
print(response.content)
```

## Core Components

### 1. Language Models

- **LLMs**: Basic language model implementations
- **Chat Models**: Conversational AI models with multiple provider support
- **Embedding Models**: Text-to-vector conversion for semantic search

### 2. RAG (Retrieval-Augmented Generation)

- **Document Loaders**: Text, PDF, CSV, and web content loaders
- **Text Splitters**: Intelligent document chunking strategies
- **Vector Stores**: Efficient similarity search databases
- **Retrievers**: Advanced document retrieval systems

### 3. Prompt Engineering

- Template-based prompts
- Few-shot learning examples
- Chain-of-thought prompting
- Dynamic prompt generation

### 4. Output Processing

- JSON output parsers
- Structured data extraction
- Custom output formatters

## Projects

### Email Generator

**Location**: `Projects/email_generator.ipynb`

An intelligent cold email generator that:

- Scrapes job postings from career websites
- Extracts job requirements using LLMs
- Generates personalized business development emails
- Includes relevant portfolio links

**Features**:

- Web scraping with BeautifulSoup integration

- JSON output parsing for structured data
- Template-based email generation
- Portfolio matching algorithm

### Chatbot Implementation

**Location**: `Projects/1_Building_Chatbot/`

A conversational AI system with:

- Multi-turn conversation handling
- Context awareness
- Integration with multiple LLM providers

### Supported Models

| Provider | Models | Integration File |
|---|---|---|
| **OpenAI** | GPT-4, GPT-3.5-turbo, GPT-4o-mini | `2.Models-ChatModel/1_openai.py` |
| **Anthropic** | Claude-3, Claude-2 | `2.Models-ChatModel/2.antropic_claude.py` |
| **Google** | Gemini Pro, PaLM | `2.Models-ChatModel/3_gemini.py` |
| **Hugging Face** | Llama, Mistral, CodeLlama | `2.Models-ChatModel/4_chat_hf.py` |
| **Groq** | Fast inference models | `2.Models-ChatModel/6_groq-api.py` |

### Featured Models

- `openai/gpt-oss-20b`
- `deepseek-ai/DeepSeek-R1`
- `meta-llama/Llama-3.1-8B-Instruct`
- `meta-llama/Llama-3.2-3B-Instruct`
- `moonshotai/Kimi-K2-Instruct-0905`

### Learning Path

1. **Start with Models**: Explore `1.Models-LLMS/` and `2.Models-ChatModel/`
2. **Learn Prompting**: Practice with examples in `4.Prompts/`
3. **Build Chains**: Understand workflows in `7.Chain/`
4. **Implement RAG**: Follow the RAG modules sequentially
5. **Create Agents**: Explore `AI_AGENT/` for autonomous systems
6. **Build Projects**: Apply knowledge with real-world examples

### Key Dependencies

```
# Core LangChain
langchain
langchain-core
langchain-community

# Model Integrations
langchain-openai
langchain-anthropic
langchain-google-genai
langchain-huggingface
langchain_groq

# Utilities
python-dotenv
streamlit
pypdf
scikit-learn
numpy
```

### Environment Setup

The repository includes comprehensive environment management:

- **Virtual Environment**: Isolated Python environment for dependencies
- **API Key Management**: Secure handling of multiple provider keys
- **Cross-Platform Support**: Works on Windows, macOS, and Linux
- **Development Tools**: Pre-configured for Jupyter notebooks and Python scripts

### Documentation

Each folder contains specific implementations with detailed comments and examples. The codebase follows best practices for:

- **Error Handling**: Robust exception management
- **Code Organization**: Modular structure for easy understanding
- **Documentation**: Inline comments and docstrings
- **Testing**: Example usage and validation scripts

### Contributing

1. Fork the repository
2. Create a feature branch (`git checkout -b feature/amazing-feature`)
3. Commit your changes (`git commit -m 'Add some amazing feature'`)
4. Push to the branch (`git push origin feature/amazing-feature`)
5. Open a Pull Request

## License

This project is open source and available under the MIT License.

## Useful Links

- LangChain Documentation
- OpenAI API Documentation
- Hugging Face Hub
- Anthropic Claude API

## Support

If you have questions or run into issues:

1. Check the existing code examples
2. Review the documentation in each folder
3. Open an issue on GitHub
4. Join the LangChain community discussions

**Happy Learning!**

*This repository is actively maintained and updated with the latest LangChain features and best practices.*