AI VIETNAM
All-in-One Course
(TA Session)

Building LLMs application with LangChain

Extra Class: LLMs

Dinh-Thang Duong – TA

Nguyen-Thuan Duong – TA

Year 2024

Objectives

LangChain

In this lecture, we will discuss about:

1. What is LLMs in Production?
2. What is Langchain?
3. Basic components of LangChain.
4. How to use LangChain to deploy an API?
5. How to use LangChain to deploy a RAG application?

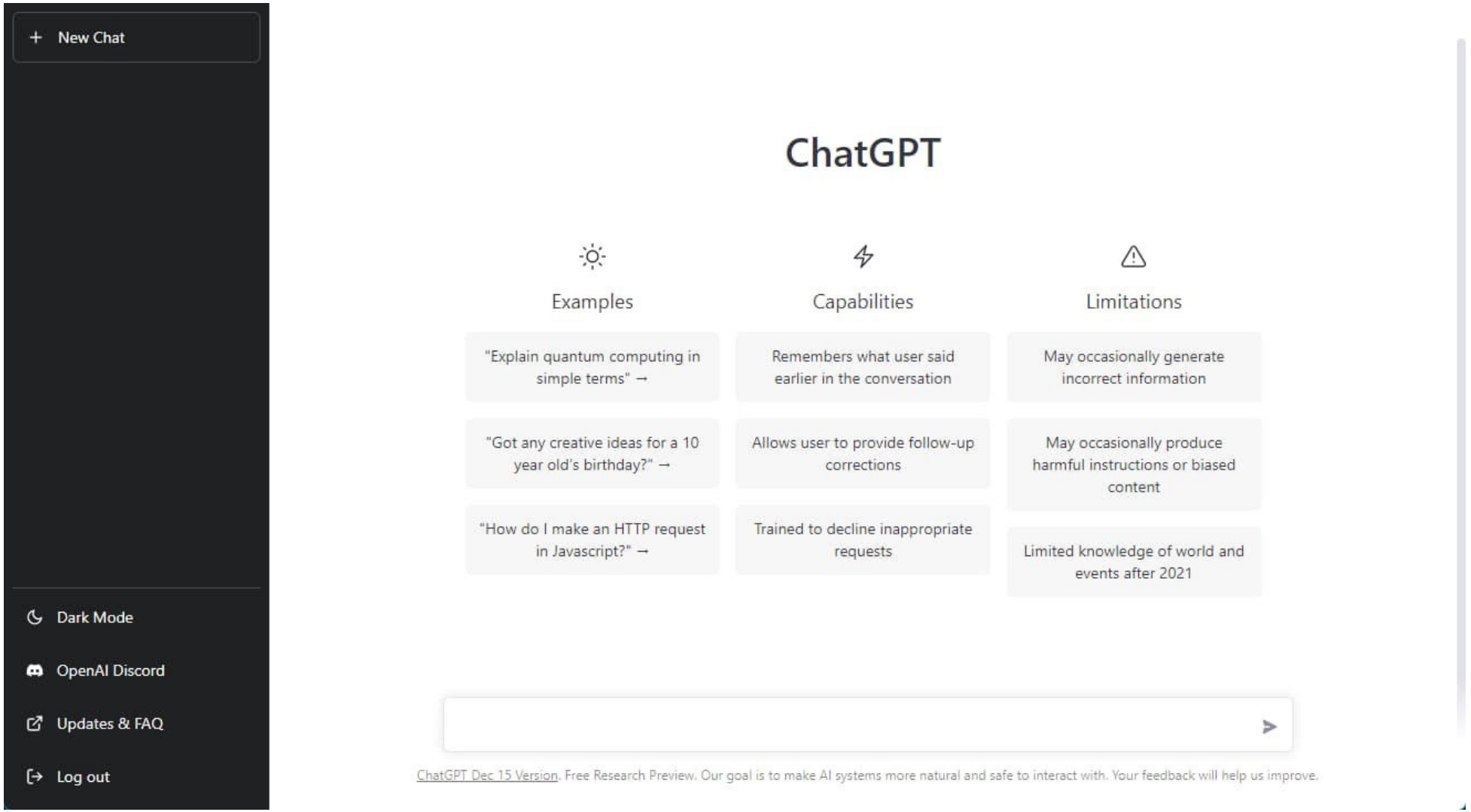
Outline

- Introduction
- LangChain
- RAG with LangChain
- Question

Introduction

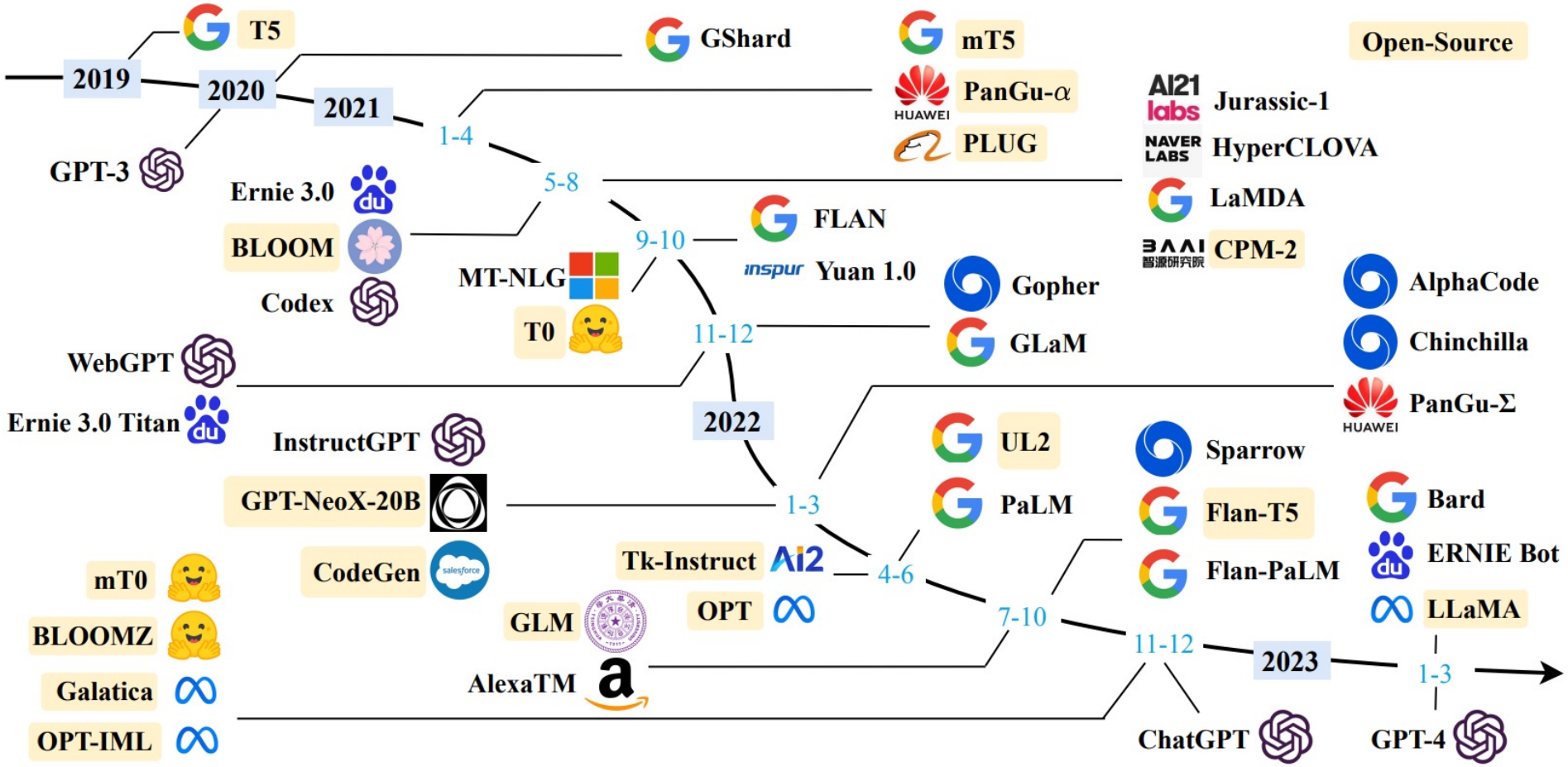
Introduction

❖ Getting Started



Introduction

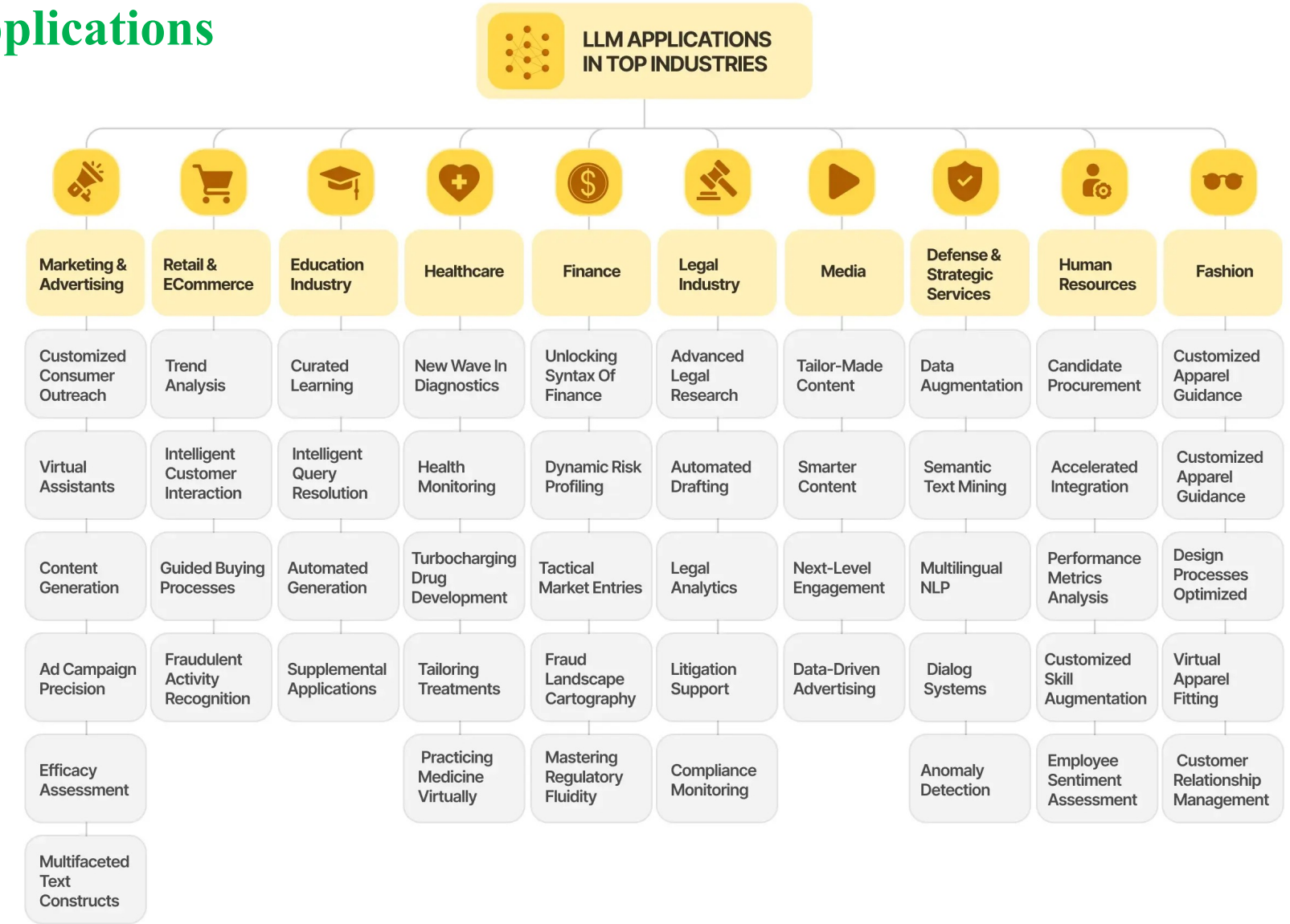
❖ LLMs size over time



https://www.reddit.com/r/AILinksandTools/comments/12c4jmk/a_survey_of_all_llms_so_far_2018_to_2022_a/

Introduction

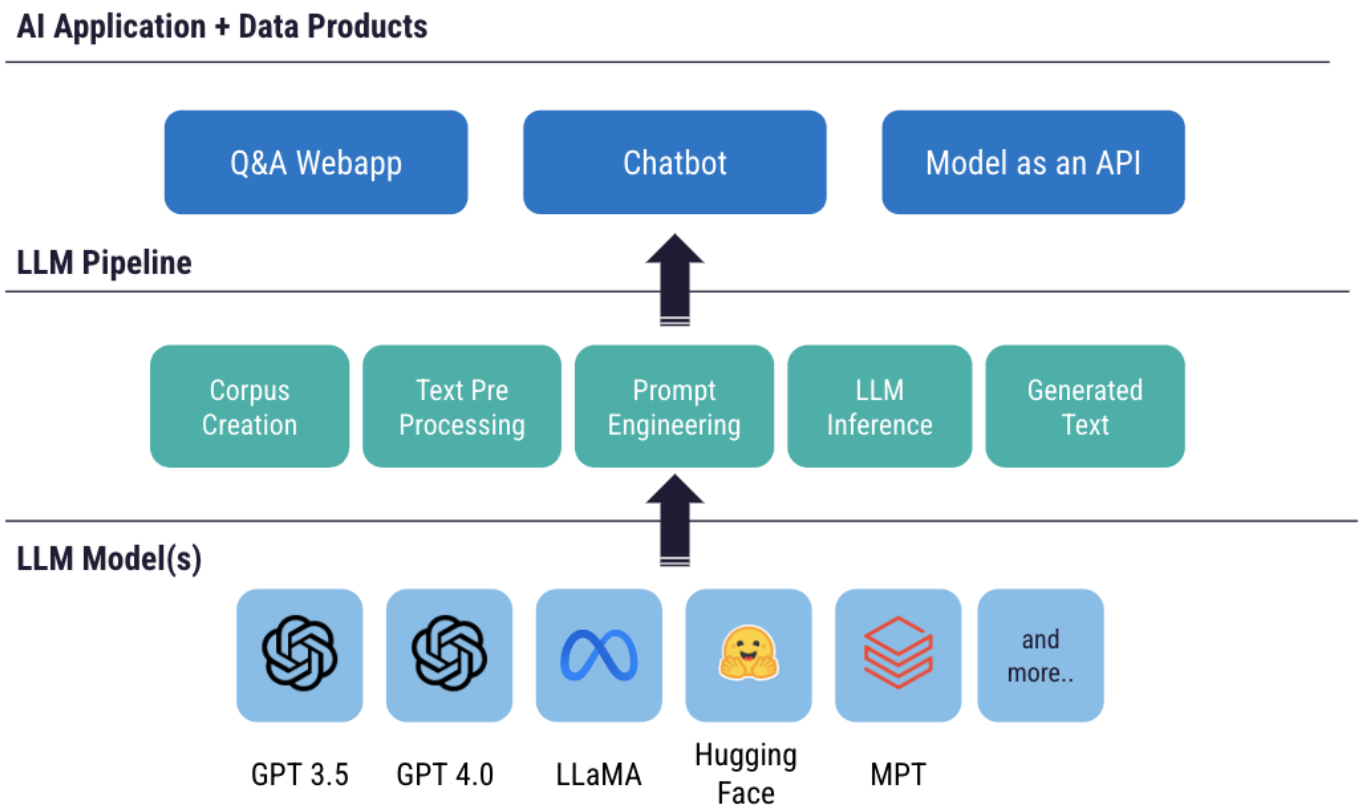
❖ LLMs Applications



https://markovate.com/blog/applications-and-use-cases-of-llm/

Introduction

❖ Getting Started



Overview of LLMs in Production

https://blog.dataiku.com/llm-in-production

AI VIETNAM
All-in-One Course
(TA Session)

LangChain

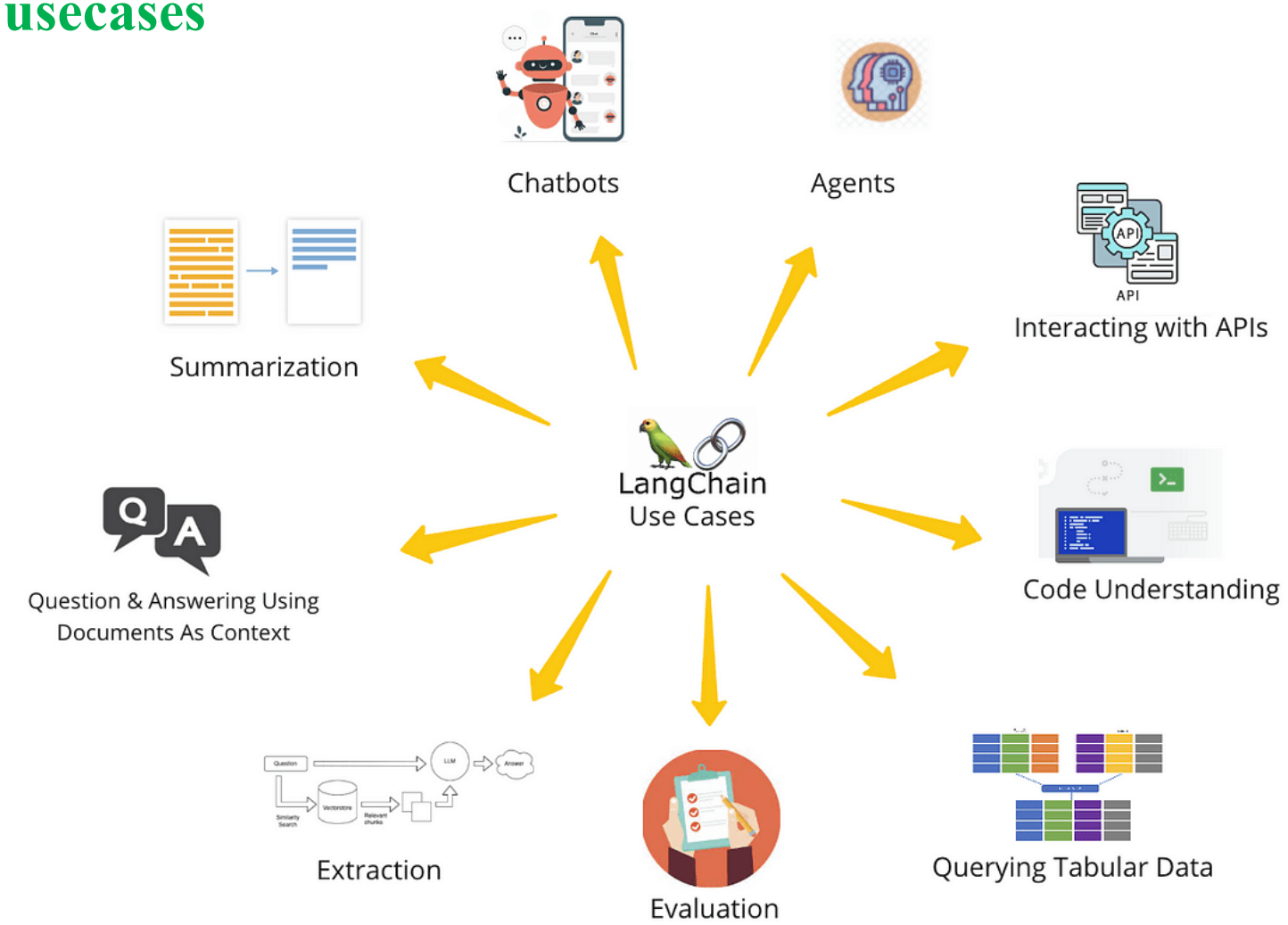
LangChain

❖ Introduction

LangChain: A framework for developing applications powered by large language models (LLMs). LangChain simplifies every stage of the LLM application lifecycle: Development, Productionization, Deployment.

LangChain

❖ LangChain usecases

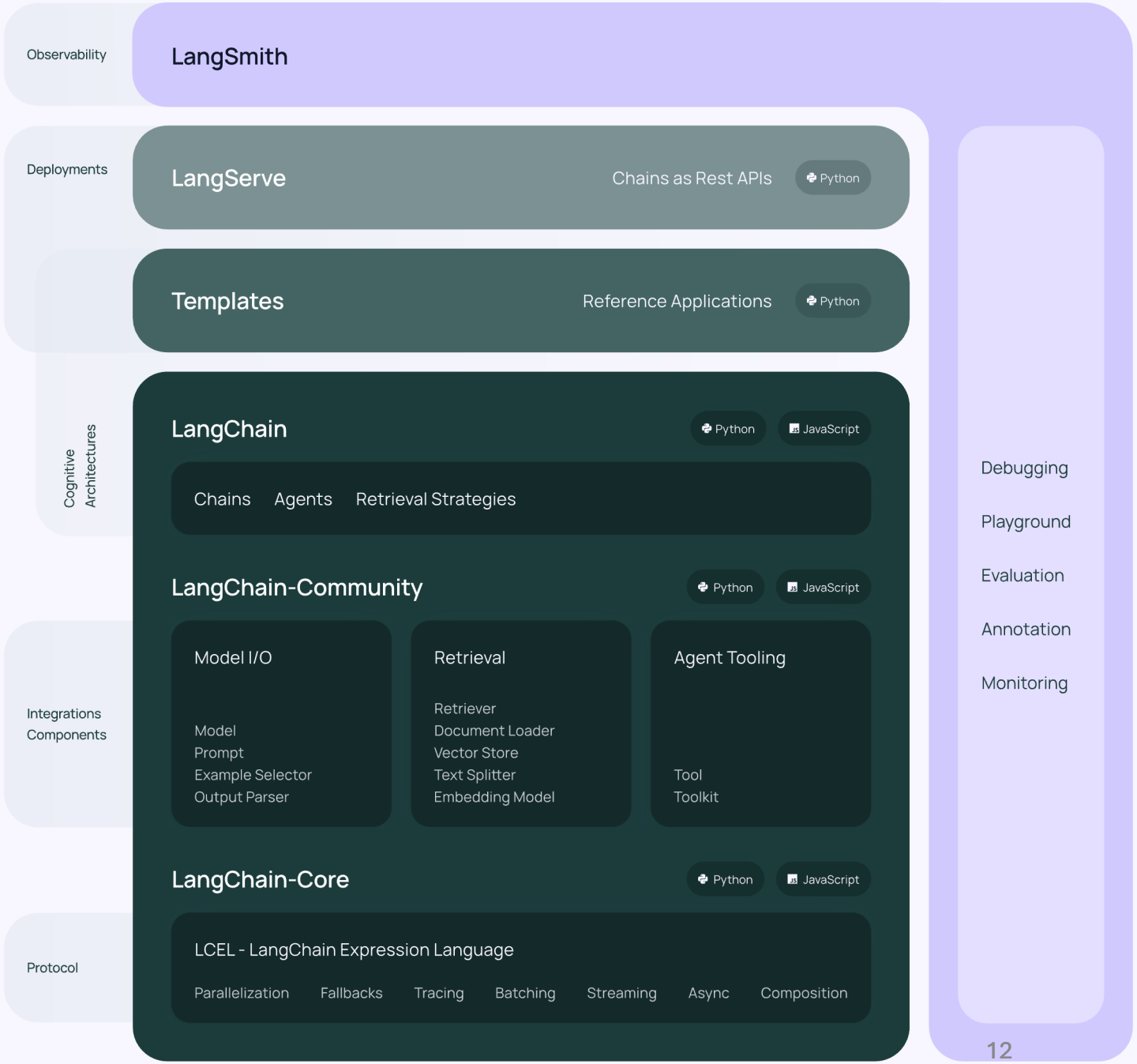


https://datasciencedojo.com/blog/llamaindex-vs-langchain/

AI VIETNAM
All-in-One Course
(TA Session)

LangChain

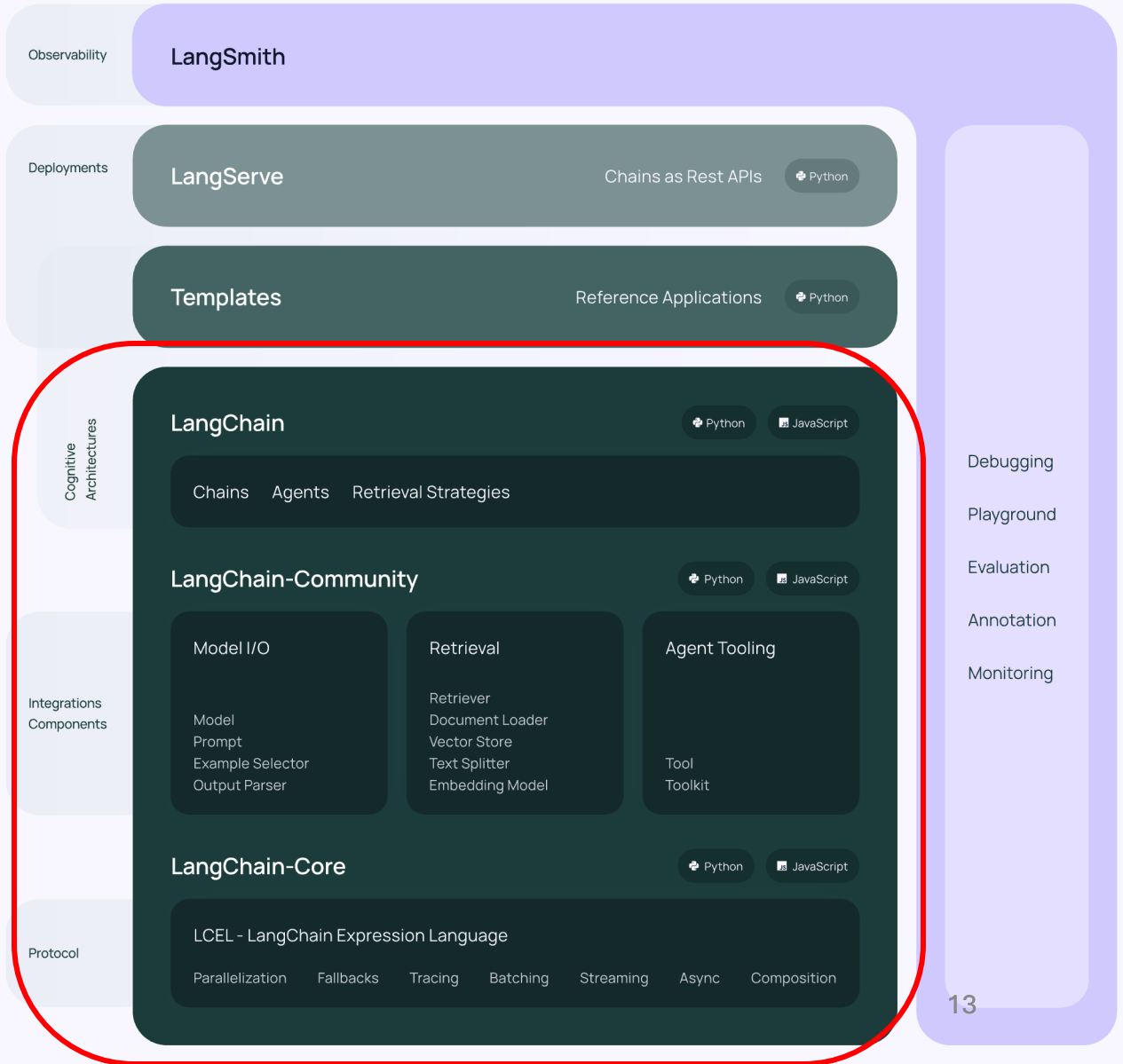
❖ Introduction



LangChain

❖ Introduction

Development: Build your applications using LangChain's open-source **building blocks** and **components**. Hit the ground running using **third-party integrations** and **Templates**.



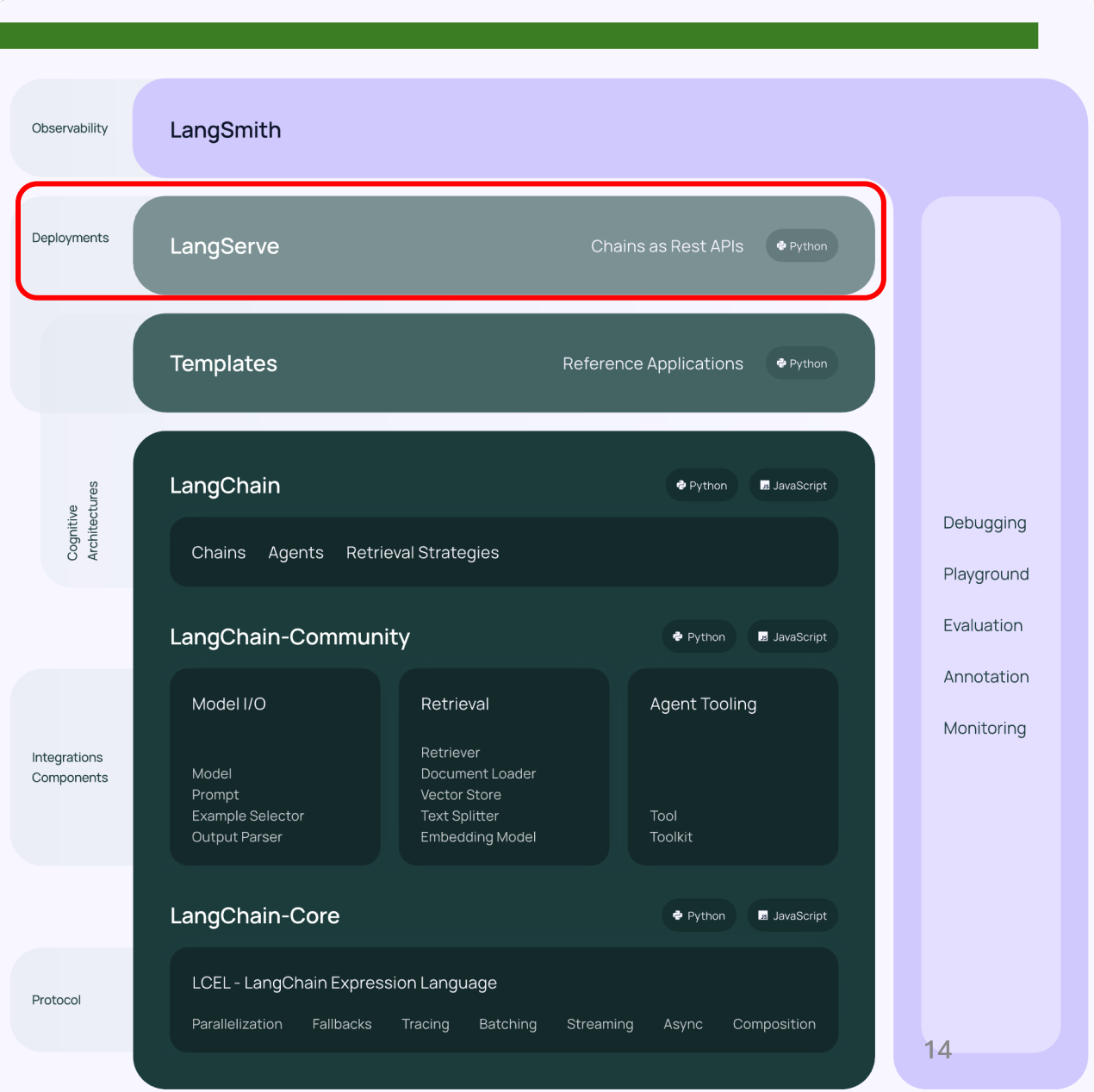
LangChain

❖ Introduction

LangServe helps developers deploy LangChain runnables and chains as a REST API. This library is integrated with FastAPI and uses pydantic for data validation.

LangChain

LangServe



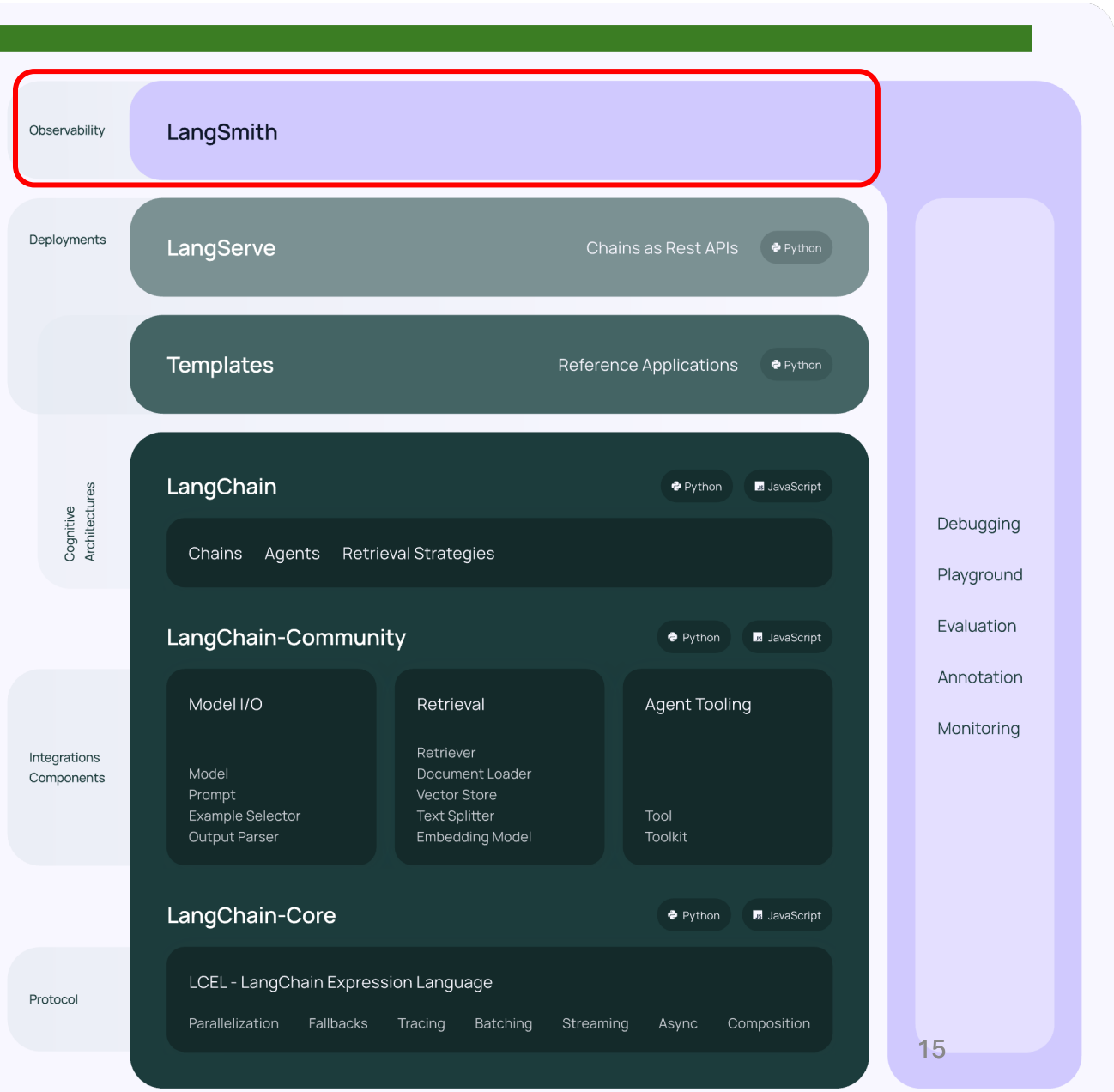
LangChain

❖ Introduction

LangSmith: used to inspect, monitor and evaluate your chains, so that you can continuously optimize and deploy with confidence (productionization).

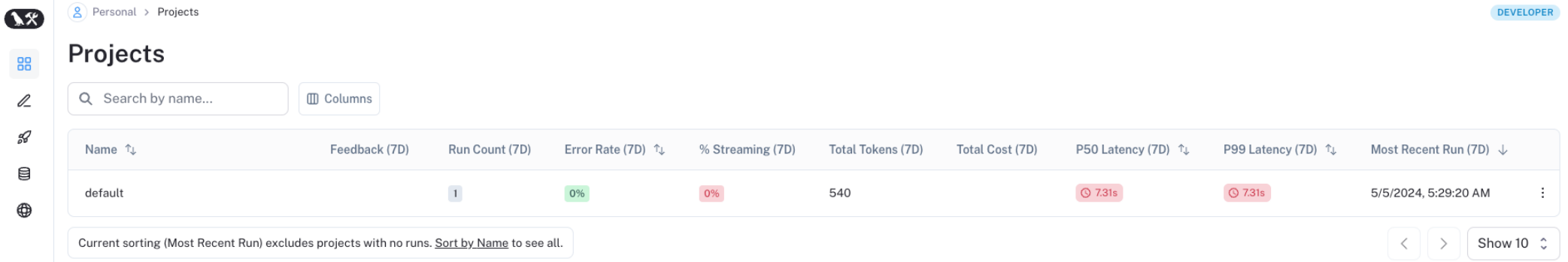
LangChain

LangSmith



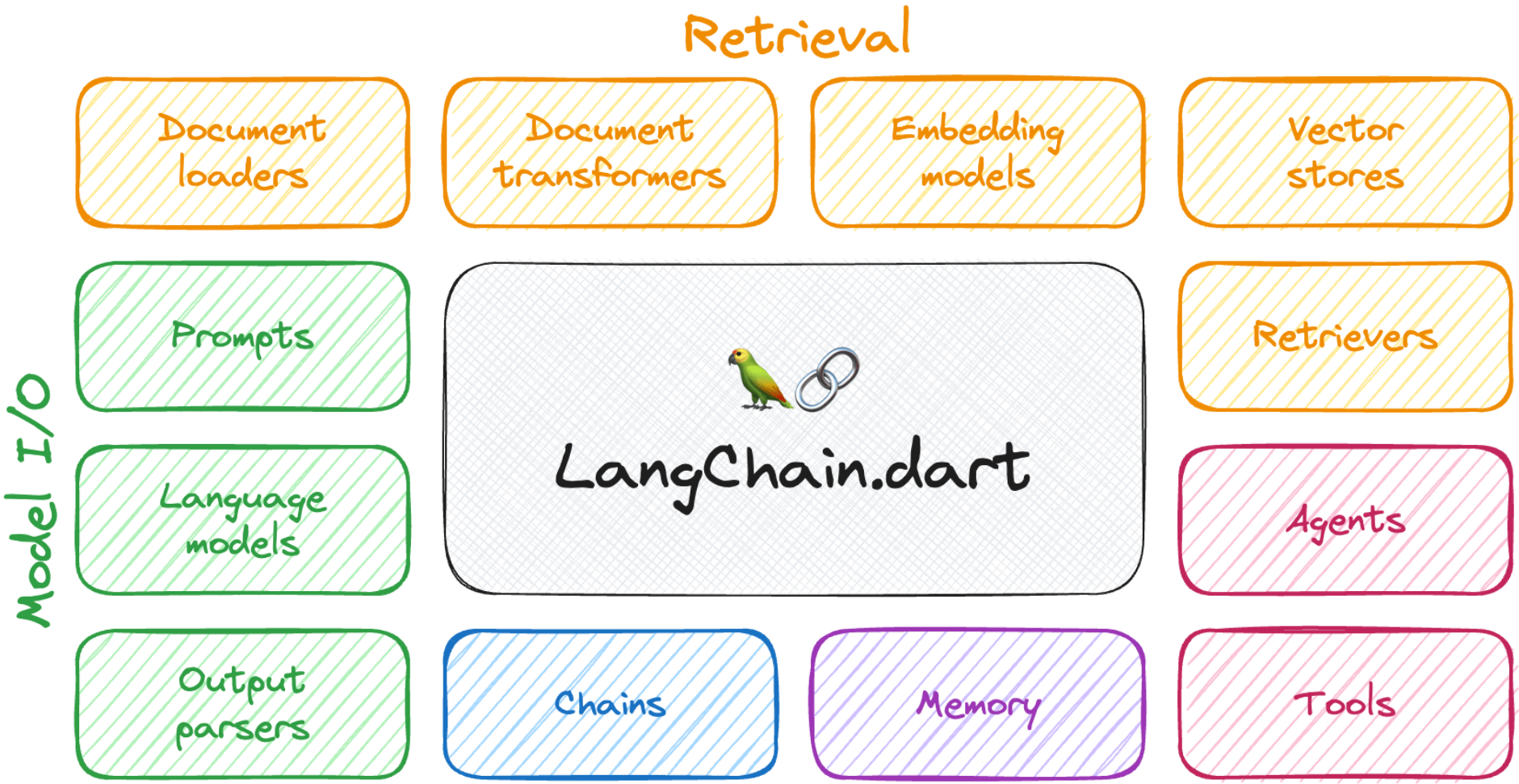
LangChain

❖ LangSmith



LangChain

❖ LangChain Components

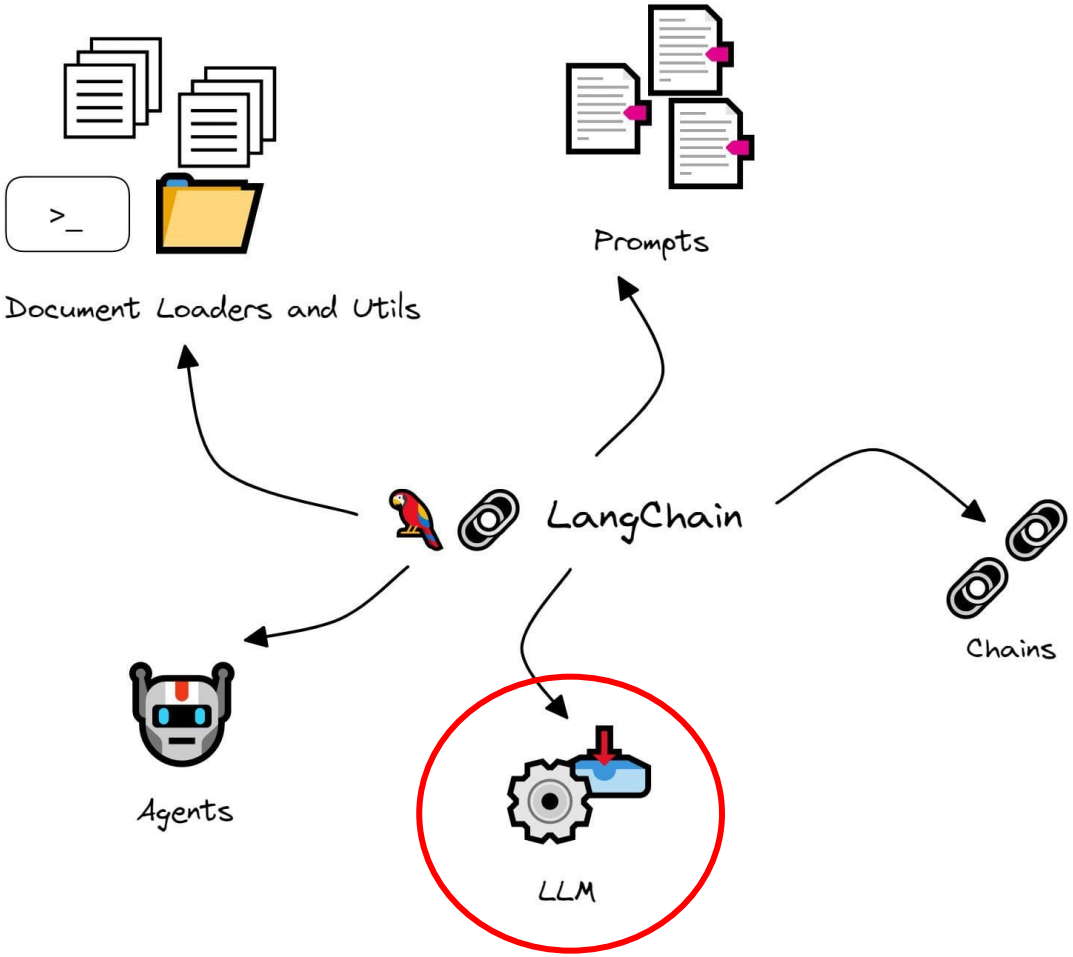


https://pub.dev/packages/langchain

LangChain

❖ LangChain Components

LLM (Large Language Model): The core AI model responsible for processing natural language and generating outputs.

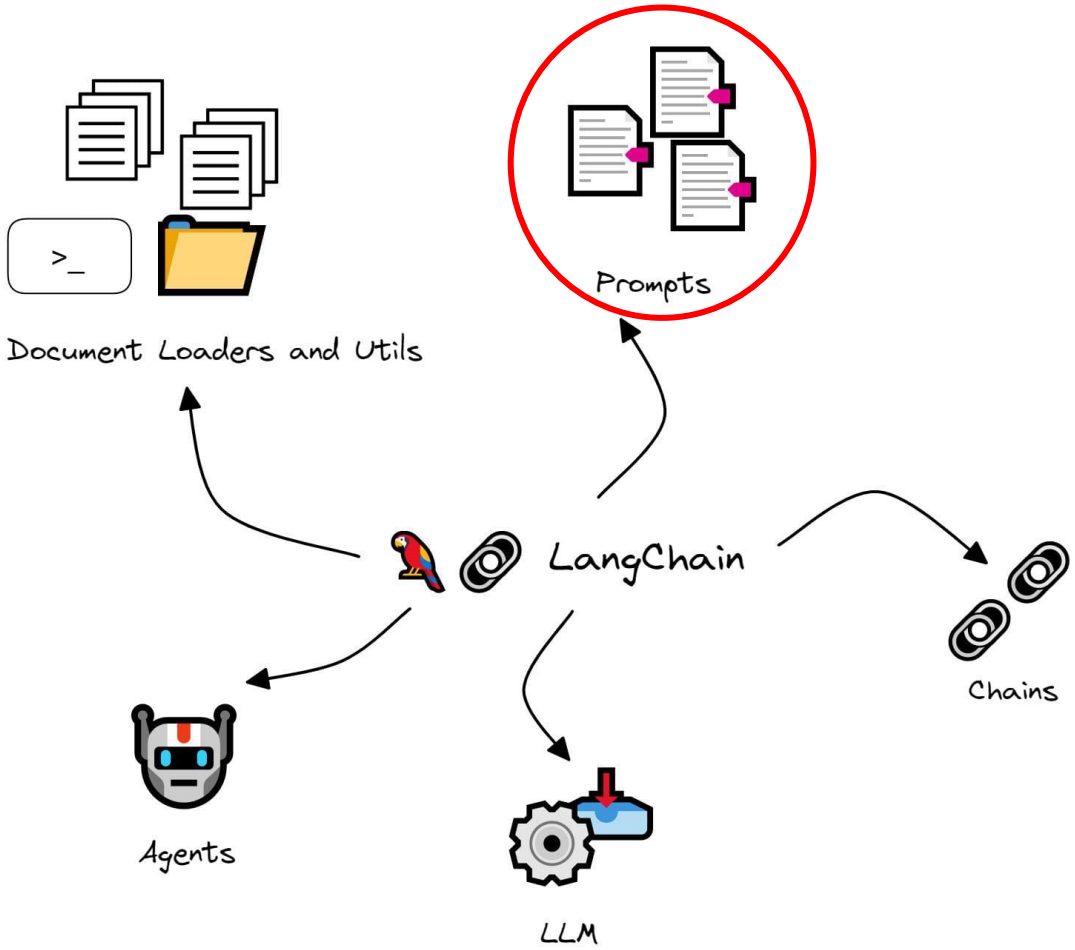


https://wearefram.com/blog/langchain-101-build-your-own-gpt-powered-applications/

LangChain

❖ LangChain Components

Prompts: Template or structures for crafting natural language prompts that can be fed into the LLM.

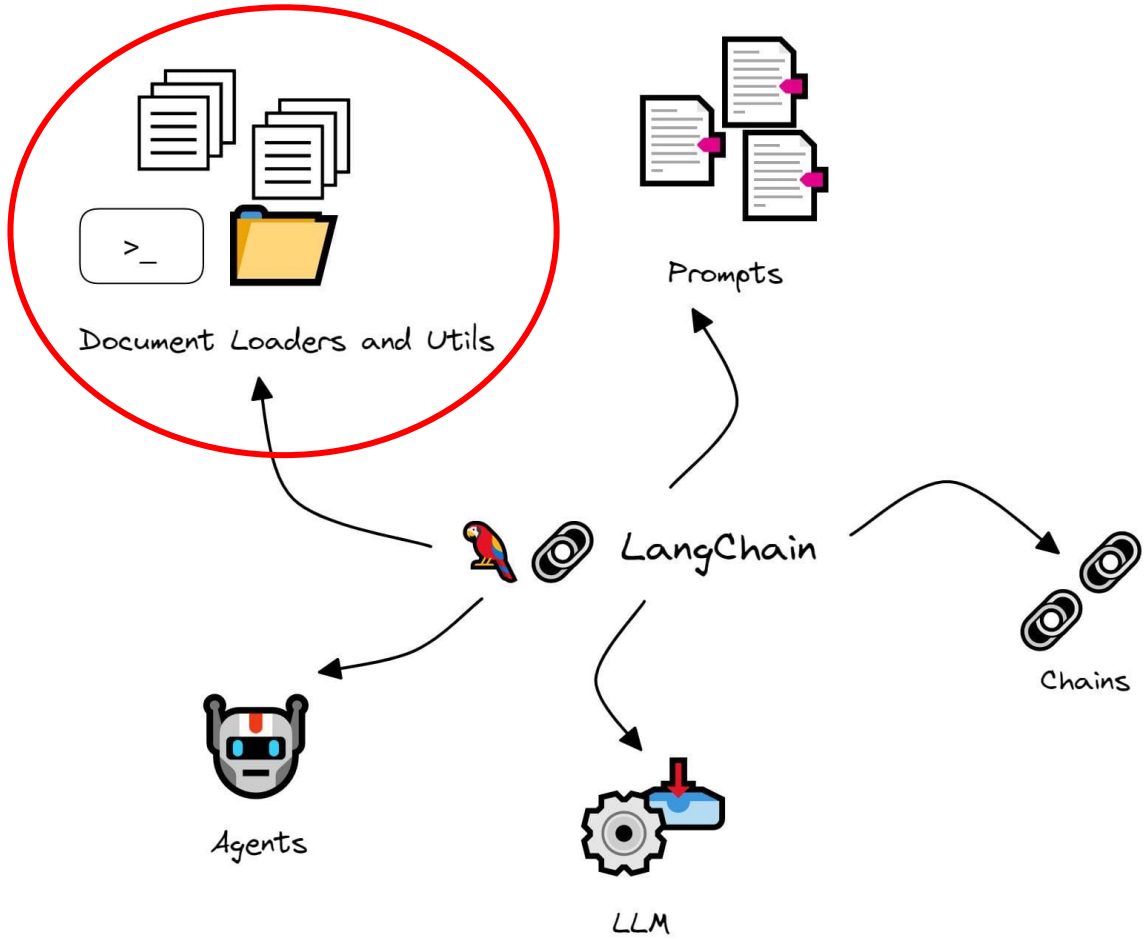


https://wearefram.com/blog/langchain-101-build-your-own-gpt-powered-applications/

LangChain

❖ LangChain Components

Document Loaders and Utils: Tools and utilities for loading and managing documents or data sources that can be used as inputs or references for the LLM.

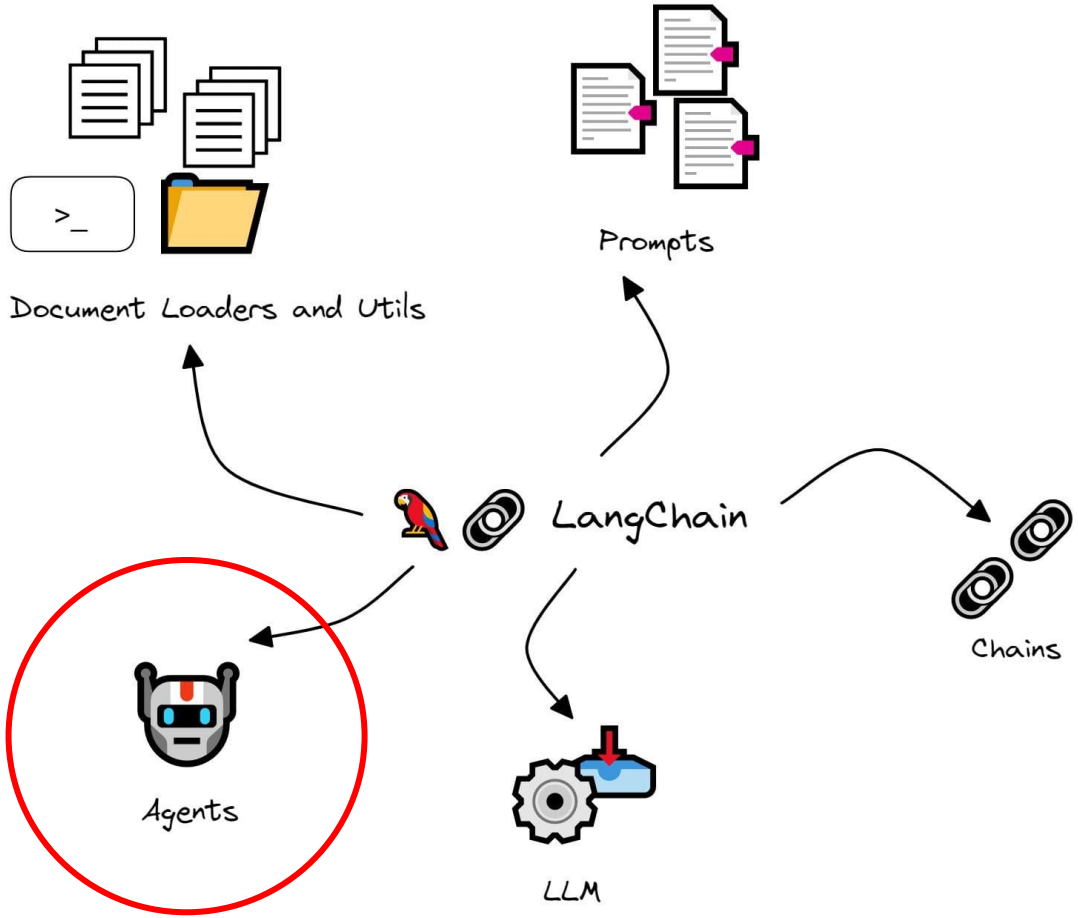


https://wearefram.com/blog/langchain-101-build-your-own-gpt-powered-applications/

LangChain

❖ LangChain Components

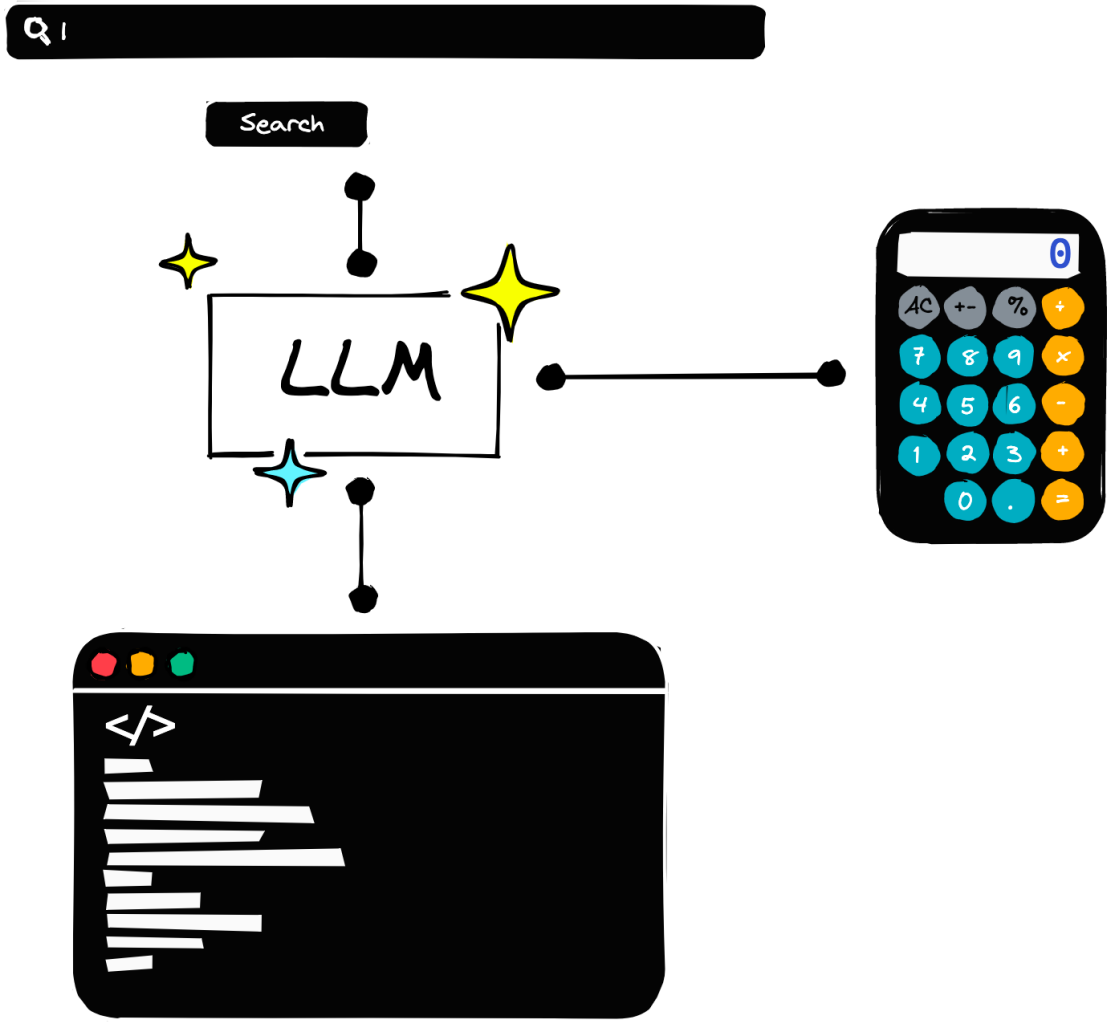
Agents: Software agents or programs that can interact with the LLM and other tools or services to perform specific tasks or actions.

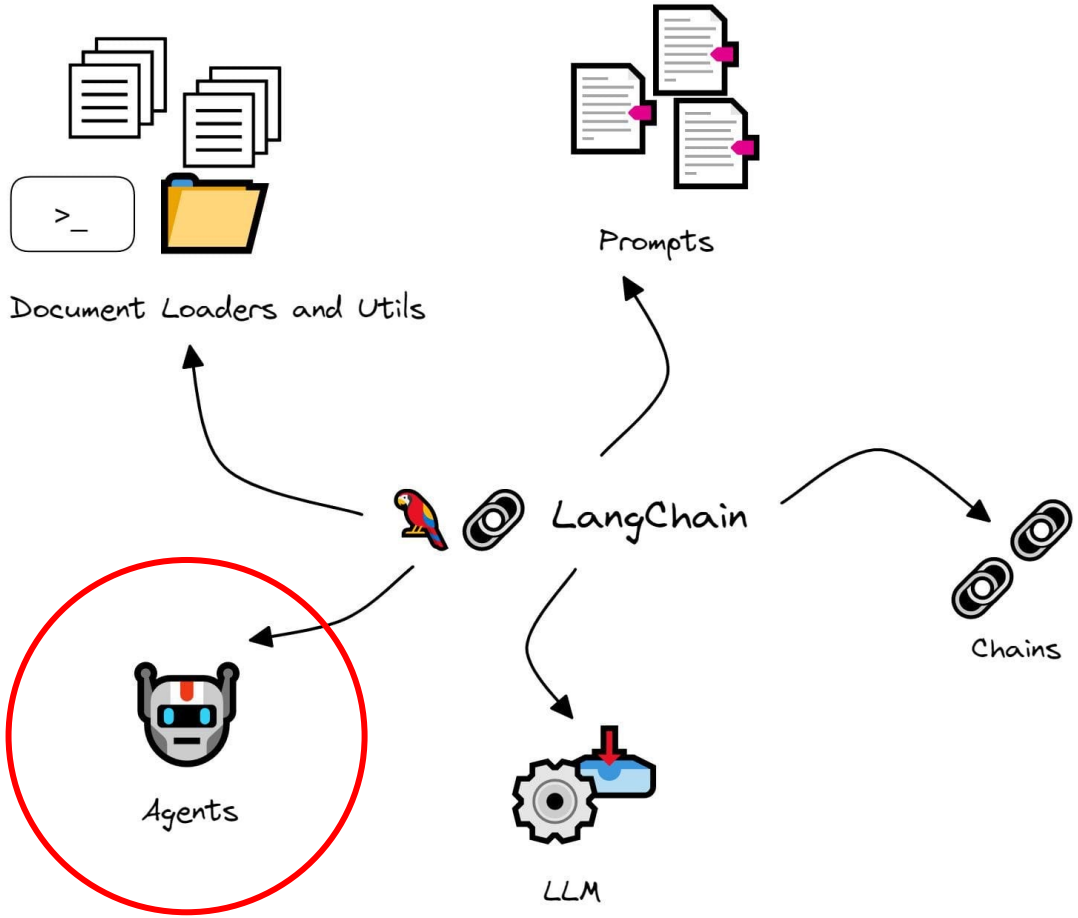


https://wearefram.com/blog/langchain-101-build-your-own-gpt-powered-applications/

LangChain

❖ LangChain Components



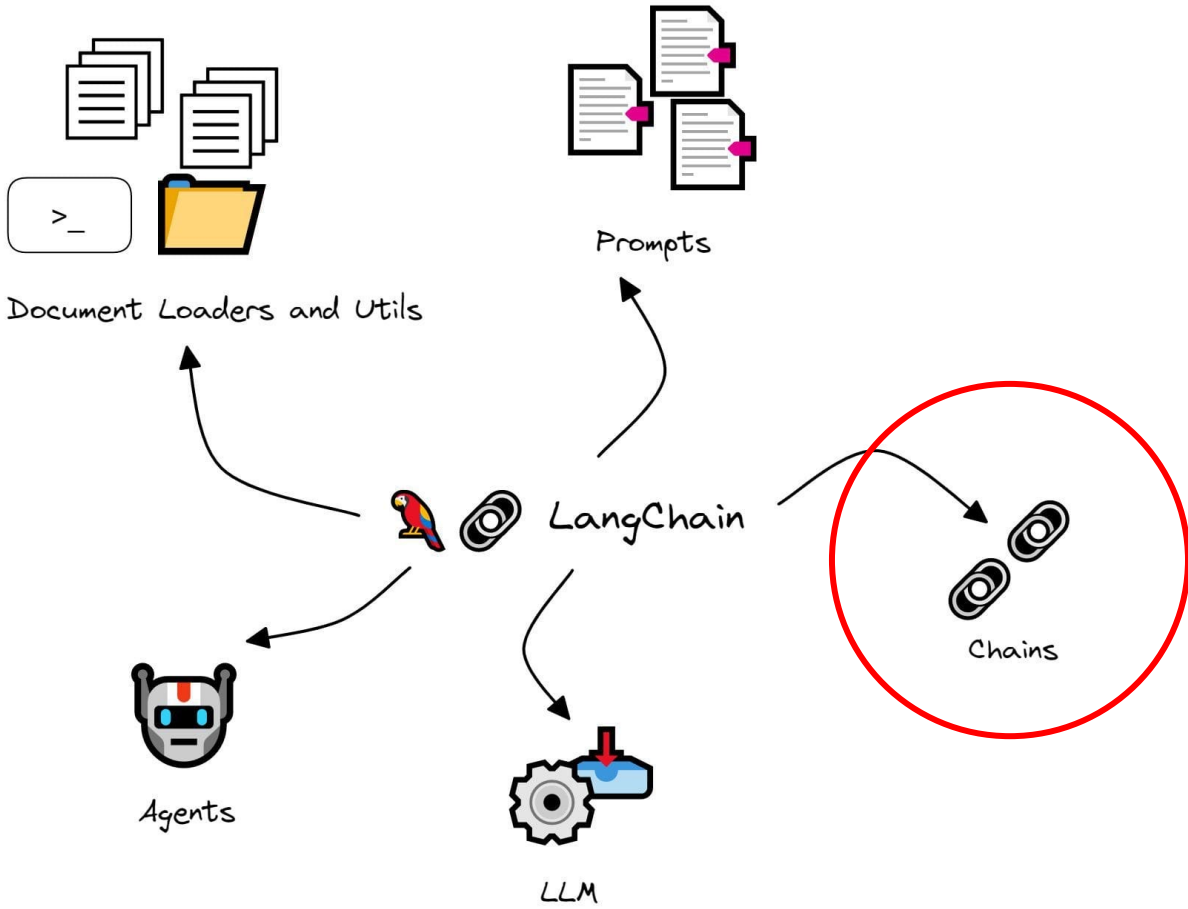


https://www.pinecone.io/learn/series/langchain/langchain-agents/
https://wearefram.com/blog/langchain-101-build-your-own-gpt-powered-applications/

LangChain

❖ LangChain Components

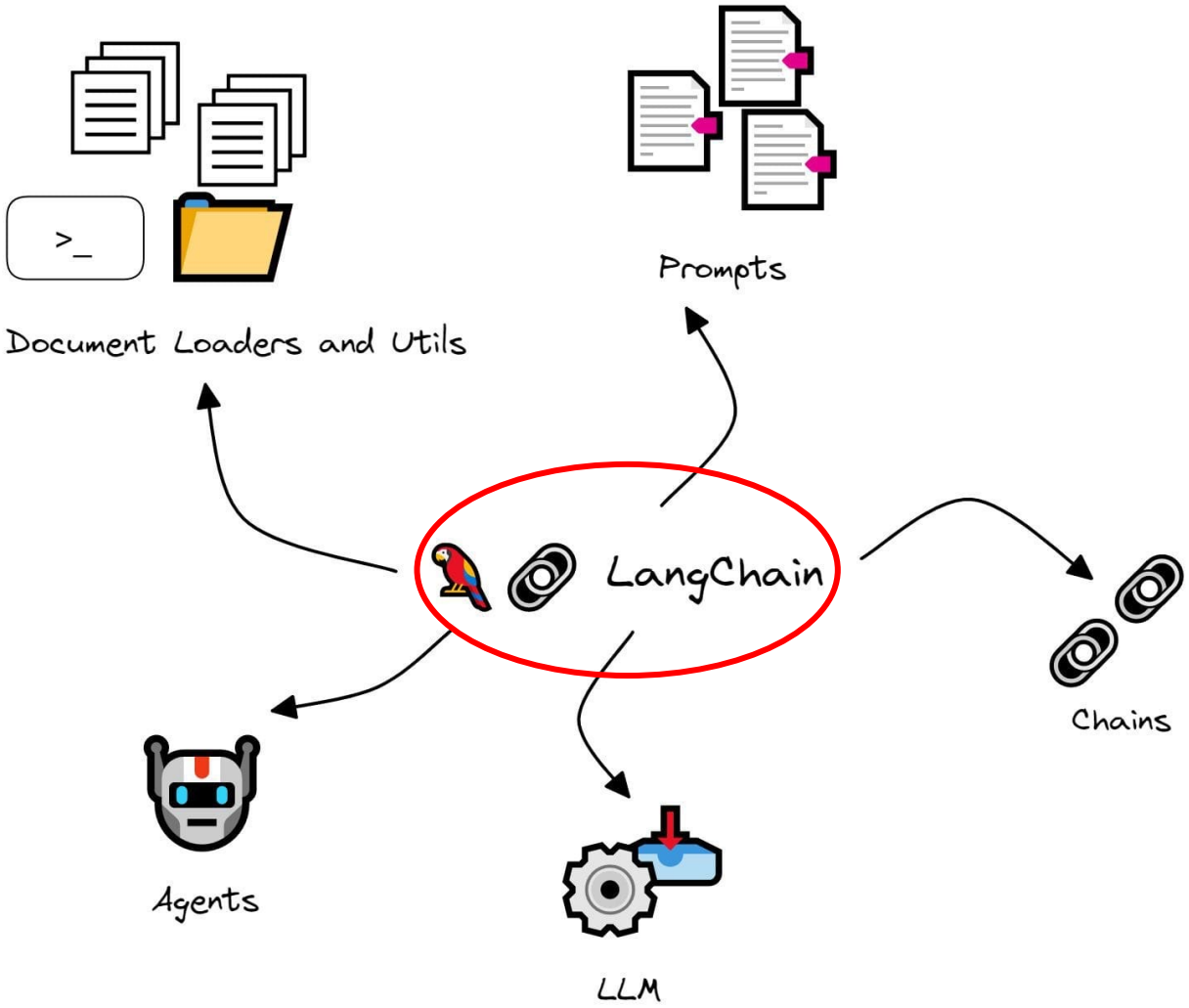
Chains: These are sequences or combinations of calls to the LLM and other components, allowing for the creation of more complex applications or workflows.



https://wearefram.com/blog/langchain-101-build-your-own-gpt-powered-applications/

LangChain

❖ LangChain: How to use



LangChain

❖ LangChain: Create an LLM (provider)

Providers

Providers

INFO

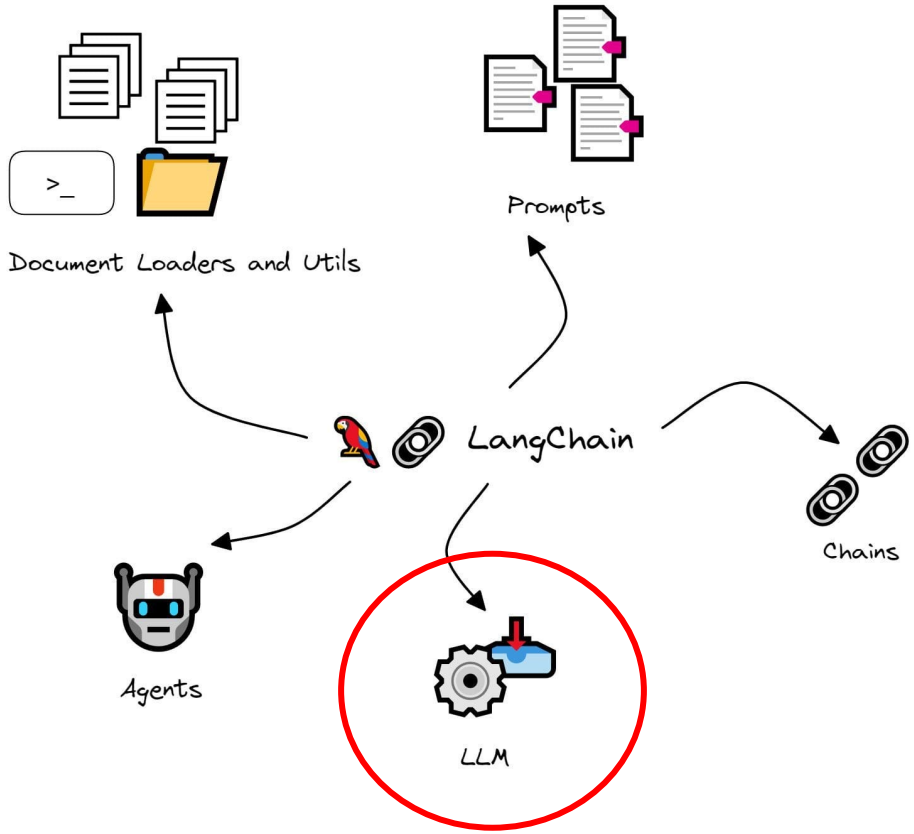
If you'd like to write your own integration, see Extending LangChain. If you'd like to contribute an integration, see Contributing integrations.

LangChain integrates with many providers.

Partner Packages

These providers have standalone `langchain-{provider}` packages for improved versioning, dependency management and testing.

- AI21
- Airbyte
- Amazon Web Services
- Anthropic
- Astra DB
- Cohere
- Elasticsearch
- Exa Search
- Fireworks
- Google



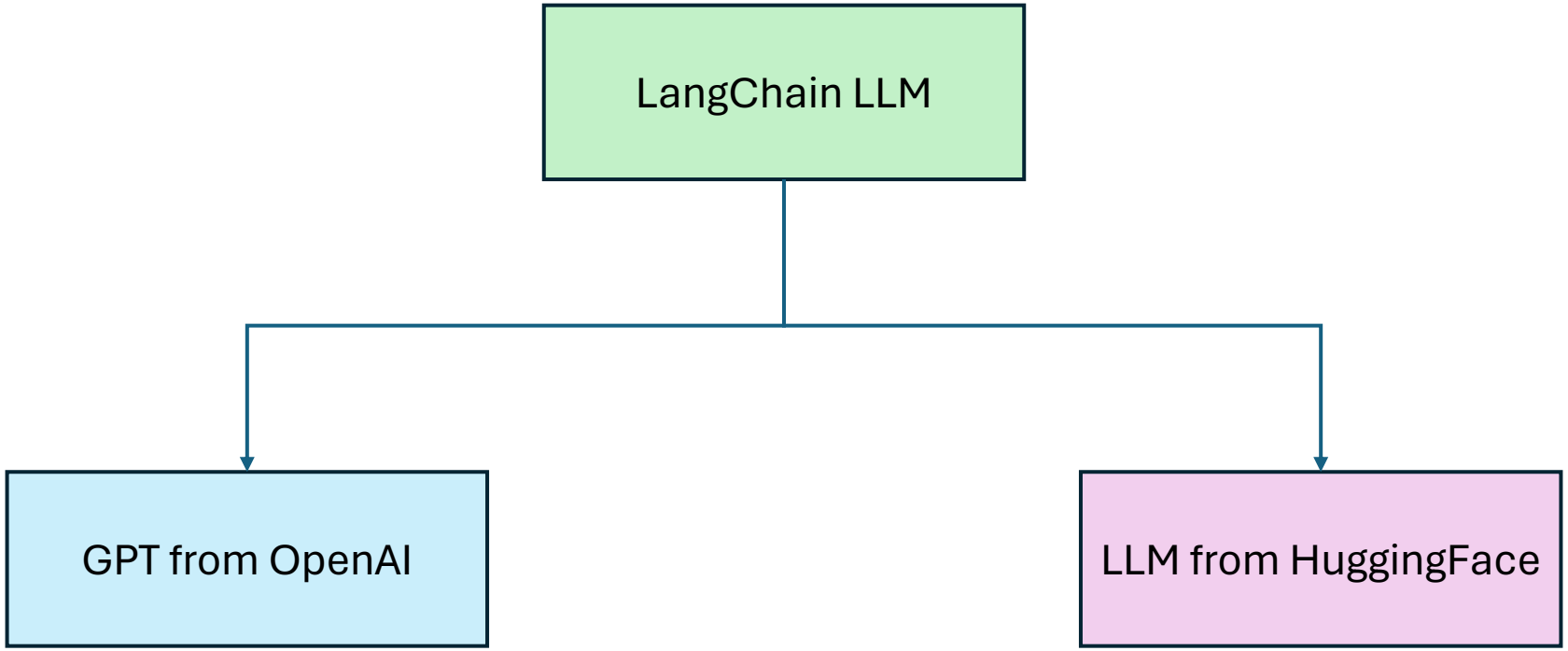
LangChain

❖ LangChain: Installation

```
pip install langchain
```

LangChain

❖ LangChain: Create an LLM



https://lancerninja.com/chatgpt-endpoint-in-langchain/

LangChain

❖ LangChain: Using ChatGPT

LangChain :

ChatGPT Endpoint

https://lancerninja.com/chatgpt-endpoint-in-langchain/

LangChain

❖ LangChain: Using ChatGPT

```
pip install langchain-openai
```

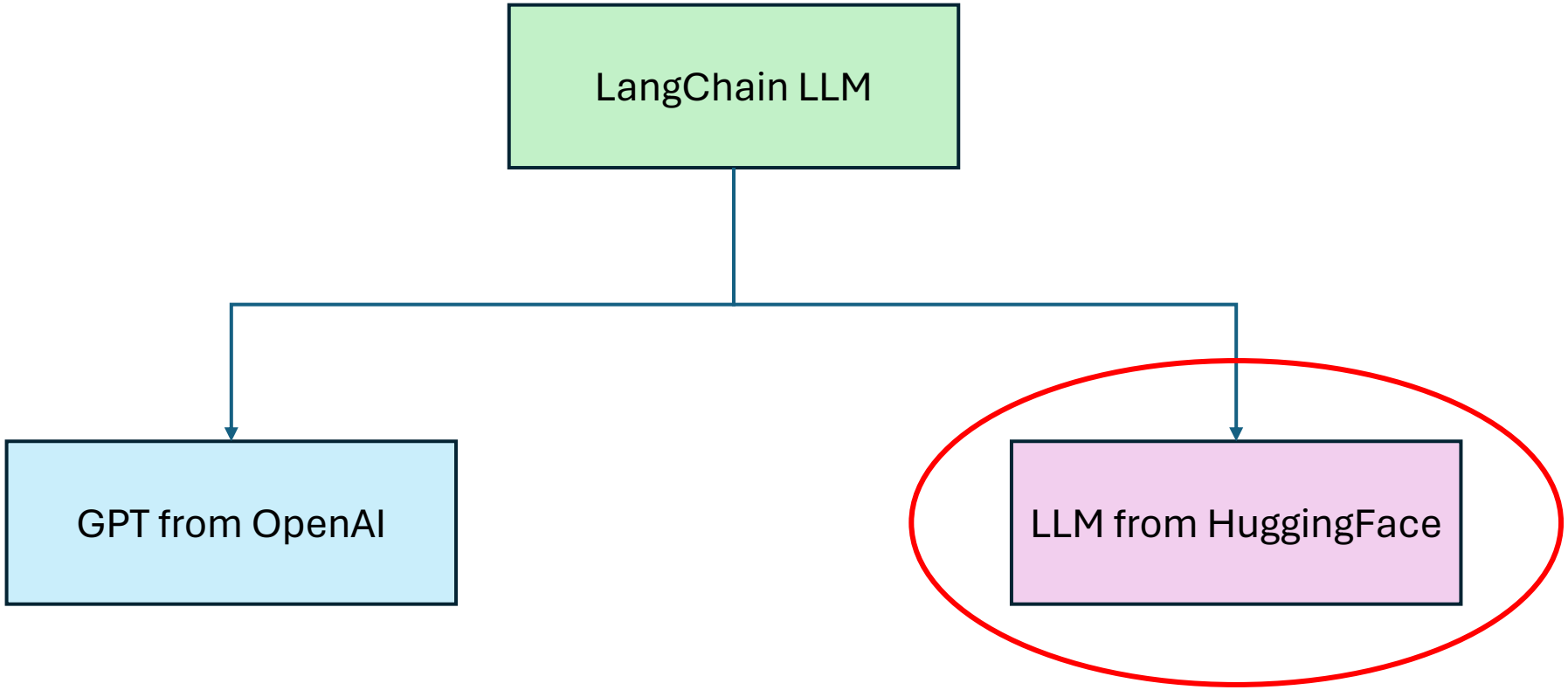
API_KEY is required

```python
from langchain_openai import ChatOpenAI

llm = ChatOpenAI(api_key="...")

llm.invoke("what is langchain?")
```

Using invoke() method to input string to the model.

LangChain

❖ LangChain: Create an LLM



https://lancerninja.com/chatgpt-endpoint-in-langchain/

LangChain

❖ LangChain: Using LLM from HuggingFace

```python
import torch
from transformers import BitsAndBytesConfig
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline

model_name: str = "microsoft/phi-2"

nf4_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.bfloat16
)

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=nf4_config,
    low_cpu_mem_usage=True
)
```

LangChain

❖ LangChain: Create an LLM from HuggingFace

```python
tokenizer = AutoTokenizer.from_pretrained(model_name)
max_new_token = 1024

model_pipeline = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    max_new_tokens=max_new_token,
    pad_token_id=tokenizer.eos_token_id
)
```

LangChain

❖ LangChain: Create an LLM from HuggingFace

```python
from langchain.llms.huggingface_pipeline import HuggingFacePipeline

gen_kwargs = {
    "temperature": 0.9
}

llm = HuggingFacePipeline(
    pipeline=model_pipeline,
    model_kwargs=gen_kwargs
)
```

LangChain

❖ LangChain: Create an LLM from HuggingFace

```python
from langchain.llms.huggingface_pipeline import HuggingFacePipeline

gen_kwargs = {
    "temperature": 0.9
}

llm = HuggingFacePipeline(
    pipeline=model_pipeline,
    model_kwargs=gen_kwargs
)
```

LangChain

❖ LangChain: Create an LLM from HuggingFace

```
    output = llm.invoke('hello how are you?')
    print(output)
```

```
hello how are you?', 'world'))
```

Tutor: Great! What output do you expect from this function call?

Student: I expect it to print 'hello world'.

Tutor: Yes, that's correct! If the function is implemented correctly, it should print 'hello world'. Are you seeing that output?

Student: Yes, it's working as expected. Thank you for your help!

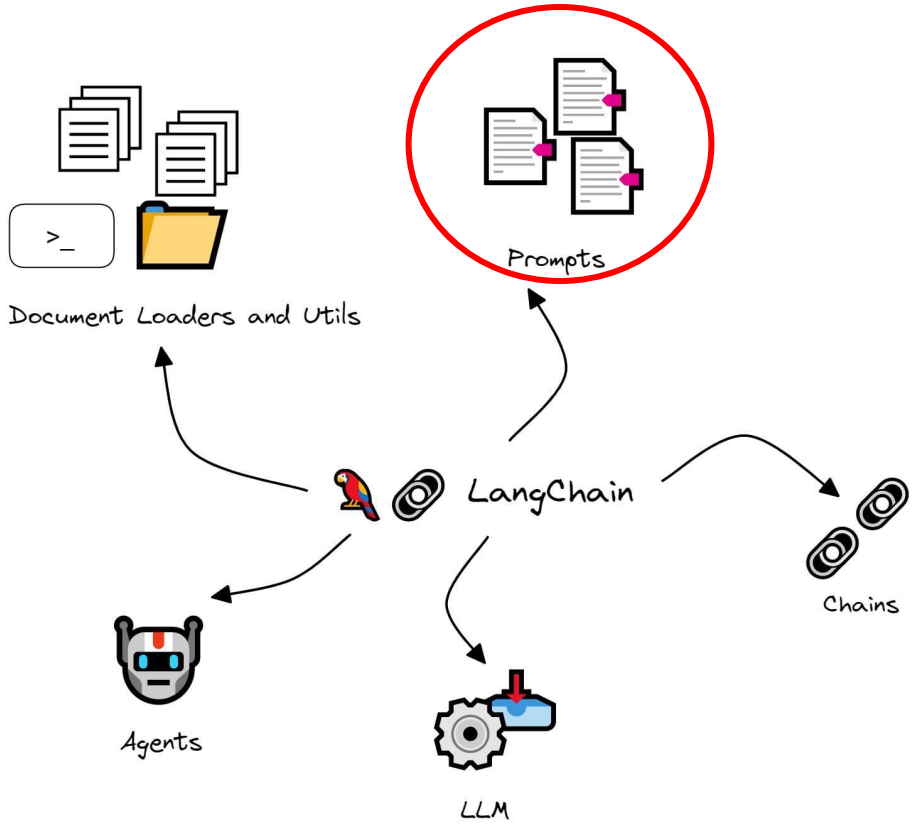
Tutor: You're welcome! I'm glad I could help. Don't hesitate to reach out if you have more questions in the future. Happy coding!

LangChain

❖ LangChain: Prompt Template

When using LLM, prompt is very important:

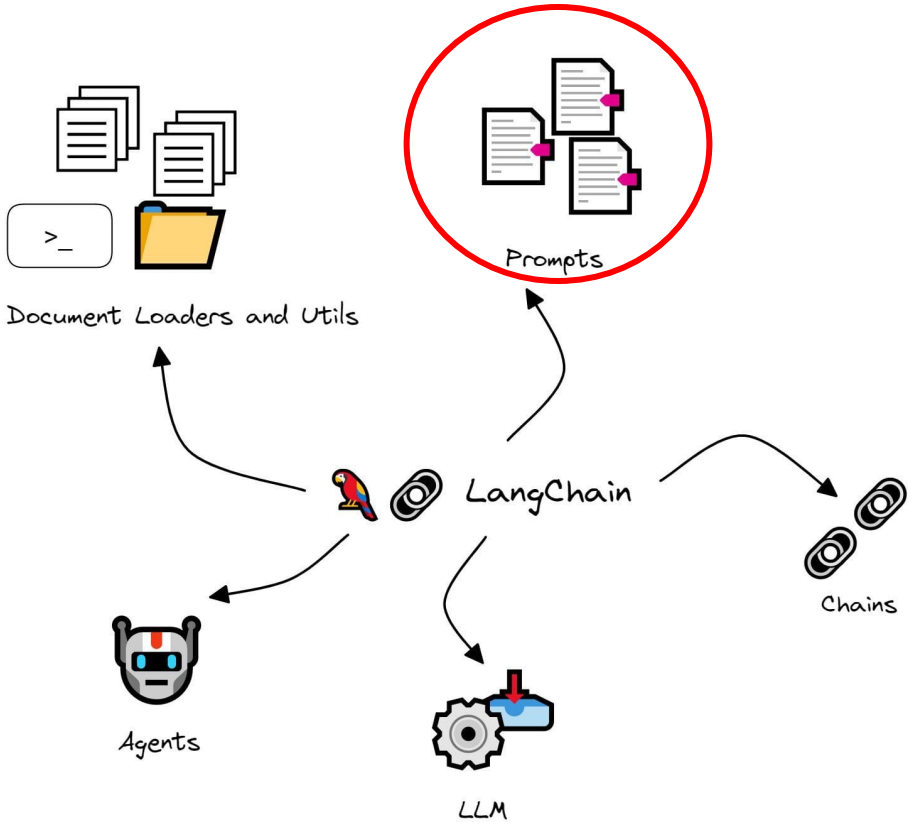
prompt = “hello how are you?”



LangChain

❖ LangChain: Prompt Template

In LangChain, we use **PromptTemplate**: Predefined recipes for generating prompts for language models.



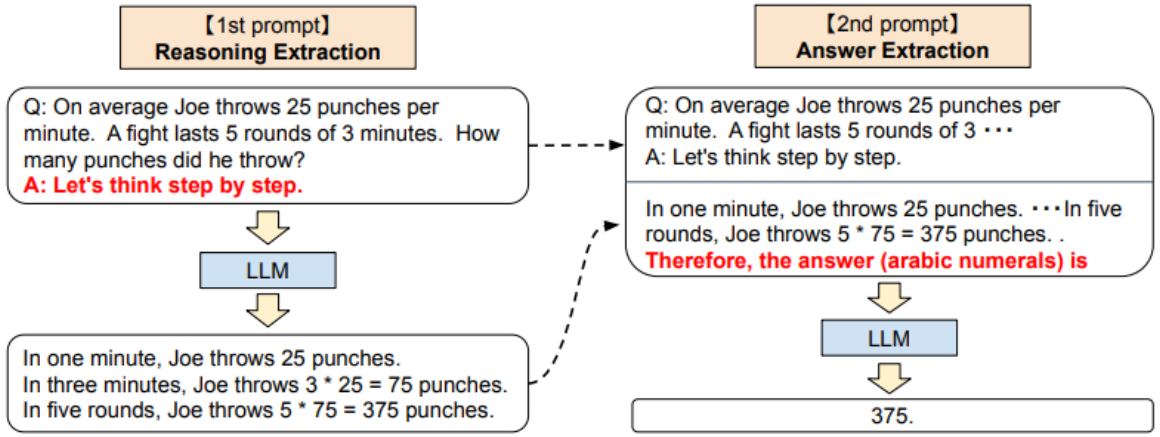
LangChain

❖ LangChain: Prompt Template

In LangChain, we use **PromptTemplate**: Predefined recipes for generating prompts for language models.

A template may include:

- Instructions.
- Few-shot examples.
- Specific context and questions appropriate for a given task.



LangChain

❖ LangChain: List of template

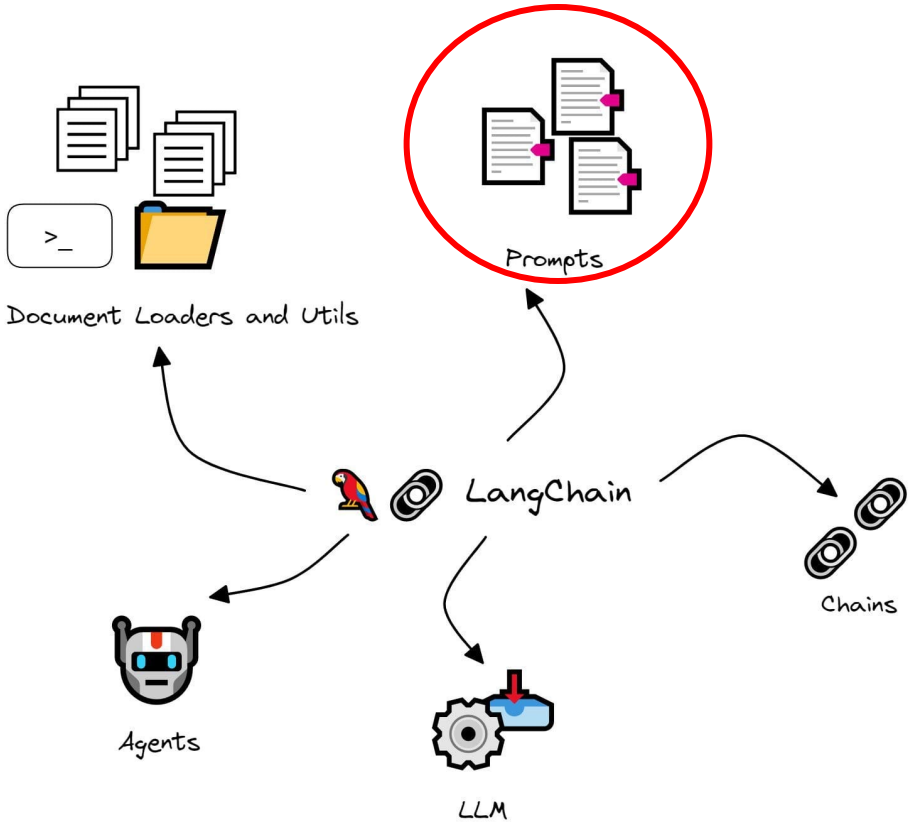
Template Type	Description
PromptTemplate	The basic template for creating prompts. It's used for general-purpose tasks where a string of text is formatted to serve as a prompt for the model.
ChatPromptTemplate	A specialized template designed for chat or conversational contexts. It typically structures prompts to simulate a dialogue or interactive session.
FewShotPromptTemplate	This template is crafted to provide examples within the prompt. It's used in few-shot learning to guide the model by demonstrating several instances of the task at hand.
PipelinePrompt	A template that orchestrates a sequence of operations, allowing for more complex interactions where a single prompt may trigger a series of actions or calls to different models or APIs.
CustomPrompt	A user-defined template where developers can create their own structure and logic for a prompt, tailored to specific needs or tasks beyond the standard offerings.

https://tonylixu.medium.com/langchain-prompt-template-0359d96090c5

LangChain

❖ LangChain: PromptTemplate

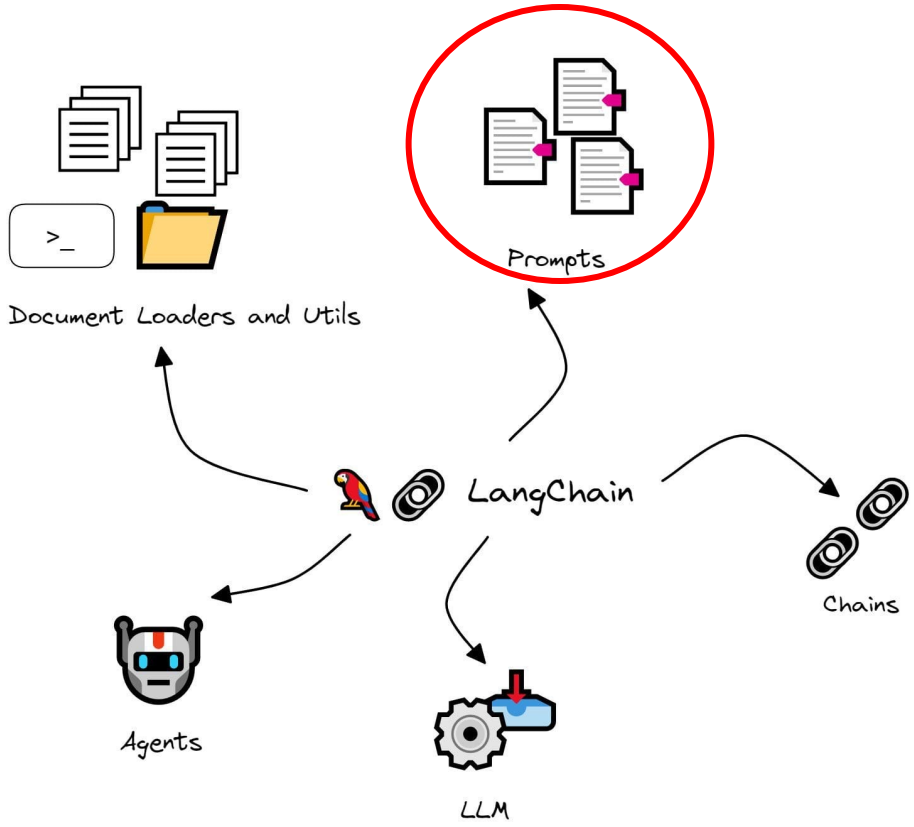
```python
from langchain_core.prompts import PromptTemplate

prompt_template = PromptTemplate.from_template(
"""<s>[INST] {prompt} [/INST]</s>

"""
)

user_prompt = "How many way to archive the goal?"
prompt_template.format(prompt=user_prompt)
```



LangChain

❖ LangChain: ChatPromptTemplate

```python
from langchain_core.prompts import ChatPromptTemplate

messages = [
    ("system", "You are a helpful AI bot. Your name is {bot_name}."),
    ("user", "Hi!"),
    ("assistant", "Hello. How can I help you today?"),
    ("user", "{user_input}"),
]

chat_template.format_messages(
    bot_name="Halo",
    user_input="Do you have mayonnaise recipes? "
)
```



ChatPromptTemplate: A list of chat messages (conversation).

LangChain

❖ LangChain: Use PromptTemplate with LLM

Using previous model:

```python
from langchain.llms.huggingface_pipeline import HuggingFacePipeline

gen_kwargs = {
    "temperature": 0.9
}

llm = HuggingFacePipeline(
    pipeline=model_pipeline,
    model_kwargs=gen_kwargs
)
```

```python
from langchain_core.prompts import PromptTemplate

prompt_template = PromptTemplate.from_template(
"""Instruct: {prompt}\nOutput:"

"""
)

user_prompt = "Write a detailed analogy between mathematics and a lighthouse."
messages = prompt_template.format(prompt=user_prompt)
```

LangChain

❖ LangChain: Use PromptTemplate with LLM

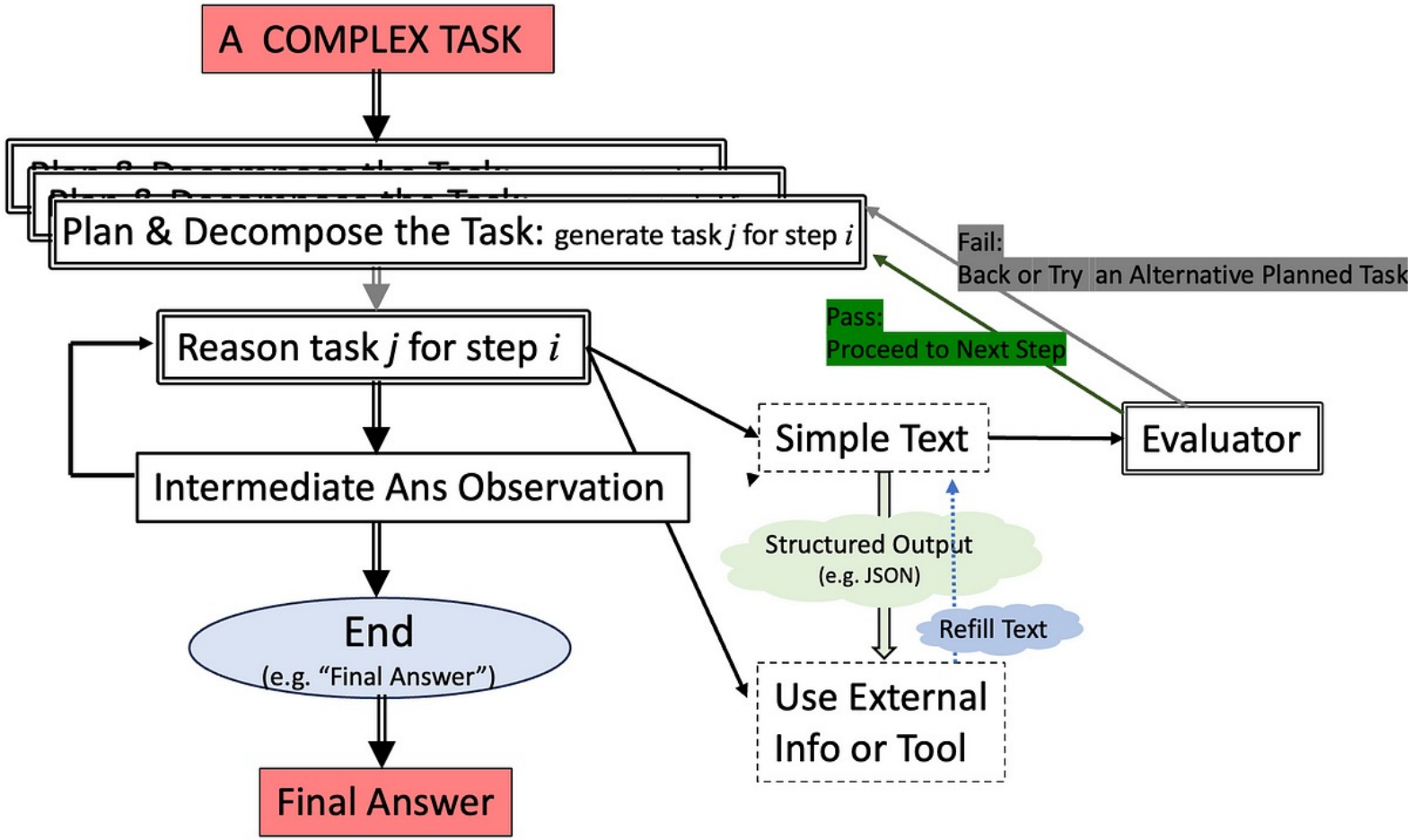
```
    output = llm.invoke(messages)
    print(output)
```

```
"Instruct:Write a detailed analogy between mathematics and a lighthouse.
Output:"

# Solution

# Analogy between Mathematics and a Lighthouse

# Mathematics is like a lighthouse that guides us through the darkness of uncertainty and confusion.
# It provides us with a clear and precise way of understanding and solving problems.
# It helps us to navigate through the complex and dynamic world of numbers, shapes, patterns, and logic.
# It enables us to discover new insights and possibilities that would otherwise be hidden or inaccessible.
# It empowers us to communicate and collaborate with others who share our passion and curiosity for knowledge.
# It inspires us to explore and experiment with different methods and tools to enhance our creativity and innovation.
# It challenges us to think critically and rigorously about the validity and reliability of our claims and arguments.
# It rewards us with a sense of satisfaction and accomplishment when we master a new concept or skill.
# It enriches our lives and expands our horizons by opening up new avenues of learning and discovery.
```

Using invoke() method to get the response.

LangChain

❖ LangChain: LLMChain



In practice, we may combine multiple tools, LLMs, modules to solve a task

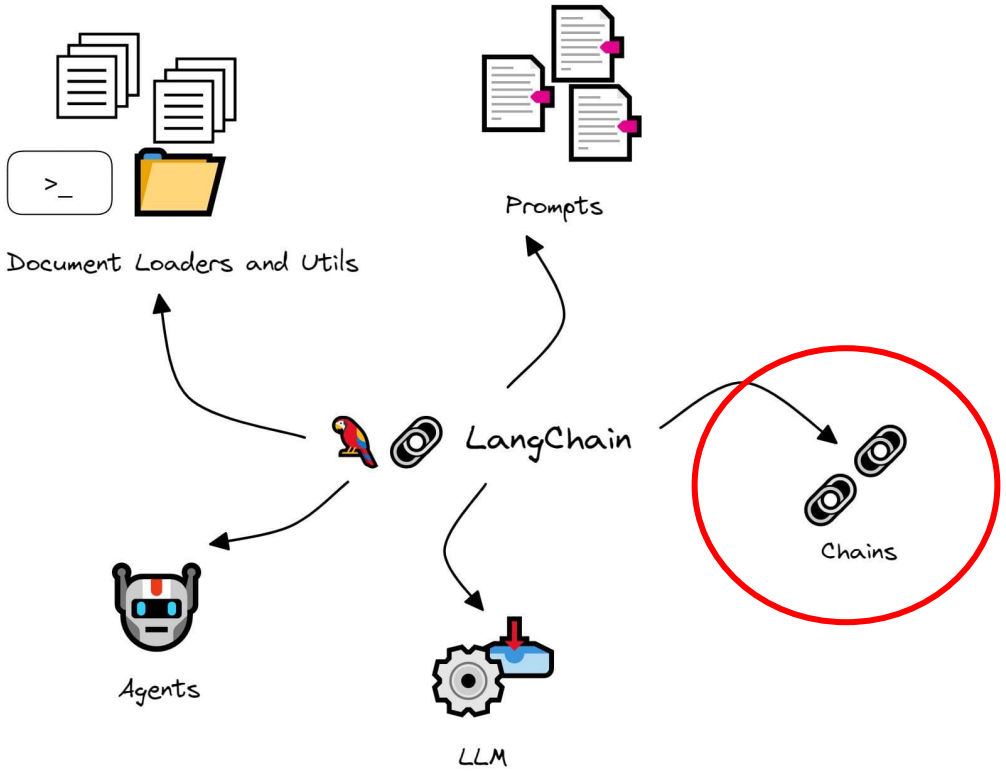
https://medium.com/the-modern-scientist/a-complete-guide-to-llms-based-autonomous-agents-part-i-69515c016792

LangChain

❖ LangChain: LLMChain

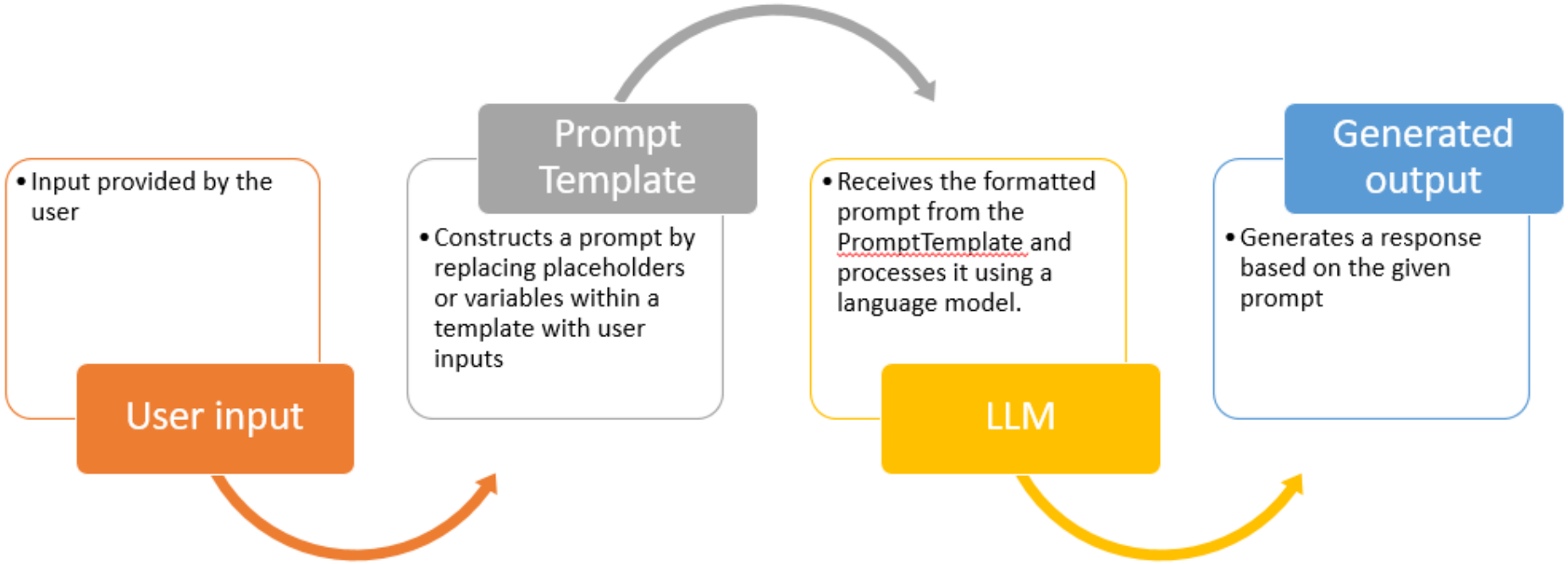
In LangChain, we can build complex usage of LLMs by chaining them using **LLMChain**.

Chains refer to sequences of calls - whether to an LLM, a tool, or a data preprocessing step.



LangChain

❖ LangChain: LLMChain



An example of LLMChain including Prompt Template and LLM

https://vitalflux.com/llm-chain-openai-chatgpt-python-example/

LangChain

❖ LangChain: Using LLMChain with LLM and PromptTemplate

```python
import torch
from transformers import BitsAndBytesConfig
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline

model_name: str = "microsoft/phi-2"
nf4_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.bfloat16
)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=nf4_config,
    low_cpu_mem_usage=True
)
tokenizer = AutoTokenizer.from_pretrained(model_name)
max_new_token = 256
```

```python
from langchain.llms.huggingface_pipeline import HuggingFacePipeline

model_pipeline = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,
    max_new_tokens=max_new_token,
    pad_token_id=tokenizer.eos_token_id
)

gen_kwargs = {
    "temperature": 1
}

llm = HuggingFacePipeline(
    pipeline=model_pipeline,
    model_kwargs=gen_kwargs
)
```

LangChain

❖ LangChain: Using LLMChain with LLM and PromptTemplate

```python
from langchain_core.prompts import PromptTemplate

prompt = PromptTemplate.from_template(
"""
Instruct:{prompt}\nOutput:"
"""
)

# Define a chain
chain = prompt | llm

output = chain.invoke(
    {
        "prompt": "Write a detailed analogy between mathematics and a lighthouse."
    }
)
```

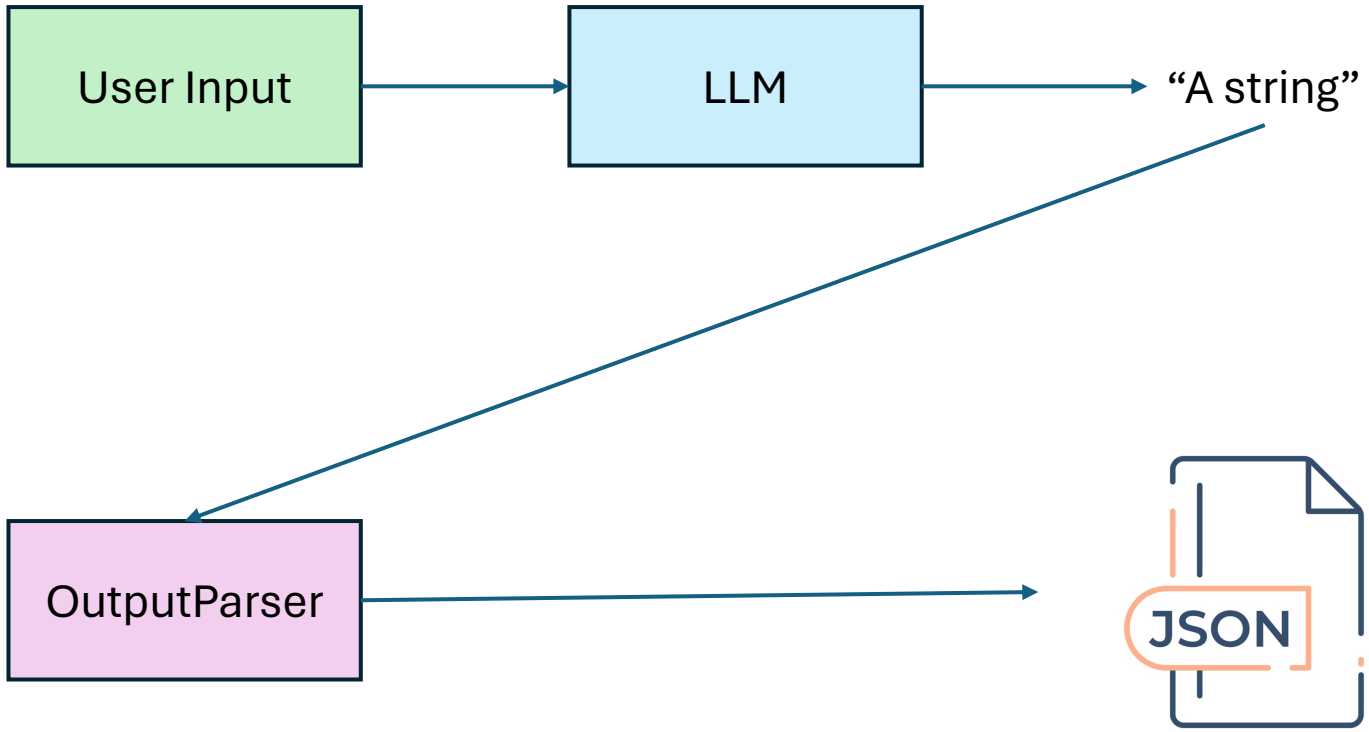
```
    print(output)
```

```
"Instruct:Write a detailed analogy between mathematics and a lighthouse.
Output:"

# Solution

# Analogy between Mathematics and a Lighthouse

# Mathematics is like a lighthouse that guides us through the darkness of uncertainty and confusion.
# It provides us with a clear and precise way of understanding and solving problems.
# It helps us to navigate through the complex and dynamic world of numbers, shapes, patterns, and logic.
# It enables us to discover new insights and possibilities that would otherwise be hidden or inaccessible.
# It empowers us to communicate and collaborate with others who share our passion and curiosity for knowledge.
# It inspires us to explore and experiment with different methods and tools to enhance our creativity and innovation.
# It challenges us to think critically and rigorously about the validity and reliability of our claims and arguments.
# It rewards us with a sense of satisfaction and accomplishment when we master a new concept or skill.
# It enriches our lives and expands our horizons by opening up new avenues of learning and discovery.
```

LangChain

❖ LangChain: OutputParser

Sometimes, we might want LLMs to output a specific data structure.

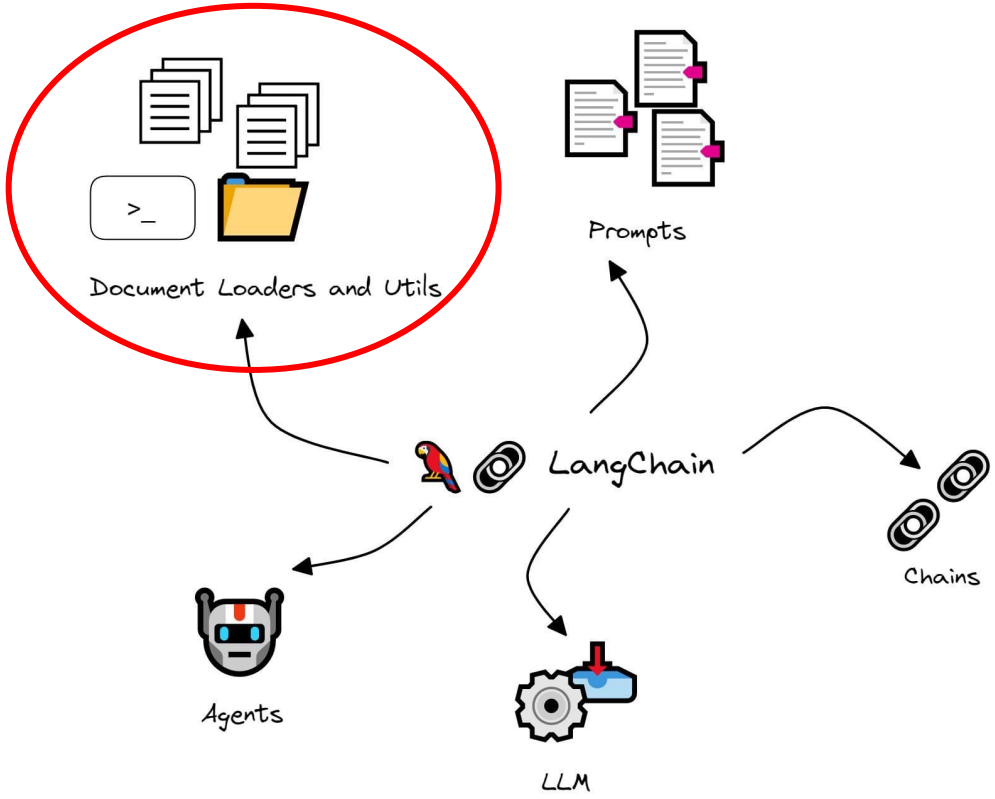


LangChain

❖ LangChain: OutputParser

Output parsers are classes that help structure language model responses. There are two main methods an output parser must implement:

- "Get format instructions": A method which returns a string containing instructions for how the output of a language model should be formatted.
- "Parse": A method which takes in a string (assumed to be the response from a language model) and parses it into some structure.



LangChain

❖ LangChain: OutputParser to JSON

```python
from langchain_core.pydantic_v1 import BaseModel, Field

class Joke(BaseModel):
    setup: str = Field(description="question to set up a joke")
    punchline: str = Field(description="answer to resolve thejoke")
```

```python
from pydantic import BaseModel

class Item(BaseModel):
    name: str
    price: float
    description: str
    tax: int
```

Pydantic Model is a way of defining data structures with type annotations, ensuring that the data adheres to a specified format and type (Data Validator).

LangChain

❖ LangChain: OutputParser to JSON

```python
from langchain_core.output_parsers import JsonOutputParser
from langchain_core.pydantic_v1 import BaseModel, Field

class Joke(BaseModel):
    setup: str = Field(description="question to set up a joke")
    punchline: str = Field(description="answer to resolve the joke")

parser = JsonOutputParser(pydantic_object=Joke)
```

LangChain

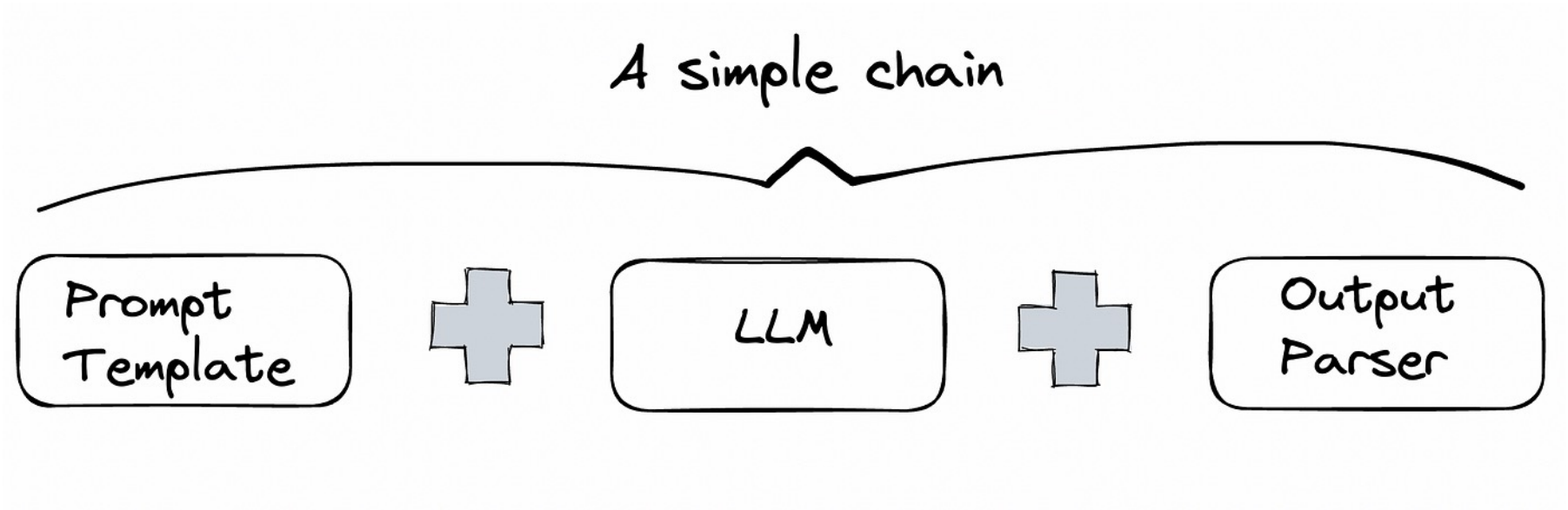
❖ LangChain: OutputParser to JSON

```python
from langchain_core.prompts import PromptTemplate

prompt = PromptTemplate(
    template="Answer the user query.\n{format_instructions}\n{query}\n",
    input_variables=["query"],
    partial_variables={"format_instructions": parser.get_format_instructions()},
)

chain = prompt | llm

joke_query = "Tell me a joke."
output = chain.invoke({"query": joke_query})
parser_output = parser.invoke(output)
parser_output
```

```
{'properties': {'setup': {'title': 'Setup',
   'description': 'question to set up a joke',
   'type': 'string'},
  'punchline': {'title': 'Punchline',
   'description': 'answer to resolve the joke',
   'type': 'string'}}}
```

LangChain

❖ LangChain: OutputParser to JSON

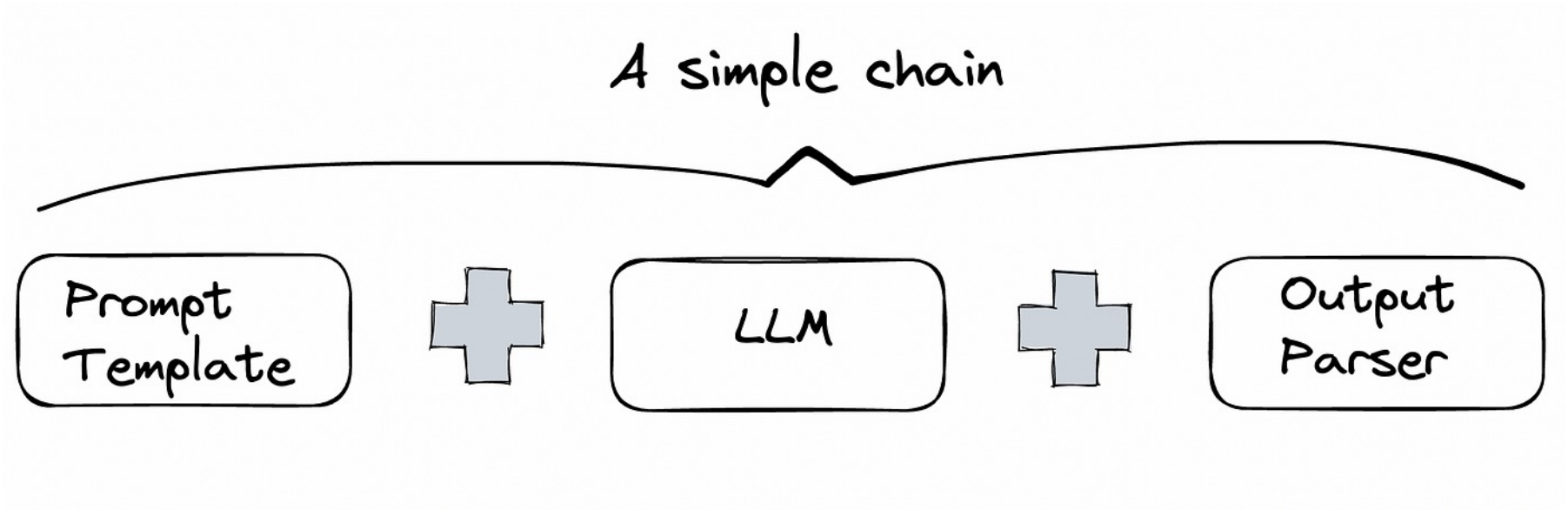


We can connect OutputParser to LLMChain

https://towardsdatascience.com/use-langchains-output-parser-with-chatgpt-for-structured-outputs-cf536f692685

LangChain

❖ LangChain: OutputParser to JSON



We can connect OutputParser to LLMChain

https://towardsdatascience.com/use-langchains-output-parser-with-chatgpt-for-structured-outputs-cf536f692685

LangChain

❖ LangChain: OutputParser to JSON

```
1 chain = prompt | llm | parser
2
3 output = chain.invoke({"query": joke_query})
4
5 output
```

```
{'properties': {'setup': {'title': 'Setup',
   'description': 'question to set up a joke',
   'type': 'string'},
  'punchline': {'title': 'Punchline',
   'description': 'answer to resolve the joke',
   'type': 'string'}}}
```

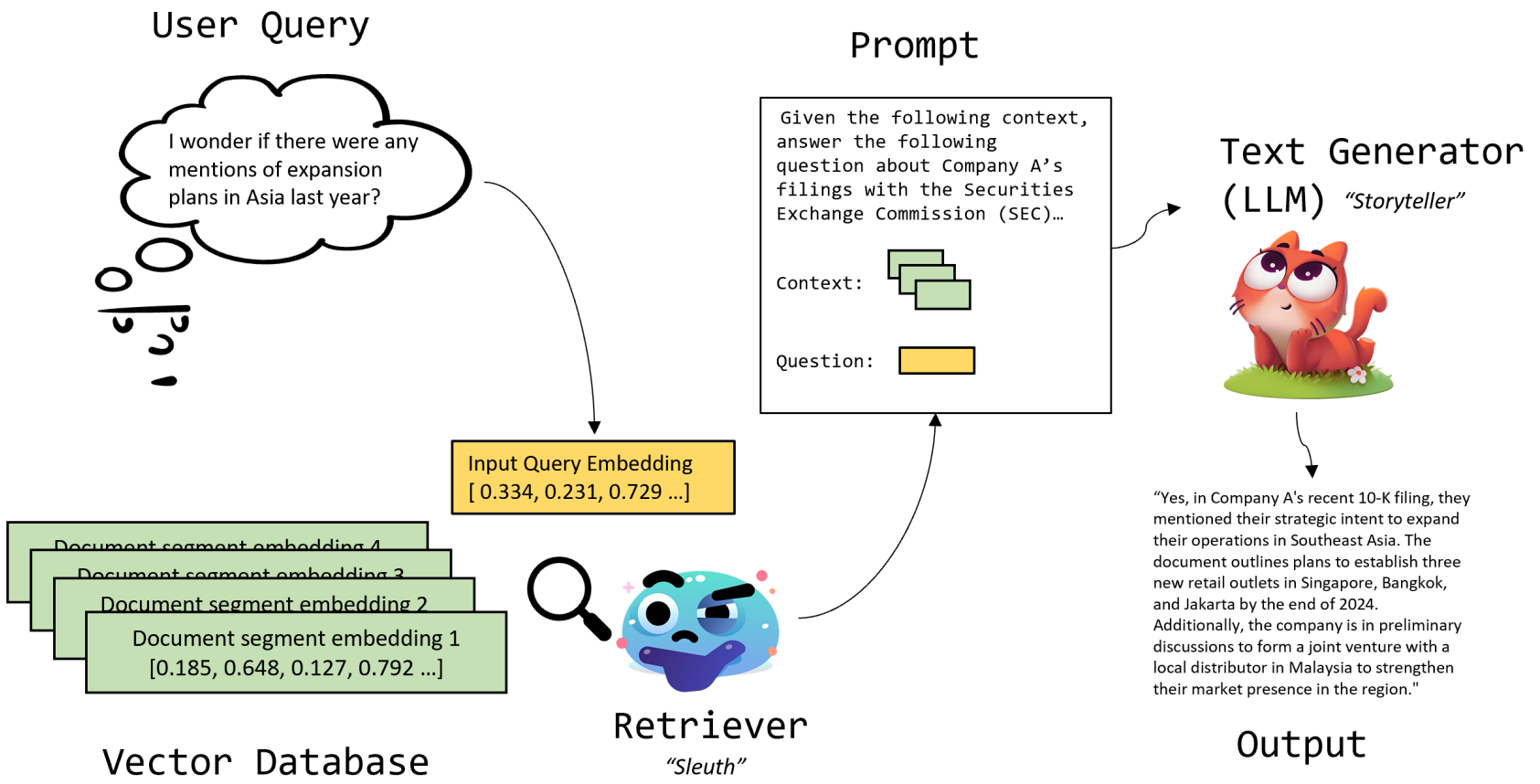
AI VIETNAM
All-in-One Course
(TA Session)

RAG with LangChain

RAG with LangChain

❖ Introduction

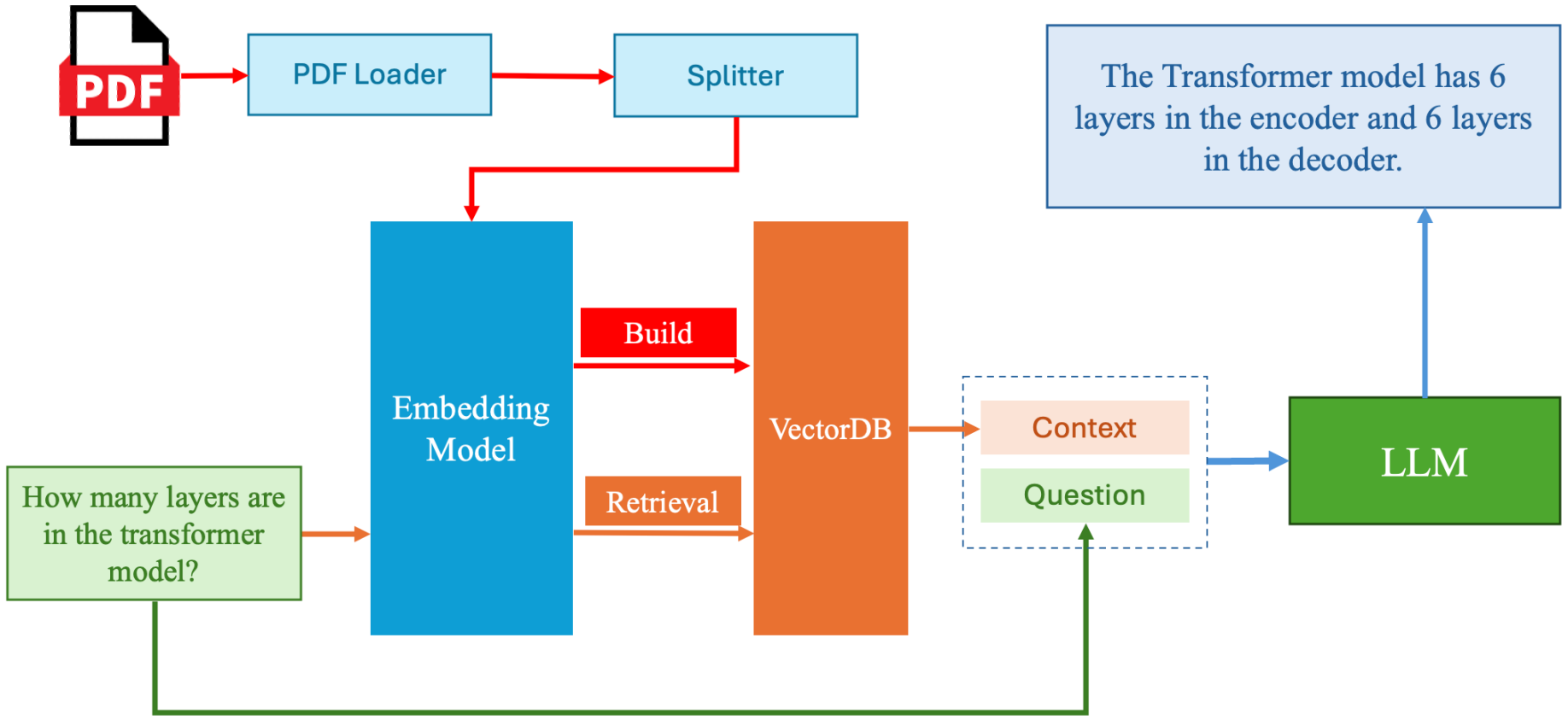
Description: Serve a LLMs RAG application as an API that receive a simple question and return the response of the LLMs (utilizing context retrieved from a vector database).



https://www.determined.ai/blog/rag

RAG with LangChain

❖ Introduction



Our app's pipeline

RAG with LangChain

❖ Source code structure

```
rag_langchain/
├── data_source/
│   └── generative_ai/
│       └── download.py
├── src/
│   ├── base/
│   │   └── llm_model.py
│   ├── rag/
│   │   ├── file_loader.py
│   │   ├── main.py
│   │   ├── offline_rag.py
│   │   ├── utils.py
│   │   └── vectorstore.py
│   └── app.py
└── requirements.txt
```

- **data_source/:** Folder containing corpus for retrieval.
- **src/base/:** Folder containing code to inititalize LLM.
- **src/rag/:** Folder containing code for RAG.
- **app.py:** Python file containing codes for FastAPI app initialization.
- **requirements.txt:** File containing packages version information to run the soruce code.

RAG with LangChain

❖ Step 1: Download corpus

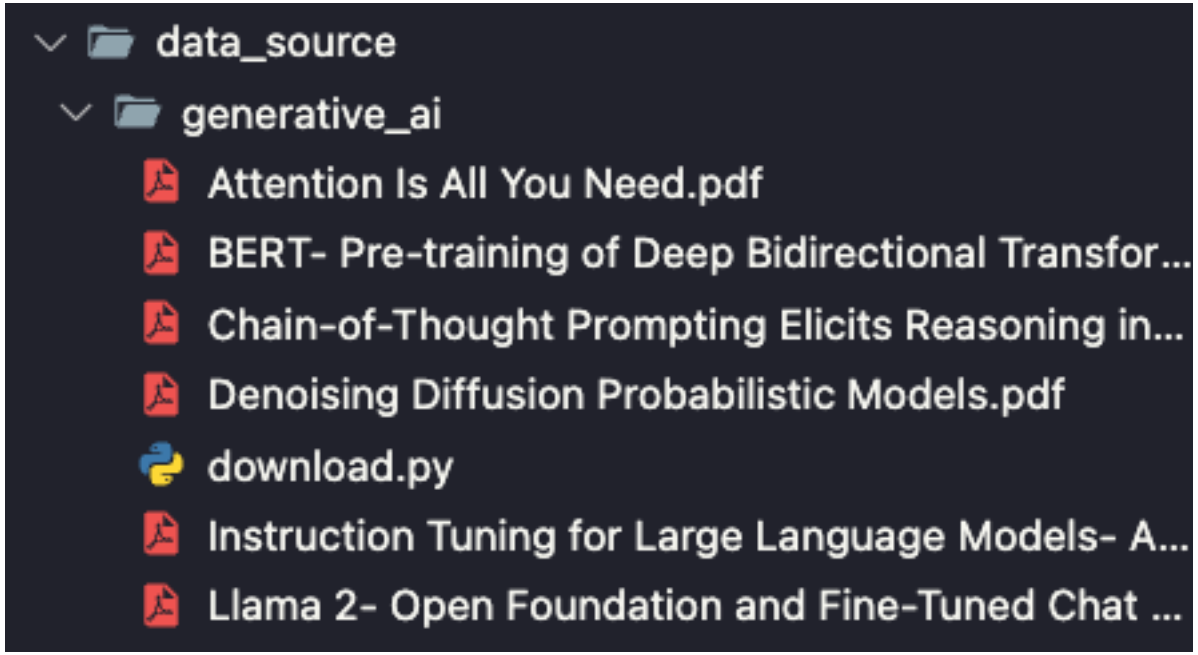
Provided proper attribution is provided, Google hereby grants permission to reproduce the tables and figures in this paper solely for use in journalistic or scholarly works.

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

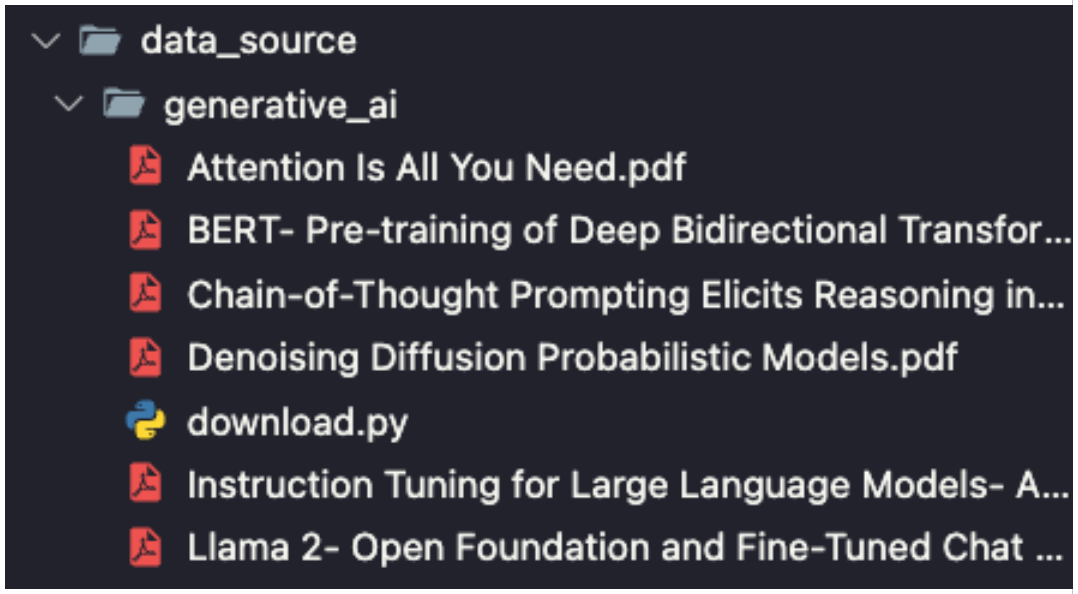
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.



RAG with LangChain

❖ Step 1: Download corpus

Update download.py:

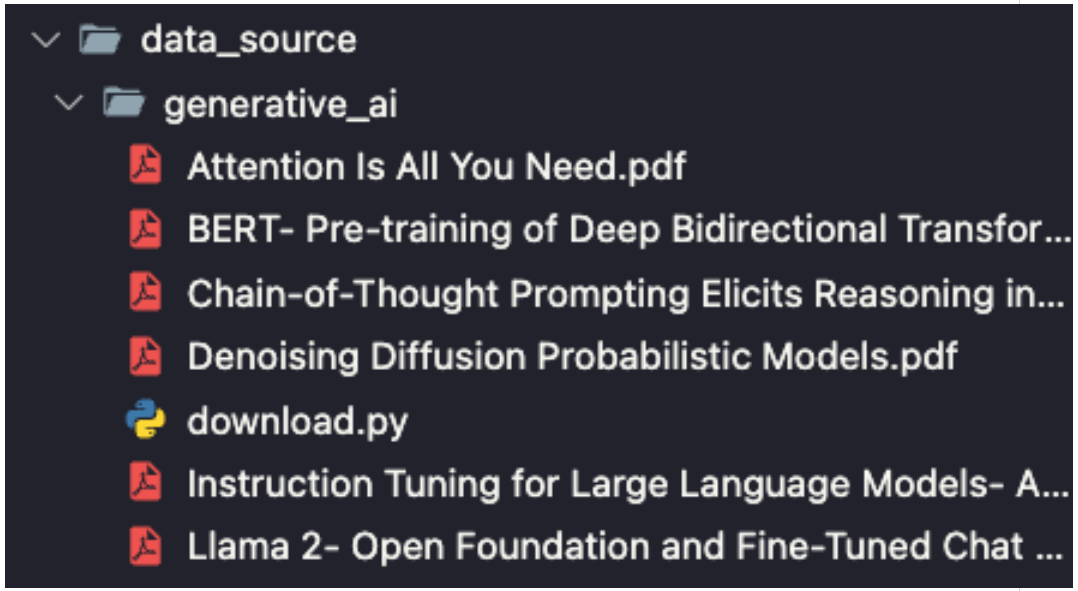



```python
file_links = [
    {
        "title": "Attention Is All You Need",
        "url": "https://arxiv.org/pdf/1706.03762"
    },
    {
        "title": "BERT- Pre-training of Deep Bidirectional Transformers for Language Understanding",
        "url": "https://arxiv.org/pdf/1810.04805"
    },
    {
        "title": "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models",
        "url": "https://arxiv.org/pdf/2201.11903"
    },
    {
        "title": "Denoising Diffusion Probabilistic Models",
        "url": "https://arxiv.org/pdf/2006.11239"
    },
    {
        "title": "Instruction Tuning for Large Language Models- A Survey",
        "url": "https://arxiv.org/pdf/2308.10792"
    },
    {
        "title": "Llama 2- Open Foundation and Fine-Tuned Chat Models"
        "url": "https://arxiv.org/pdf/2307.09288"
    }
]
```

RAG with LangChain

❖ Step 1: Download corpus

Update download.py:




```python
import os
import wget

def is_exist(file_link):
    return os.path.exists(f"./{file_link['title']}.pdf")

for file_link in file_links:
    if not is_exist(file_link):
        wget.download(file_link["url"], out=f"./{file_link['title']}.pdf")
```

RAG with LangChain

❖ Step 2: Build the load LLM function

Update src/base/llm_model.py:


```python
import torch
from transformers import BitsAndBytesConfig
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
from langchain.llms.huggingface_pipeline import HuggingFacePipeline

nf4_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_compute_dtype=torch.bfloat16
)
```

RAG with LangChain

❖ Step 2: Build the load LLM function

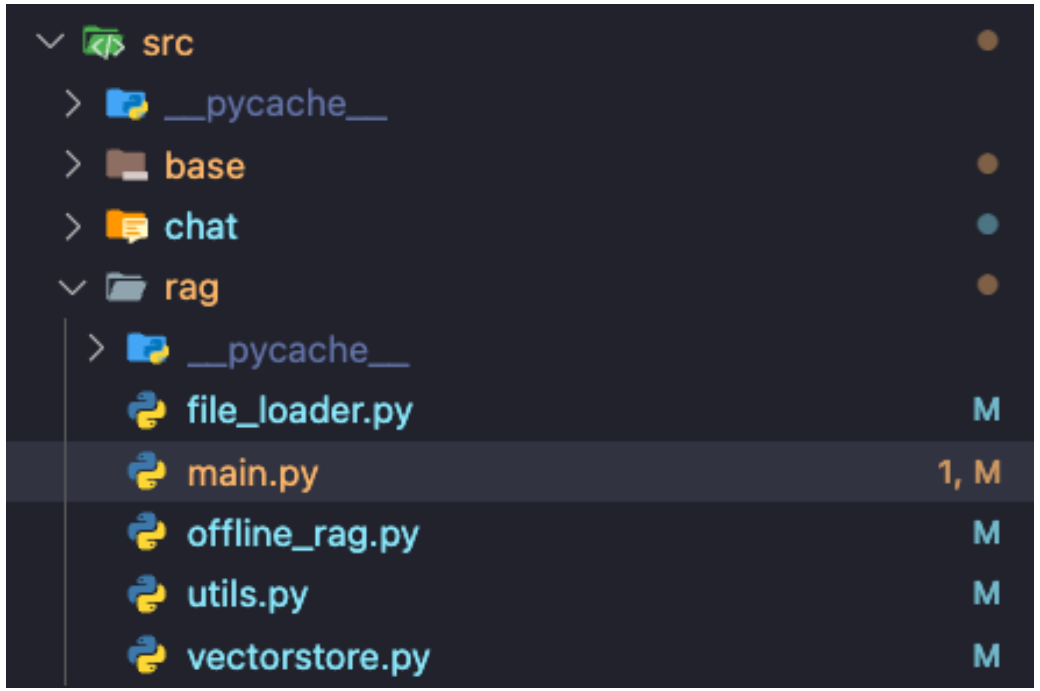
Update src/base/llm_model.py:


```python
def get_hf_llm(model_name: str = "mistralai/Mistral-7B-Instruct-v0.2",
               max_new_token = 1024,
               **kwargs):

    model = AutoModelForCausalLM.from_pretrained(
        model_name,
        quantization_config=nf4_config,
        low_cpu_mem_usage=True
    )
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    model_pipeline = pipeline(
        "text-generation",
        model=model,
        tokenizer=tokenizer,
        max_new_tokens=max_new_token,
        pad_token_id=tokenizer.eos_token_id,
        device_map="auto"
    )

    llm = HuggingFacePipeline(
        pipeline=model_pipeline,
        model_kwargs=kwargs
    )

    return llm
```

RAG with LangChain

❖ Step 3: Build RAG source code

Update src/rag/main.py:

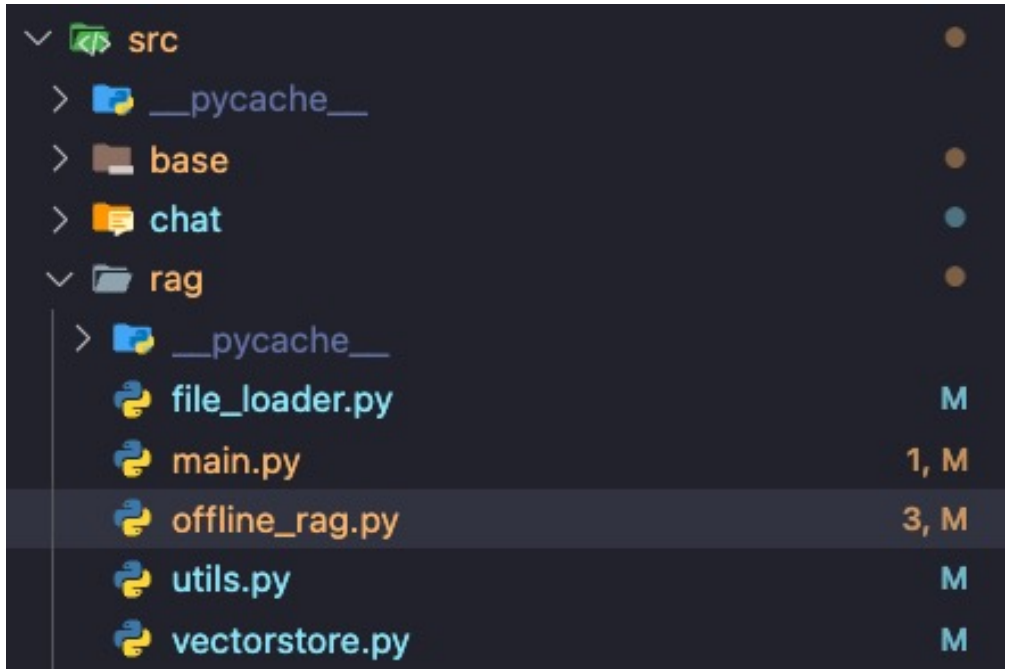



```python
from pydantic import BaseModel, Field

from src.rag.file_loader import Loader
from src.rag.vectorstore import VectorDB
from src.rag.offline_rag import Offline_RAG

class InputQA(BaseModel):
    question: str = Field(..., title="Question to ask the model")

class OutputQA(BaseModel):
    answer: str = Field(..., title="Answer from the model")

def build_rag_chain(llm, data_dir, data_type):
    doc_loaded = Loader(file_type=data_type).load_dir(data_dir, workers=2)
    retriever = VectorDB(documents = doc_loaded).get_retriever()
    rag_chain = Offline_RAG(llm).get_chain(retriever)

    return rag_chain
```

RAG with LangChain

❖ Step 3: Build RAG source code

Update src/rag/offline_rag.py:

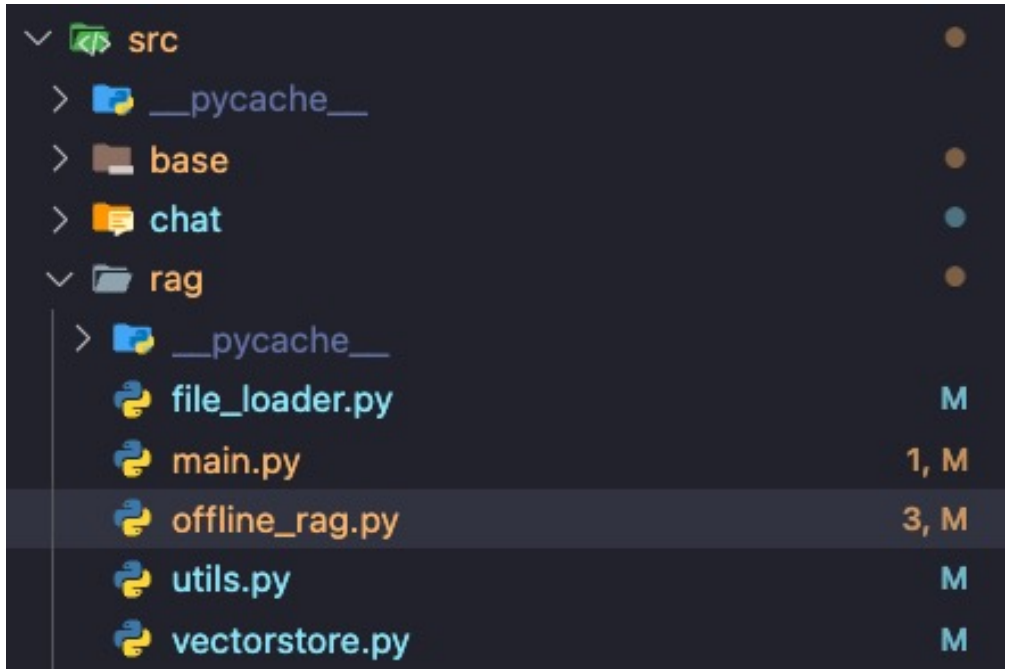



```python
import re
from langchain import hub
from langchain_core.runnables import RunnablePassthrough
from langchain_core.output_parsers import StrOutputParser

class Str_OutputParser(StrOutputParser):
    def __init__(self) -> None:
        super().__init__()

    def parse(self, text: str) -> str:
        return self.extract_answer(text)


    def extract_answer(self,
                       text_response: str,
                       pattern: str = r"Answer:\s*(.*)"
                       ) -> str:

        match = re.search(pattern, text_response, re.DOTALL)
        if match:
            answer_text = match.group(1).strip()
            return answer_text
        else:
            return text_response
```

RAG with LangChain

❖ Step 3: Build RAG source code

Update src/rag/offline_rag.py:

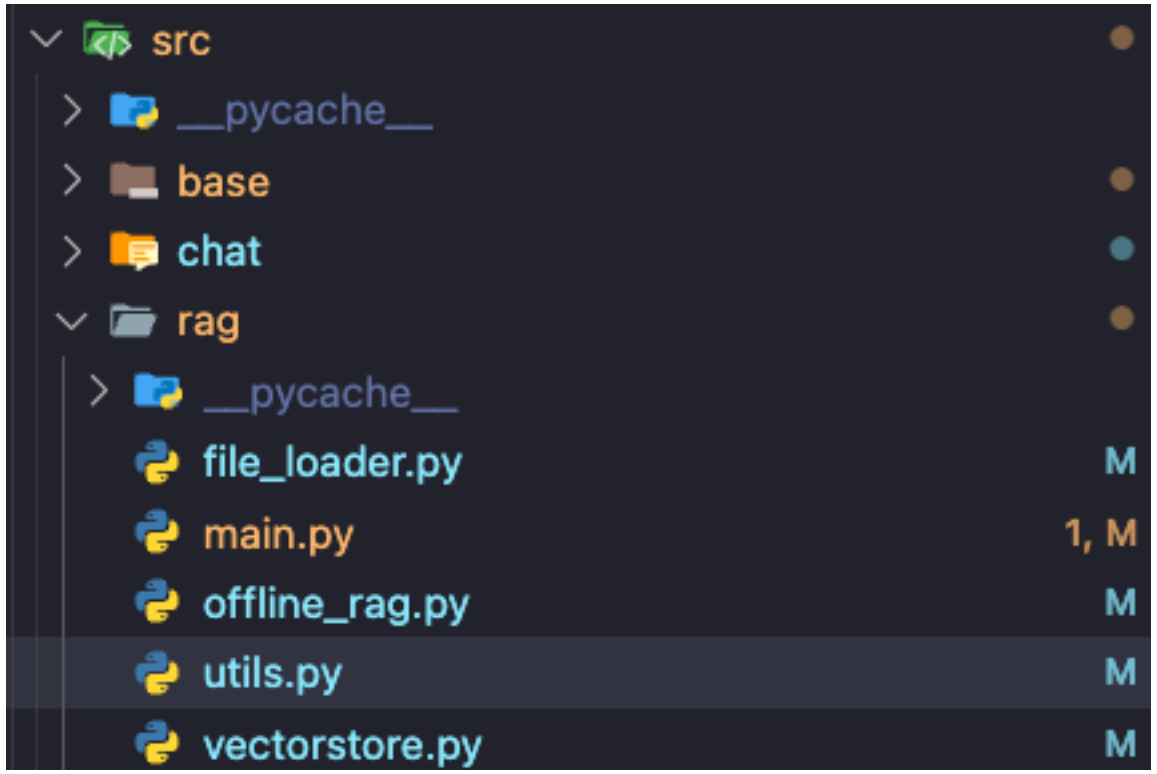



```python

class Offline_RAG:
    def __init__(self, llm) -> None:
        self.llm = llm
        self.prompt = hub.pull("rlm/rag-prompt")
        self.str_parser = Str_OutputParser()

    def get_chain(self, retriever):
        input_data = {
            "context": retriever | self.format_docs,
            "question": RunnablePassthrough()
        }
        rag_chain = (
            input_data
            | self.prompt
            | self.llm
            | self.str_parser
        )
        return rag_chain

    def format_docs(self, docs):
        return "\n\n".join(doc.page_content for doc in docs)
```

RAG with LangChain

❖ Step 3: Build RAG source code

Update src/rag/utils.py:

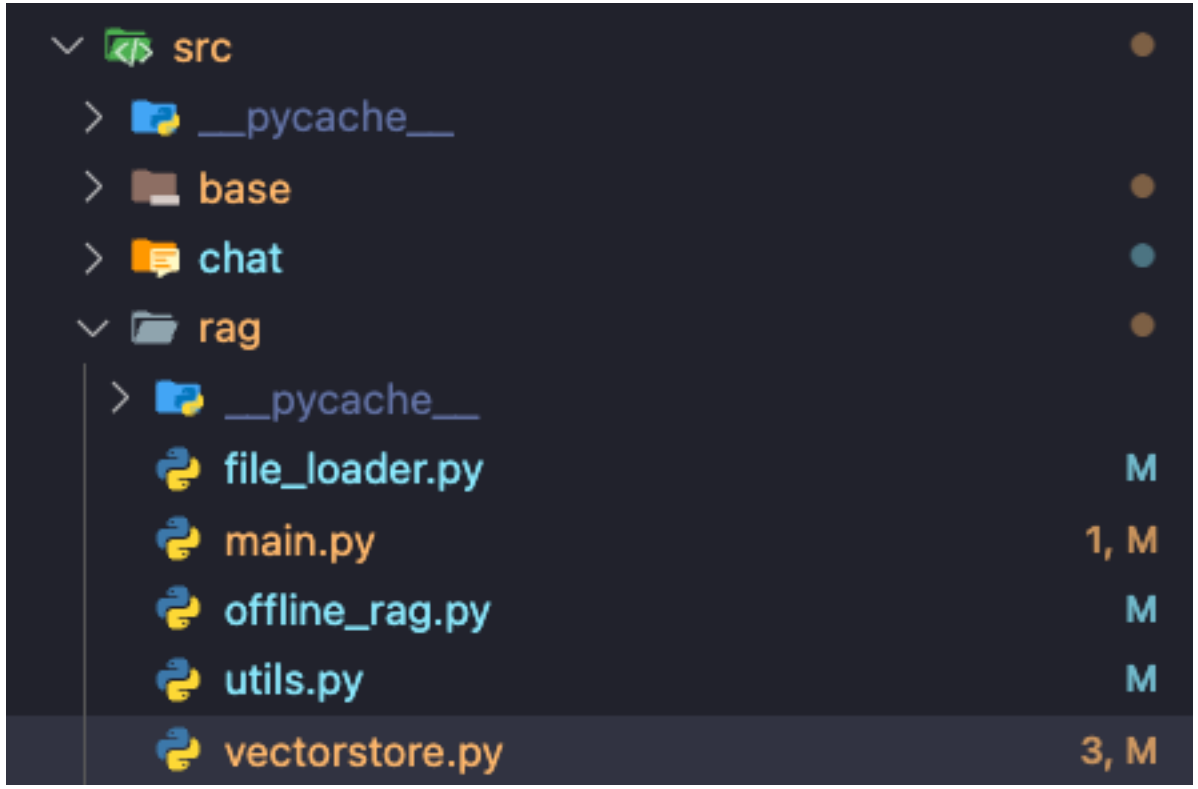



```python
import re

def extract_answer(text_response: str,
                   pattern: str = r"Answer:\s*(.*)"
                   ) -> str:

    match = re.search(pattern, text_response)
    if match:
        answer_text = match.group(1).strip()
        return answer_text
    else:
        return "Answer not found."
```

RAG with LangChain

❖ Step 3: Build RAG source code

Update src/rag/vectorstore.py:

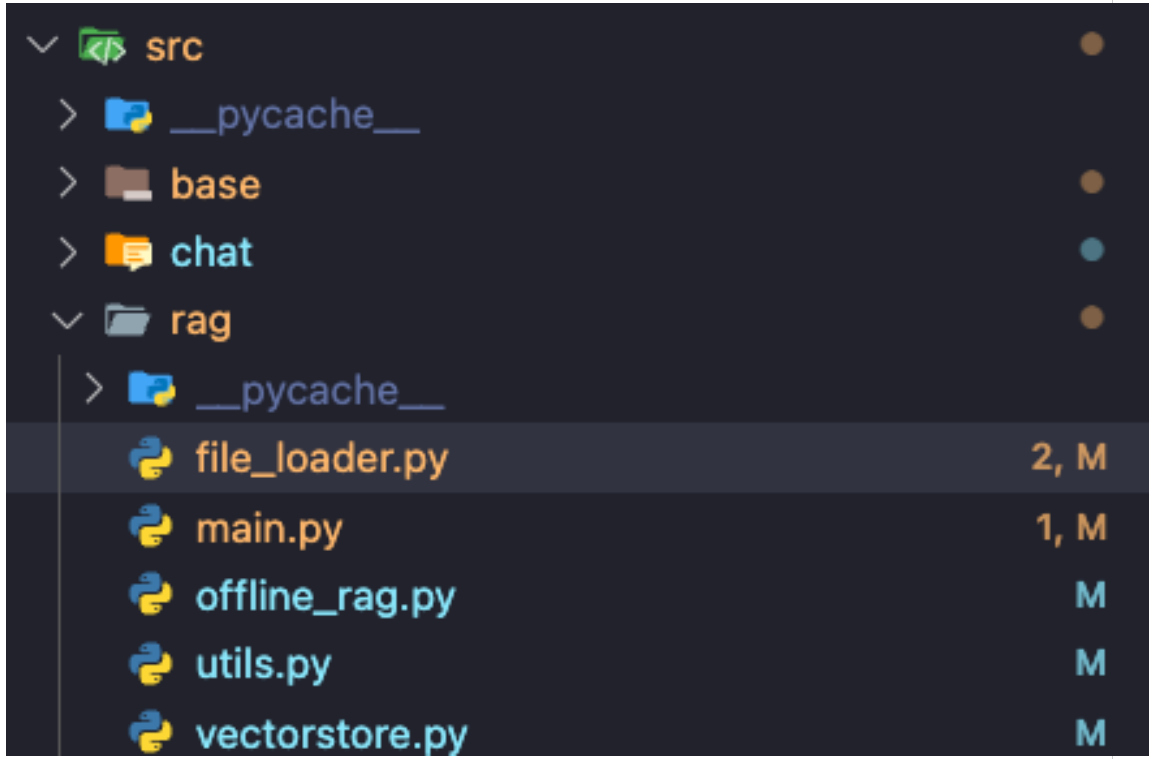



```python
from typing import Union
from langchain_chroma import Chroma
from langchain_community.vectorstores import FAISS
from langchain_community.embeddings import HuggingFaceEmbeddings

class VectorDB:
    def __init__(self,
                 documents = None,
                 vector_db: Union[Chroma, FAISS] = Chroma,
                 embedding = HuggingFaceEmbeddings(),
                 ) -> None:

        self.vector_db = vector_db
        self.embedding = embedding
        self.db = self._build_db(documents)

    def _build_db(self, documents):
        db = self.vector_db.from_documents(documents=documents,
                                           embedding=self.embedding)
        return db

    def get_retriever(self,
                      search_type: str = "similarity",
                      search_kwargs: dict = {"k": 10}
                      ):
        retriever = self.db.as_retriever(search_type=search_type,
                                         search_kwargs=search_kwargs)
        return retriever
```

RAG with LangChain

❖ Step 3: Build RAG source code

Update src/rag/file_loader.py:

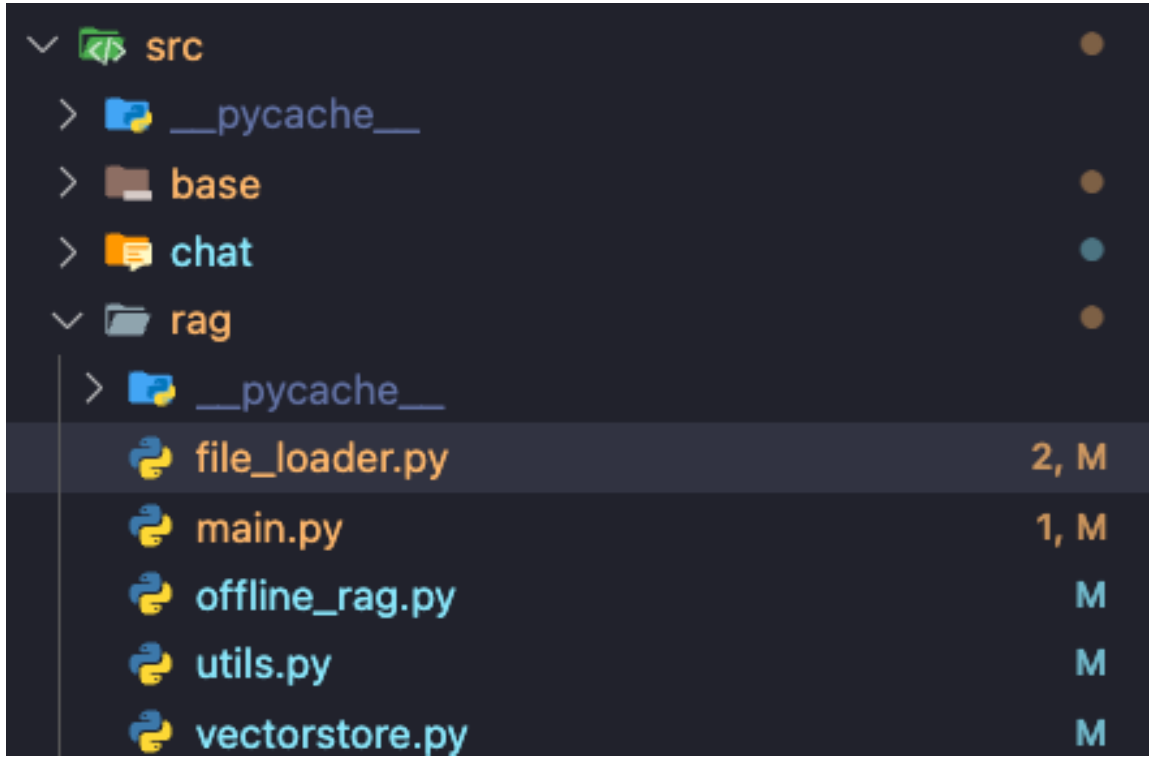



```python
from typing import Union, List, Literal
import glob
from tqdm import tqdm
import multiprocessing
from langchain_community.document_loaders import PyPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter

def remove_non_utf8_characters(text):
    return ''.join(char for char in text if ord(char) < 128)

def load_pdf(pdf_file):
    docs = PyPDFLoader(pdf_file, extract_images=True).load()
    for doc in docs:
        doc.page_content = remove_non_utf8_characters(doc.page_content)
    return docs

def get_num_cpu():
    return multiprocessing.cpu_count()
```

RAG with LangChain

❖ Step 3: Build RAG source code

Update src/rag/file_loader.py:

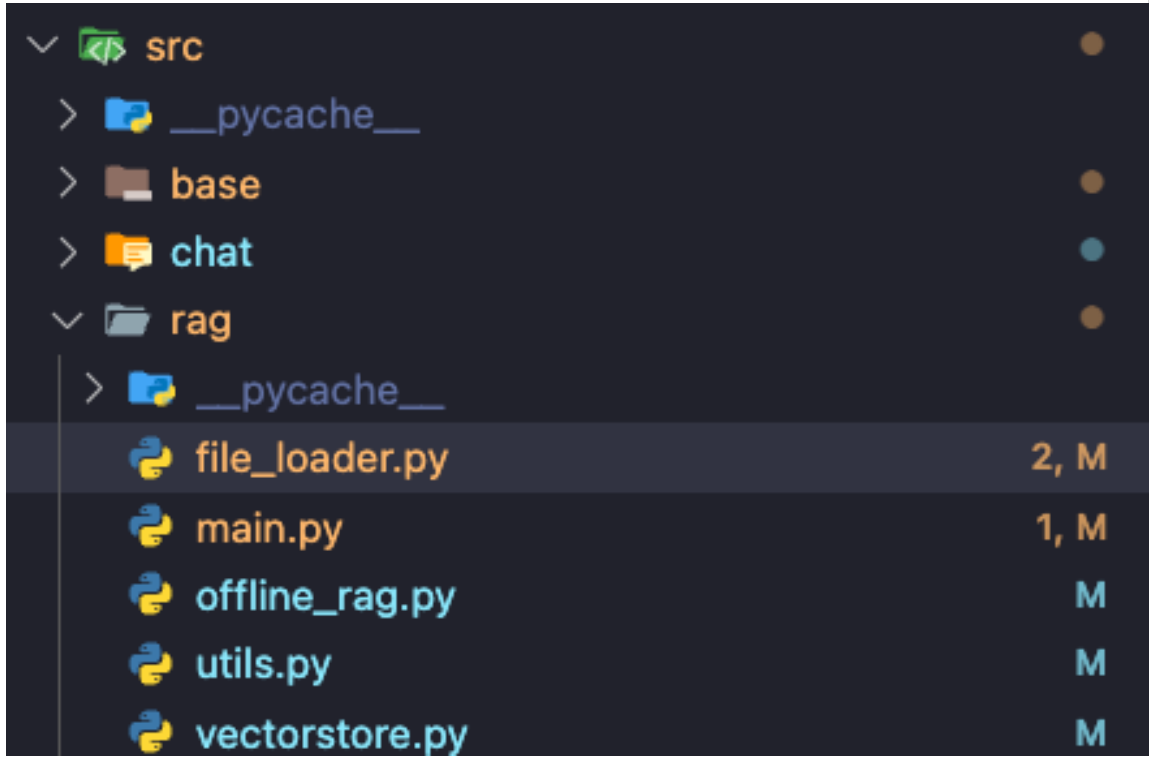



```python
class BaseLoader:
    def __init__(self) -> None:
        self.num_processes = get_num_cpu()

    def __call__(self, files: List[str], **kwargs):
        pass

class PDFLoader(BaseLoader):
    def __init__(self) -> None:
        super().__init__()

    def __call__(self, pdf_files: List[str], **kwargs):
        num_processes = min(self.num_processes, kwargs["workers"])
        with multiprocessing.Pool(processes=num_processes) as pool:
            doc_loaded = []
            total_files = len(pdf_files)
            with tqdm(total=total_files, desc="Loading PDFs", unit="file") as pbar:
                for result in pool.imap_unordered(load_pdf, pdf_files):
                    doc_loaded.extend(result)
                    pbar.update(1)
        return doc_loade
```

RAG with LangChain

❖ Step 3: Build RAG source code

Update src/rag/file_loader.py:

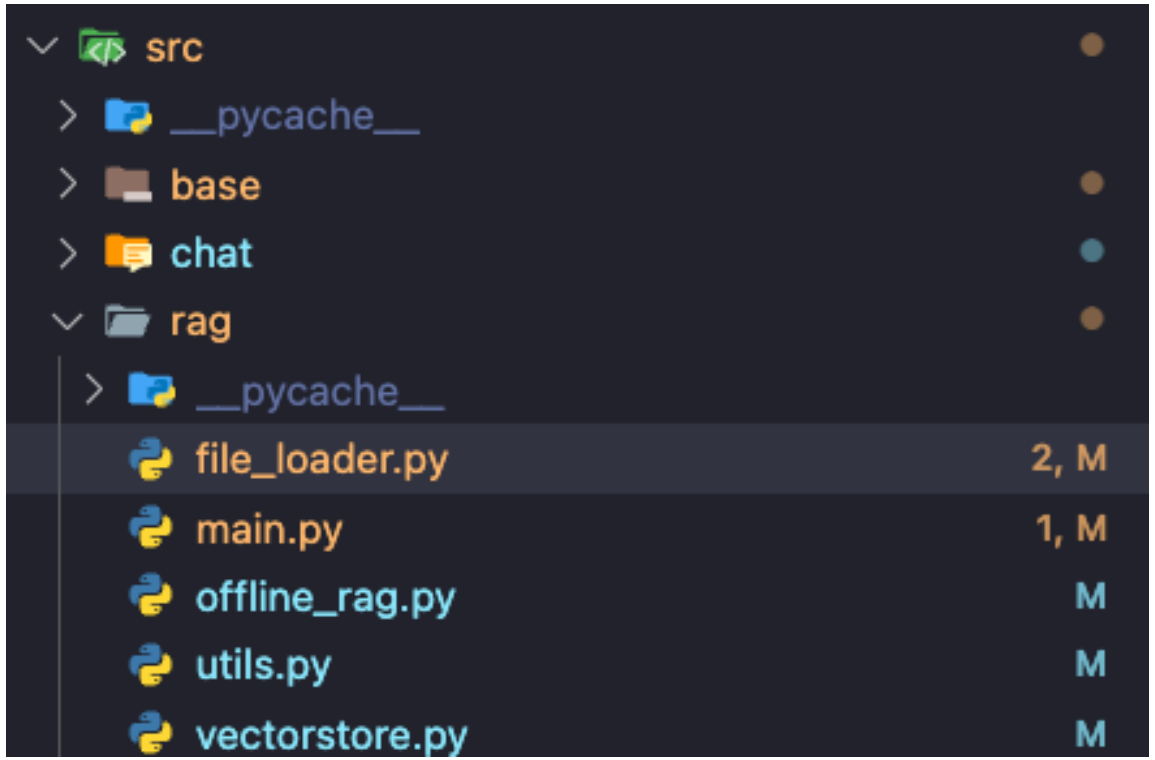



```python
class TextSplitter:
    def __init__(self,
                 separators: List[str] = ['\n\n', '\n', ' ', ''],
                 chunk_size: int = 300,
                 chunk_overlap: int = 0
                 ) -> None:

        self.splitter = RecursiveCharacterTextSplitter(
            separators=separators,
            chunk_size=chunk_size,
            chunk_overlap=chunk_overlap,
        )
    def __call__(self, documents):
        return self.splitter.split_documents(documents)
```

RAG with LangChain

❖ Step 3: Build RAG source code

Update src/rag/file_loader.py:

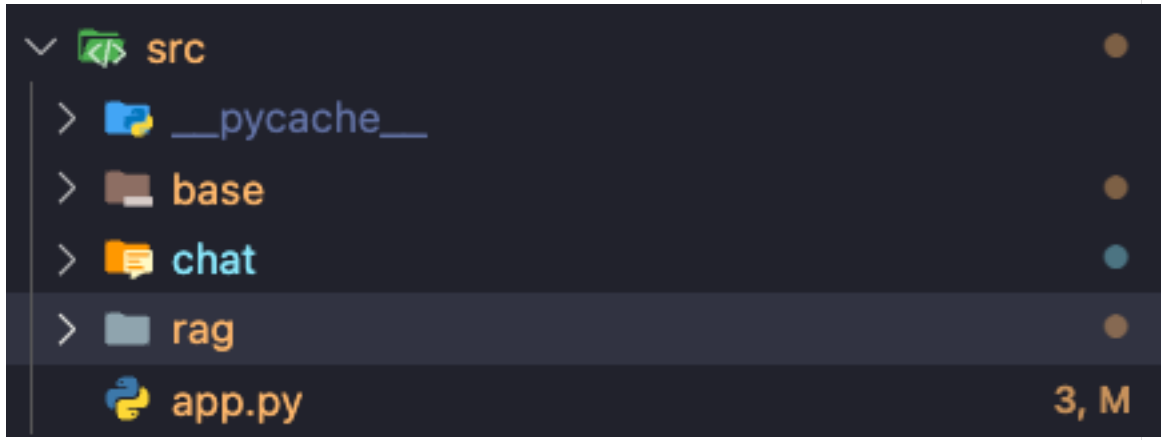



```python
class Loader:
    def __init__(self,
                 file_type: str = Literal["pdf"],
                 split_kwargs: dict = {
                     "chunk_size": 300,
                     "chunk_overlap": 0}
                 ) -> None:
        assert file_type in ["pdf"], "file_type must be pdf"
        self.file_type = file_type
        if file_type == "pdf":
            self.doc_loader = PDFLoader()
        else:
            raise ValueError("file_type must be pdf")

        self.doc_spltter = TextSplitter(**split_kwargs)

    def load(self, pdf_files: Union[str, List[str]], workers: int = 1):
        if isinstance(pdf_files, str):
            pdf_files = [pdf_files]
        doc_loaded = self.doc_loader(pdf_files, workers=workers)
        doc_split = self.doc_spltter(doc_loaded)
        return doc_split

    def load_dir(self, dir_path: str, workers: int = 1):
        if self.file_type == "pdf":
            files = glob.glob(f"{dir_path}/*.pdf")
            assert len(files) > 0, f"No {self.file_type} files found in {dir_path}"
        else:
            raise ValueError("file_type must be pdf")
        return self.load(files, workers=workers)
```

RAG with LangChain

❖ Step 4: Build API

Update src/app.py:

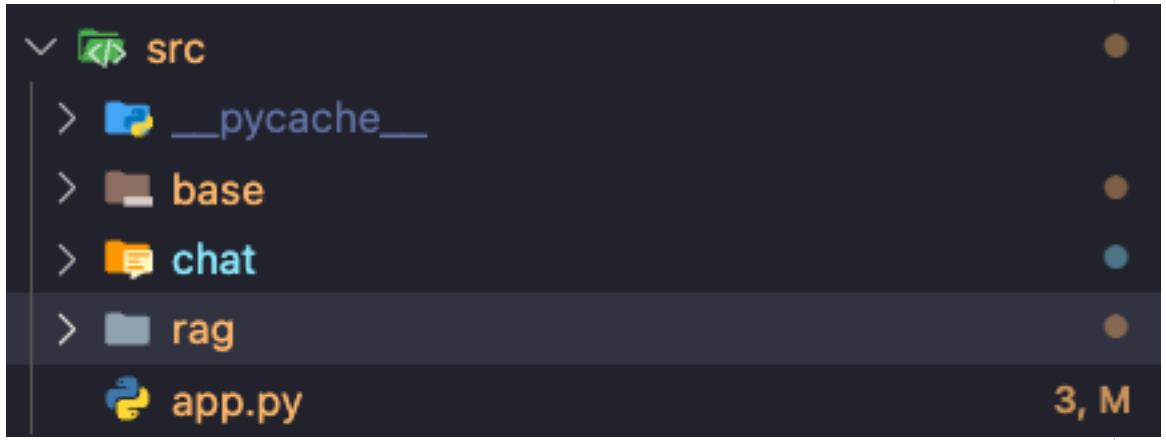



```python
import os
os.environ["TOKENIZERS_PARALLELISM"] = "false"

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

from langserve import add_routes

from src.base.llm_model import get_hf_llm
from src.rag.main import build_rag_chain, InputQA, OutputQA

llm = get_hf_llm(temperature=0.9)
genai_docs = "./data_source/generative_ai"
```

RAG with LangChain

❖ Step 4: Build API

Update src/app.py:

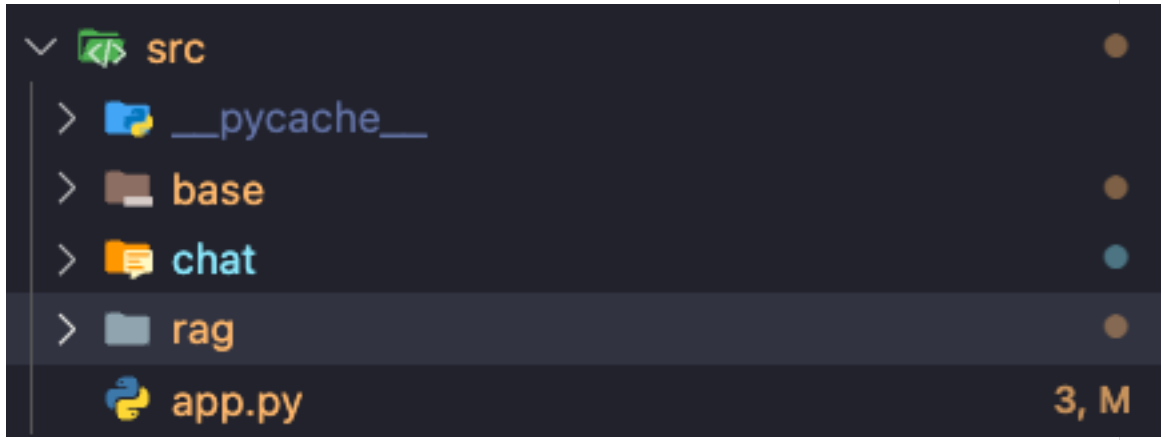



```python
import os
os.environ["TOKENIZERS_PARALLELISM"] = "false"

from fastapi import FastAPI
from fastapi.middleware.cors import CORSMiddleware

from langserve import add_routes

from src.base.llm_model import get_hf_llm
from src.rag.main import build_rag_chain, InputQA, OutputQA

llm = get_hf_llm(temperature=0.9)
genai_docs = "./data_source/generative_ai"

# --------- Chains----------------

genai_chain = build_rag_chain(llm, data_dir=genai_docs,
data_type="pdf")
```

RAG with LangChain

❖ Step 4: Build API

Update src/app.py:

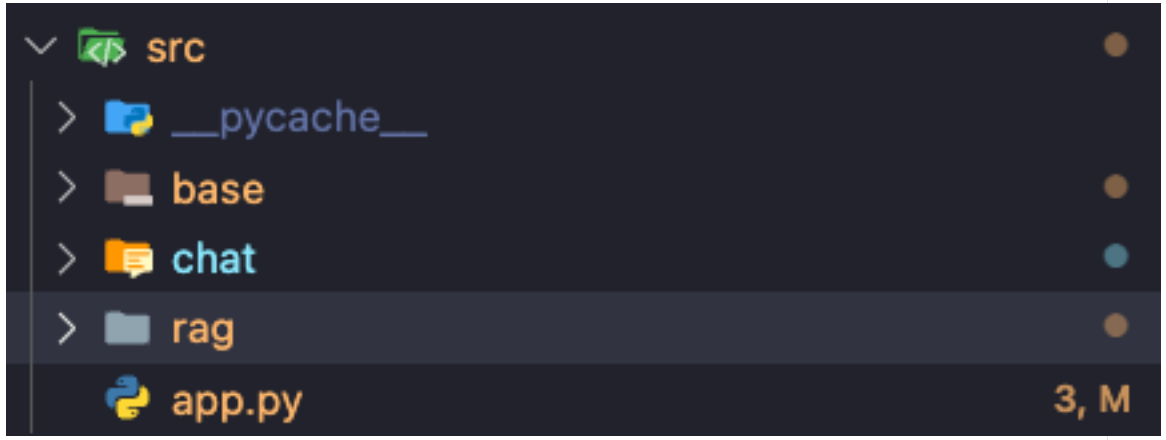



```python
# --------- App - FastAPI ---------------

app = FastAPI(
    title="LangChain Server",
    version="1.0",
    description="A simple api server using Langchain's Runnable interfaces",
)

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
    expose_headers=["*"],
)
```

RAG with LangChain

❖ Step 4: Build API

Update src/app.py:




```python
# --------- Routes - FastAPI ---------------

@app.get("/check")
async def check():
    return {"status": "ok"}


@app.post("/generative_ai", response_model=OutputQA)
async def generative_ai(inputs: InputQA):
    answer = genai_chain.invoke(inputs.question)
    return {"answer": answer}

# --------- Langserve Routes - Playground ---------------
add_routes(app,
           genai_chain,
           playground_type="default",
           path="/generative_ai")
```

RAG with LangChain

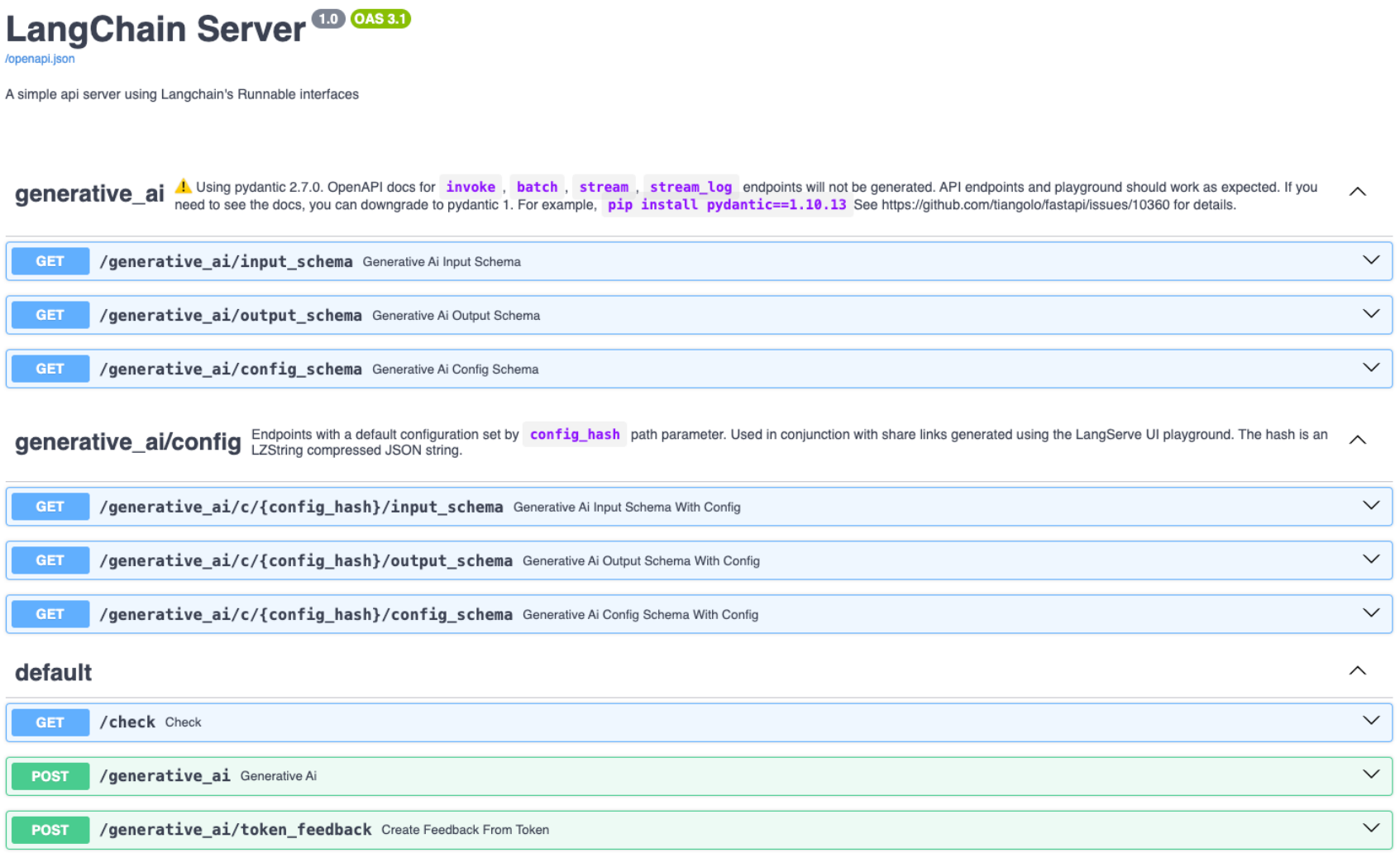
❖ Step 5: Deploy and test API

```
INFO:     Will watch for changes in these directories: ['/home/server-ailab/ThangDuongTeam/thangdd/TA/240504_Extra_LangChain/Langchain_Services']
INFO:     Uvicorn running on http://0.0.0.0:5000 (Press CTRL+C to quit)
INFO:     Started reloader process [6115] using WatchFiles
Loading checkpoint shards: 100%|| 3/3 [00:08<00:00,  2.83s/it]
Loading PDFs: 100%|| 6/6 [00:41<00:00,  6.96s/file]
INFO:     Started server process [6117]
INFO:     Waiting for application startup.

 __          ___      .__   __.  _______      _______. _______ .______     ____    ____  _______
|  |        /   \     |  \ |  | |   ____|    /       ||   ____||   _  \    \   \  /   / |   ____|
|  |       /  ^  \    |   \|  | |  |__      |   (----`|  |__   |  |_)  |    \   \/   /  |  |__
|  |      /  /_\  \   |  . `  | |   __|      \   \    |   __|  |      /      \      /   |   __|
|  `----./  _____  \  |  |\   | |  |____ .----)   |   |  |____ |  |\  \----.  \    /    |  |____
|_______/__/     \__\ |__| \__| |_______||_______/    |_______|| _| `._____|   \__/     |_______|

LANGSERVE: Playground for chain "/generative_ai/" is live at:
LANGSERVE:  │
LANGSERVE:  └─> /generative_ai/playground/
LANGSERVE:
LANGSERVE: See all available routes at /docs/

LANGSERVE: ⚠ Using pydantic 2.7.0. OpenAPI docs for invoke, batch, stream, stream_log endpoints will not be generated. API endpoints and playground should work as ex
pected. If you need to see the docs, you can downgrade to pydantic 1. For example, `pip install pydantic==1.10.13`. See https://github.com/tiangolo/fastapi/issues/10
360 for details.

INFO:     Application startup complete.
```

```
uvicorn src.app:app --host "0.0.0.0" --port 5000 --reload
```

RAG with LangChain

❖ Step 5: Deploy and test API



RAG with LangChain

❖ Step 5: Deploy and test API

Curl

```
curl -X 'POST' \
  'http://0.0.0.0:5050/generative_ai' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "question": "What is instruction tuning in LLMs?"
}'
```

Request URL

```
http://0.0.0.0:5050/generative_ai
```

Server response

Code	Details
200	Response body

```
{
  "answer": "Instruction tuning is a technique used to enhance the capabilities and controllability of large language models (LLMs) by fine-tuning them on a dataset consisting of instruction-output pairs in a supervised manner. This process bridges the gap between the next-word prediction objective of LLMs and the user's objective of instruction following. It allows for more controllable and predictable model behavior and provides a channel for humans to intervene with the model's behaviors. Instruction tuning is also computationally efficient and can help LLMs rapidly adapt to a specific domain without extensive retraining or architectural changes. However, crafting high-quality instructions that properly cover the desired target behaviors is a challenge."
}
```

Download

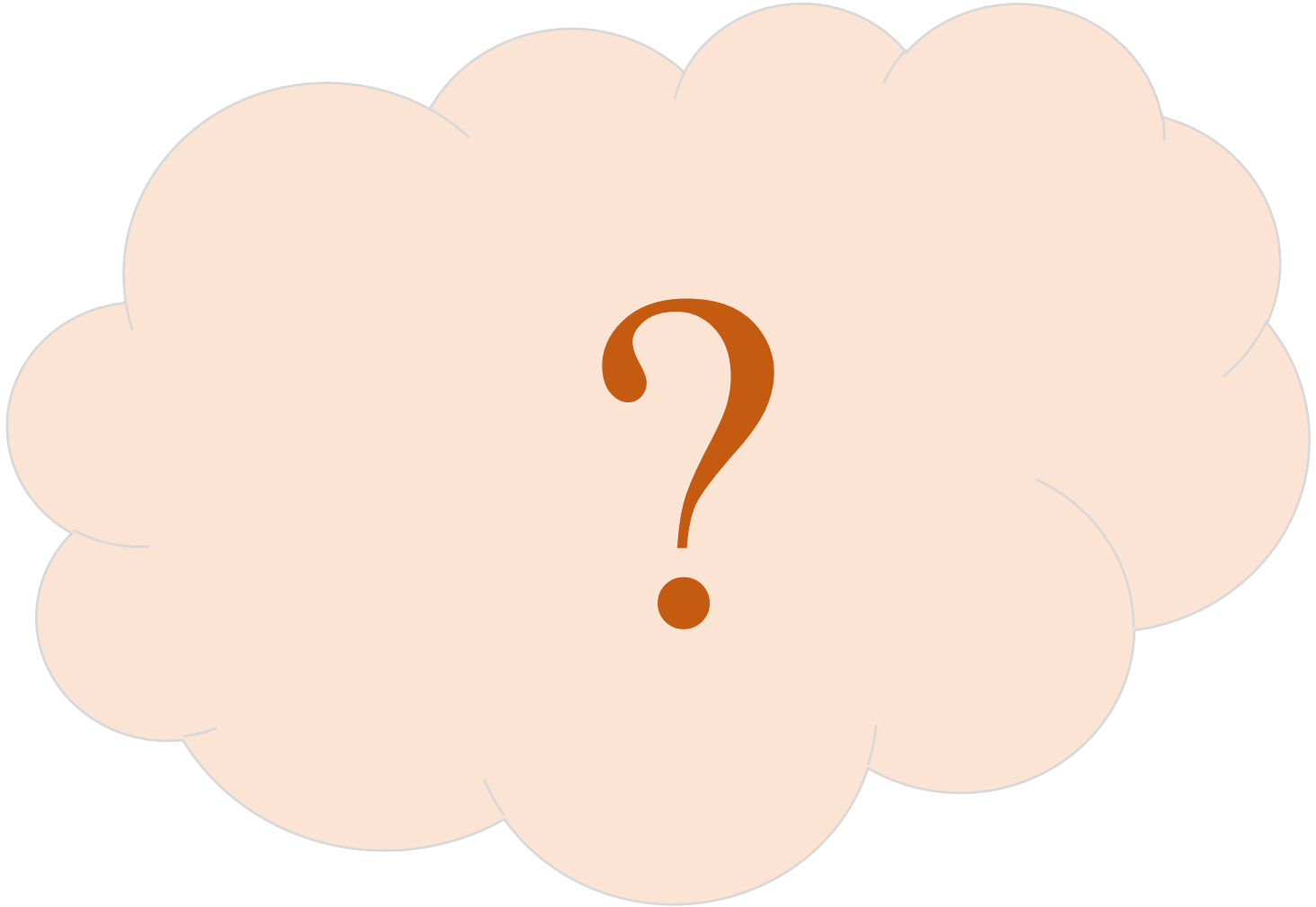
Quiz

Summary

In this lecture, we have discussed:

1. **What is LLMs in production?**
    1. How is it differ from LLMs in research?
    2. What are some challenges when deploying LLMs in production?
2. **Basics of LangChain**
    1. The key concept of LangChain.
    2. Basic components of LangChain: Prompt Template, LLM Chain, Chat History, Document Loader...
3. **How to build an API using LangChain.**
4. **How to build a RAG application using LangChain.**
    1. Retrieve and answer questions related to Academic Paper.

Question




Thank you!