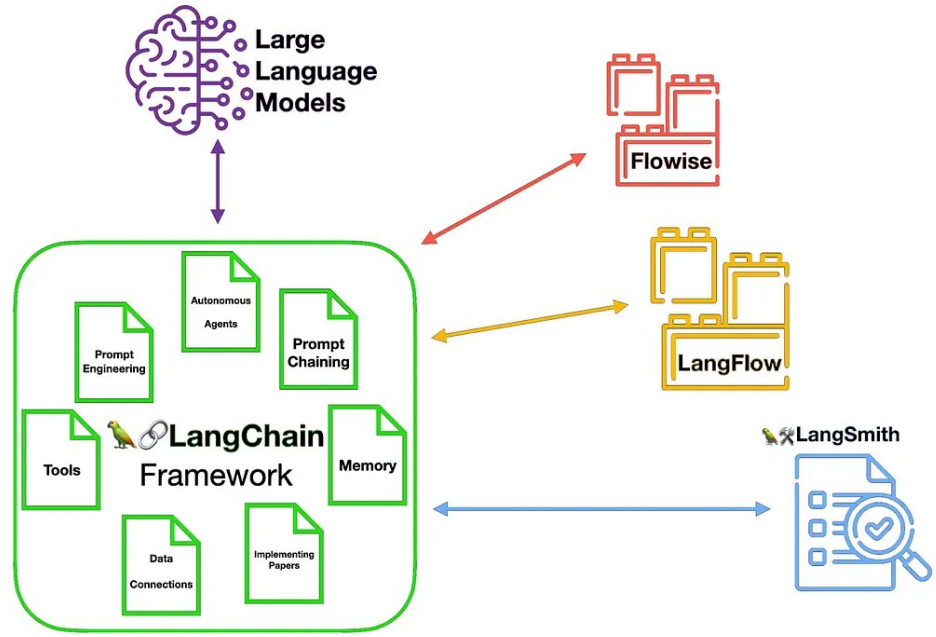
LangChain Ecosystem



Represented By:

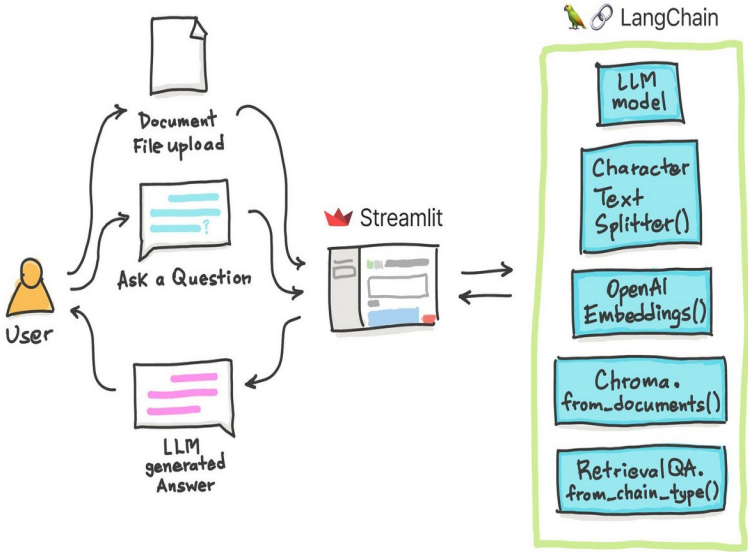
Ayush Sharma

What is Langchain?

Langchain is an open-source tool, ideal for enhancing chat models like GPT-4 or GPT-3.5. It connects external data seamlessly, making models More agentic and data-aware.

With Langchain, you can introduce fresh data to models like never before. The platform offers multiple chains, simplifying interactions with language models.

In addition to Langchain, tools like Models for creating vector embeddings play a crucial role. When dealing with Langchain, the capability to render images of a PDF file is also noteworthy. Now let's delve into the significance of text embeddings.



Text Embedding

Text embeddings are the heart and soul of Large Language Operations. Technically, we can work with language models with natural language but storing and retrieving natural language is highly inefficient. For example, in this project, we will need to perform high-speed search operations over large chunks of data. It is impossible to perform such operations on natural language data.

Text embeddings are the heart and soul of Large Language Operations. Technically, we can work with language models with natural language but storing and retrieving natural language is highly inefficient. For example, in this project, we will need to perform high-speed search operations over large chunks of data. It is impossible to perform such operations on natural language data.

To make it more efficient, we need to transform text data into vector forms. There arededicated ML models for creating embeddings from texts. The texts are converted into multidimensional vectors. Once embedded, we can group, sort, search, and more over these data. We can calculate the distance between two sentences to know how closely they are related. And the best part of it is these operations are not just limited to keywords like the traditional database searches but rather capture the semantic closeness of two sentences. This makes it a lot more powerful, thanks to Machine Learning.

Langchain Tools

Langchain has wrappers for all major vector databases like Chroma, Redis, Pinecone, Alpine db, and more. And same is true for LLMs, along with OpeanAI models, it also supports Cohere's models, GPT4ALL- an open-source alternative for GPT models. For embeddings, it provides wrappers for OpeanAI, Cohere, and HuggingFace embeddings. You can also use your custom embedding models as well.

Langchain is a meta-tool that abstracts away a lot of complications of interacting with underlying technologies, which makes it easier for anyone to build AI applications quickly.

Working Model of LangChain

It contains LangChain's core components:

LLM: Large language model, the core "brain"

Prompt: Users interact with the LLM via Prompts

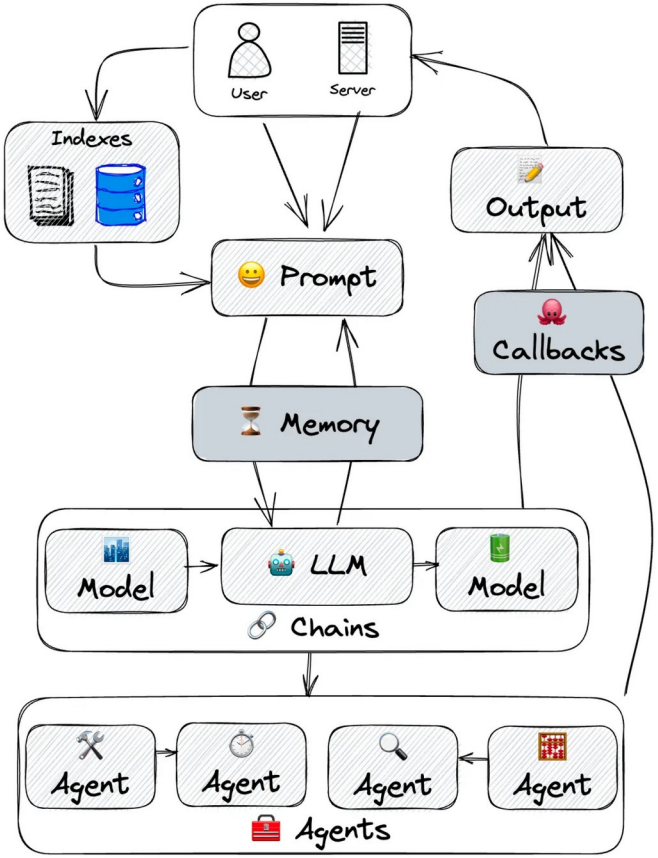
Memory: Enables the LLM to remember conversation history for multi-turn dialogues

Chains: Connect and chain multiple different models together to accomplish complex tasks

Agents: The LLM is the brain with thinking capabilities; Agents are the hands to take actions with the LLM (e.g. querying weather, drawing diagrams, etc.)

Indexes: LLMs are trained on public data with time limitations (e.g. ChatGPT's training data cuts off at 2021).

So knowledge coverage and freshness are restricted. Indexes provide retrieval capabilities when users want to build applications based on proprietary knowledge bases or business data, allowing the LLM to interact with them.



What are Large Language Models(LLMs)?

- A large language model is a computer program that learns and generates human-like language using a transformer architecture trained on vast text data.
- Large Language Models (LLMs) are foundational machine learning models that use deep learning algorithms to process and understand natural language. These models are trained on massive amounts of text data to learn patterns and entity relationships in the language.
- LLMs can perform many types of language tasks, such as translating languages, analyzing sentiments, chatbot conversations, and more. They can understand complex textual data, identify entities and relationships between them, and generate new text that is coherent and grammatically accurate.

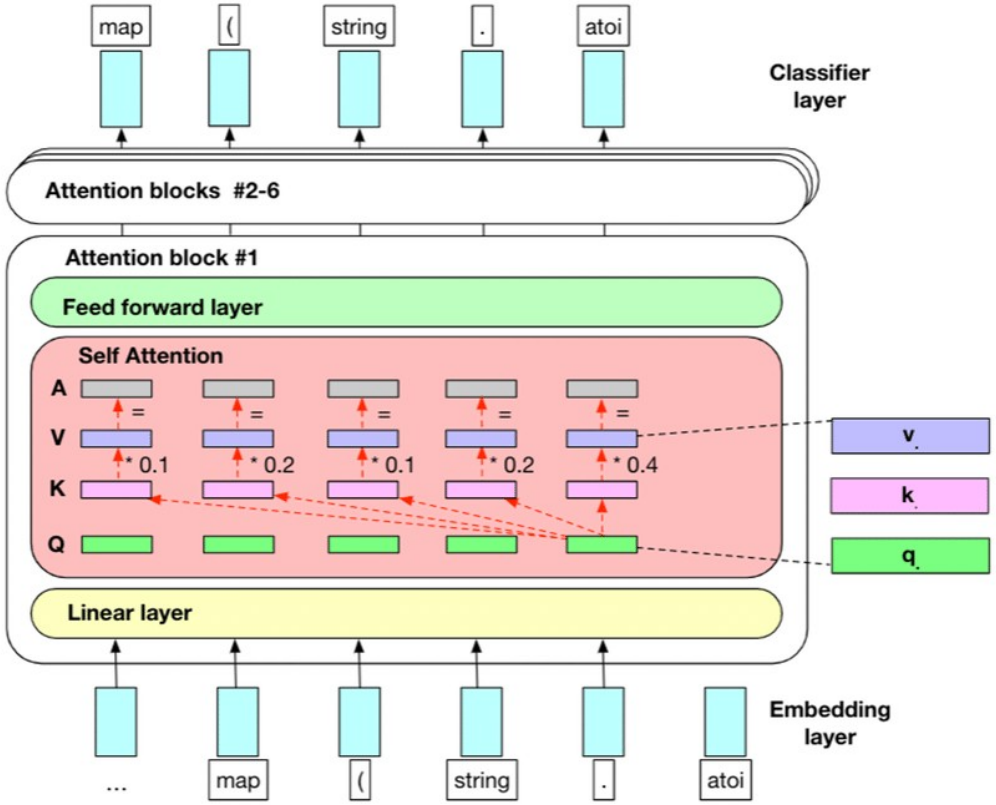
How a Large Language Model (LLM) Is Built?

- A large-scale transformer model known as a “large language model” is typically too massive to run on a single computer and is, therefore, provided as a service over an API or web interface.
- These models are trained on vast amounts of text data from sources such as books, articles, websites, and numerous other forms of written content.
- By analyzing the statistical relationships between words, phrases, and sentences through this training process, the models can generate coherent and contextually relevant responses to prompts or queries.

How do large language models work?

- Large language models like GPT-3 (Generative Pre-trained Transformer 3) work based on a transformer architecture. Here's a simplified explanation of how they Work:
- Learning from Lots of Text: These models start by reading a massive amount of text from the internet. It's like learning from a giant library of information.
- Innovative Architecture: They use a unique structure called a transformer, which helps them understand and remember lots of information.
- Breaking Down Words: They look at sentences in smaller parts, like breaking words into pieces. This helps them work with language more efficiently.
- Understanding Words in Sentences: Unlike simple programs, these models understand individual words and how words relate to each other in a sentence. They get the whole picture.
- Getting Specialized: After the general learning, they can be trained more on specific topics to get good at certain things, like answering questions or writing about particular subjects.
- Doing Tasks: When you give them a prompt (a question or instruction), they use what they've learned to respond. It's like having an intelligent assistant that can understand and generate text.

Working Model LLM



Word Embedding

When building a large language model, word embedding is a crucial first step. This involves representing words as vectors in a high-dimensional space where similar words are grouped together. This helps the model to understand the meaning of words and make predictions based on that understanding.

Vocabulary:
Man, woman, boy, girl, prince, princess, queen, king, monarch

	1	2	3	4	5	6	7	8	9
man	1	0	0	0	0	0	0	0	0
woman	0	1	0	0	0	0	0	0	0
boy	0	0	1	0	0	0	0	0	0
girl	0	0	0	1	0	0	0	0	0
prince	0	0	0	0	1	0	0	0	0
princess	0	0	0	0	0	1	0	0	0
queen	0	0	0	0	0	0	1	0	0
king	0	0	0	0	0	0	0	1	0
monarch	0	0	0	0	0	0	0	0	1

Each word gets a 1x9 vector representation

Creating word embeddings involves training a neural network on a large corpus of text data, such as news articles or books. During training, the network learns to predict the likelihood of a word appearing in a given context based on the

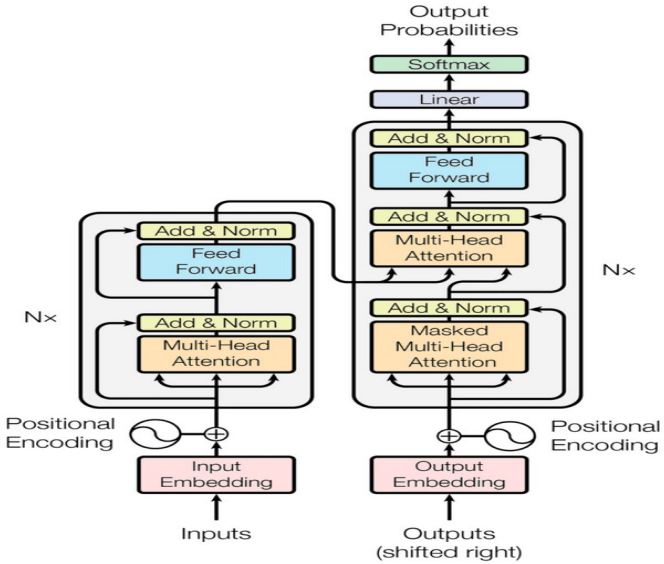
words that come before and after it in a sentence. The vectors that are learned through this process capture the semantic relationships between different words in the corpus. A similar approach is followed with the words "King," "Queen," "Man," and "Woman."

Positional Encoding

- Positional encoding is all about helping the model figure out where words are in a sequence. It doesn't deal with the meaning of words or how they relate to each other, like how "cat" and "dog" are pretty similar. Instead, positional encoding is all about keeping track of word order.
- For example, when translating a sentence like "The cat is on the mat" to another language, it's crucial to know that "cat" comes before "mat." Word order is super important for tasks like translation, summarizing stuff, and answering questions.

Transformers

- Advanced large language models utilize a certain architecture known as Transformers. Consider the transformer layer as a separate layer that comes after the traditional neural network layers.
- In fact, the transformer layer is often added as an additional layer to the traditional neural network architecture to improve the LLM's ability to model long-range dependencies in natural language text.
- The transformer layer works by processing the entire input sequence in parallel rather than sequentially. It consists of two essential components: the self-attention mechanism and the feedforward neural network.

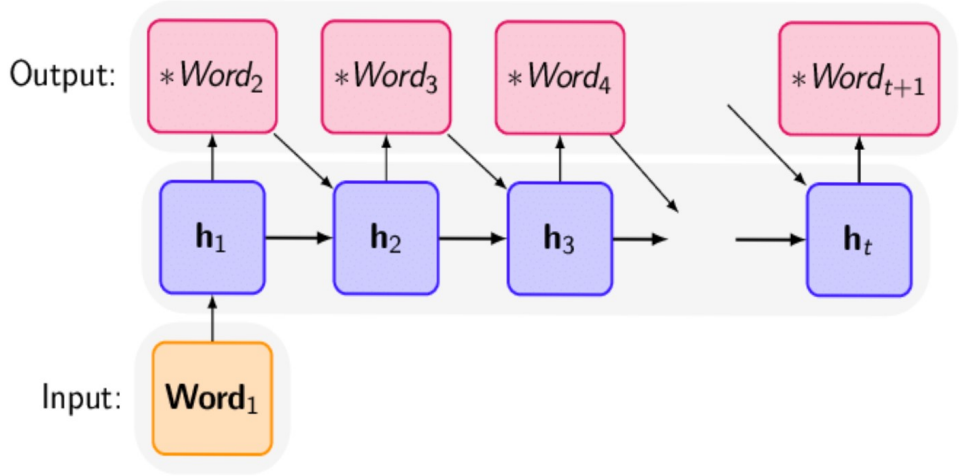


Text Generation

Often the last step performed by an LLM model; after the LLM has been trained and fine-tuned, the model can be used to generate highly sophisticated text in response to a prompt or question. The model is typically "primed" with a seed input, which can be a few words,

a sentence, or even an entire paragraph. The LLM then uses its learned patterns to generate a coherent and contextually-relevant response. Text generation relies on a technique called autoregression, where the model generates each word or token of the output sequence one at a time based on the previous words it has generated.

The model uses the parameters it has learned during training to calculate the probability distribution of the next word or token and then selects the most likely choice as the next output.



Step 1: Preparing the Dataset

Before we can train our model, we need to prepare the data in a format suitable for training. This involves the following steps:

1.1. Import the necessary libraries and read the Excel file:

```
import pandas as pd
import numpy as np

# Read the Excel file
data = pd.read_excel('your_large_excel_file.xlsx')
1.2. Clean and preprocess the data:
Remove any unnecessary columns
Fill missing values or drop rows with missing data
Convert text data to lowercase
Tokenize text and remove stop words
1.3. Split the dataset into training and validation sets:
from sklearn.model_selection import train_test_split
train_data, val_data = train_test_split(data, test_size=0.2, random_state=42)
```

2: Fine-tuning the OpenAI Model

In this step, we'll fine-tune a pre-trained OpenAI model on our dataset.

2.1. Install the OpenAI library and import necessary modules:

```
!pip install openai
import openai
import torch
from transformers import GPT2Tokenizer, GPT2LMHeadModel, TextDataset,
DataCollatorForLanguageModeling, Trainer, TrainingArguments
```

```
2.2. Load the pre-trained model and tokenizer:
MODEL_NAME = 'gpt-4'
tokenizer = GPT2Tokenizer.from_pretrained(MODEL_NAME)
model = GPT2LMHeadModel.from_pretrained(MODEL_NAME)
```

Generating Responses to Business Prompts

3.1. Define a function to generate responses:

```
def generate_response(prompt, max_length=150, num_responses=1):
    input_ids = tokenizer.encode(prompt, return_tensors='pt')
    output = model.generate(
     input_ids,
     max_length=max_length,
     num_return_sequences=num_responses,
     no_repeat_ngram_size=2,
     temperature=0.7,
     top_k=50,
     top_p=0.95,
     )
    decoded_output = [tokenizer.decode(response,
skip_special_tokens=True) for response in output]
return decoded_output
```