LlamaIndex
Bottoms-Up Development

LlamaIndex
Bottoms-Up Development

Project Overview

The overall aim of the project is to build a pipeline for answering questions about the LlamaIndex documentation.

To do this, we will cover key components in LlamaIndex from the bottom-up, showing how things can be customized as we build our pipeline. This includes:

1. LLMs
2. Documents/Nodes and Data Loading
3. Embeddings
4. Retrievers
5. Node Postprocessors
6. Response Synthesizers
7. Indexes

Components

LLMs

OpenAI
HuggingFaceLLM
LangchainLLM
CustomLLM

Methods

complete() - basic text completion

chat() - basic chat response

stream_complete() - streaming completion, returns a generator

stream_chat() - streaming chat, returns a generator

In addition, there are “aXX” async versions of each method!

Low Level Usage

Completion
Chat

```
from llama_index.llms import OpenAI

llm = OpenAI(temperature=0, model=”gpt-3.5-turbo”, max_tokens=256)
response = llm.complete(“Tell me a joke!”)

print(response.text)
> “Sure, here's a classic one for you: Why don't scientists trust atoms? Because they make
up everything!”

print(response.raw)
> “[raw openai JSON response]”
```

Components

LLMs
Generate text for a given input.

Documents/Nodes
Customizable representations of data.

Embeddings
Create vector representations for text, used for search.

Retrievers
Retrieve nodes for a given query.

Node Postprocessors
Modify and parse the output nodes from a retriever.

Response Synthesizers
Generate a response to a query over retrieved text.

Indexes
Save and load your data with an index.

LLMs

OpenAI
HuggingFaceLLM
LangchainLLM
CustomLLM

Methods

complete() - basic text completion

chat() - basic chat response

stream_complete() - streaming completion, returns a generator

stream_chat() - streaming chat, returns a generator

In addition, there are "aXX" async versions of each method!

Low Level Usage

Completion

Chat

```
from llama_index.llms import OpenAI, ChatMessage

llm = OpenAI(temperature=0, model='gpt-3.5-turbo', max_tokens=256)
messages = [
    ChatMessage(role="system", content="Talk like a pirate in responses."),
    ChatMessage(role="user", content="Tell me a joke!")
]

response = llm.chat(messages)
print(response.text)
> assistant: Why did the pirate go to school? To improve his "arrrrrrrr"
```

Components

LLMs
Generate text for a given input.

Embeddings
Create vector representations for text, used for search.

Prompts
Structure your LLM inputs with pre-defined templates.

Pydantic Programs
A method for converting text to structured objects.

Retrievers
Retrieve nodes for a given query.

Response Synthesizer
Generate a response to a query over retrieved text.

Output-Parsing
Modify and parse the output of an LLM response.

Documents/Nodes
Customizable representations of data.

Embeddings

OpenAIEmbedding
LangchainEmbedding

Methods

get_text_embedding() - get embeddings for text

queue_text_for_embedding() - add text to a queue for generating embeddings

get_queued_text_embeddings() - return a list of embeddings for queued text

Low Level Usage

```
from llama_index.embeddings import OpenAIEmbedding

text = "Hello world!"

# uses text-embedding-ada-002
embed_model = OpenAIEmbedding()

embedding = embed_model.get_text_embedding(text)

print(embedding)
> [0.0065824370831251144, 0.0037026209756731987, ... ]
```

Components

LLMs
Generate text for a given input.

Embeddings
Create vector representations for text, used for search.

Prompts
Structure your LLM inputs with pre-defined templates.

Pydantic Programs
A method for converting text to structured objects.

Retrievers
Retrieve nodes for a given query.

Response Synthesizer
Generate a response to a query over retrieved text.

Output-Parsing
Modify and parse the output of an LLM response.

Documents/Nodes
Customizable representations of data.

Prompts

Completion Prompts

```
from llama_index.prompts import Prompt

prompt = Prompt("Hello {world}")
print(prompt.format(world="Earth"))
> Hello Earth

print(prompt.get_langchain_prompt())
> input_variables=['world'] output_parser=None partial_variables={} template='Hello {world}'
template_format='f-string' validate_template=True
```

Chat Prompts

```
from llama_index.prompts import Prompt
from langchain.prompts.chat import (
    AIMessagePromptTemplate,
    HumanMessagePromptTemplate,
    ChatPromptTemplate
)

??
```

Pydantic Programs

OpenAIPydantic Program

GuidancePydantic Program

LLMTextCompletion Program

Details

Pydantic programs can be seen as a simple way to get a structured output from an LLM.

Being able to convert text to a structured object easily means you always know what to expect in the output and how to parse it. Furthermore, with pydantic, you can directly define objects with their own methods and validations.

Low Level Usage

```
from pydantic import BaseModel
from llama_index.program import OpenAIPydanticProgram

class Joke(BaseModel):
    """"A setup and punchline for a joke.""""
    setup: str
    punchline: str

program = OpenAIPydanticProgram.from_defaults(output_cls=Joke, prompt_template_str=”Generate a joke inspired
by the topic {topic}.”)

output = program(topic=”Elephants”)
print(output.setup, “→”, output.punchline)
> Why don't elephants use computers? → Because they're afraid of the mouse!
```

Retrievers

BaseKeywordTableRetriever
KGTableRetriever
ListIndexRetriever
VectorIndexRetriever
TreeSelectLeafRetriever
DocumentSummaryIndexRetriever
VectorIndexAutoRetriever

Details

Retrievers use a query string to retrieve nodes. Depending on the type of index, this can take many forms.

Vector retrieval, tree traversal, and keyword lookup are some of the most common methods. Additionally, custom retrievers can be written to retrieve nodes any way you want!

Low Level Usage

Basic
Custom

```
from llama_index.indices.vector_store import VectorIndexRetriever
from llama_index import VectorStoreIndex, SimpleDirectoryReader

documents = SimpleDirectoryReader(“./data”).load_data()
index = VectorStoreIndex.from_documents(documents)

# retrieve using maximal marginal relevance (mmr)
retriever = VectorIndexRetriever(index, similarity_top_k=5, vector_store_query_mode=”mmr”)

nodes = retriever.retrieve(“What did the author do growing up?”)

print(nodes)
> [list of NodesWithScore objects]
```

Components

LLMs
Generate text for a given input.

Embeddings
Create vector representations for text, used for search.

Prompts
Structure your LLM inputs with pre-defined templates.

Pydantic Programs
A method for converting text to structured objects.

Retrievers
Retrieve nodes for a given query.

Response Synthesizer
Generate a response to a query over retrieved text.

Output-Parsing
Modify and parse the output of an LLM response.

Documents/Nodes
Customizable representations of data.

Retrievers

Custom Usage

Basic

Custom

```
from typing import List

from llama_index import QueryBundle
from llama_index.schema import NodeWithScore
from llama_index.retrievers import BaseRetriever

class CustomRetriever(BaseRetriever):
    """
    A simple retriever that replaces spaces with underscores.
    """
    def __init__(self, retriever: BaseRetriever) → None:
        self.retriever = retriever


    def _retrieve(self, query_bundle: QueryBundle) → List[NodeWithScore]:
        """Retrieve nodes given query."""

        nodes_with_score = self.retriever.retrieve(query_bundle)

        for n in nodes_with_score:
            n.node.text = node.text.replace(“ “, “_”)

        return nodes_with_score

nodes = CustomRetriever(index.as_retriever()).retrieve(“What did the author do growing up?”)

print(nodes)
```