1. Introduction

In this tutorial, we'll examine the details of LangChain, a framework for developing applications powered by language models. We'll begin by gathering basic concepts around the language models that will help in this tutorial.

Although LangChain is primarily available in Python and JavaScript/TypeScript versions, there are options to use LangChain in Java. We'll discuss the building blocks of LangChain as a framework and then proceed to experiment with them in Java.

2. Background

Before we go deeper into why we need a framework for building applications powered by language models, it's imperative that we first understand what language models are. We'll also cover some of the typical complexities encountered when working with language models.

2.1. Large Language Models

A language model is **a probabilistic model of a natural language** that can generate probabilities of a series of words. A large language model (LLM) is a language model characterized by its large size. They're artificial neural networks with possibly billions of parameters.

An LLM is often **pre-trained on a vast amount of unlabeled data** using self-supervised and semi-supervised learning techniques. Then, the pre-trained model is adapted for specific tasks using various techniques like fine-tuning and prompt engineering:

These LLMs are capable of performing several natural language processing tasks like language translation and content summarization. They're also **capable of generative tasks like content creation**. Hence, they can be extremely valuable in applications like answering questions.

Almost all **major cloud service providers have included large language models** in their service offerings. For instance, Microsoft Azure offers LLMs like Llama 2

and OpenAI GPT-4. Amazon Bedrock offers models from AI21 Labs, Anthropic, Cohere, Meta, and Stability AI.

2.2. Prompt Engineering

LLMs are foundation models trained on a massive set of text data. Hence, they can capture the syntax and semantics inherent to human languages. However, they **must be adapted to perform specific tasks** that we want them to perform.

Prompt engineering is one of the quickest ways to adapt an LLM. It's a **process of structuring text that can be interpreted and understood by an LLM**. Here, we use natural language text to describe the task that we expect an LLM to perform:

Word Embeddings

As we've seen, LLMs are capable of processing a large volume of natural language text. The performance of LLMs vastly improves if we represent the words in natural languages as word embeddings. This is a **real-valued vector capable of encoding words' meanings**.

Typically, the word embeddings are generated using an algorithm like Tomáš Mikolov's Word2vec or Stanford University's GloVe. The **GloVe is an unsupervised learning algorithm** trained on aggregated global word-word co-occurrence statistics from a corpus:

LangChain for Java

LangChain was **launched in 2022 as an open-source project** and soon gathered momentum through community support. It was originally developed by Harrison Chase in Python and soon turned out to be one of the fastest-growing start-ups in the AI space.

There was **a JavaScript/TypeScript version of LangChain that followed the Python version** in early 2023. It soon became quite popular and started supporting multiple JavaScript environments like Node.js, web browsers, CloudFlare workers, Vercel/Next.js, Deno, and Supabase Edge functions.

Unfortunately, there is **no official Java version of LangChain** that is available for Java/Spring applications. However, **there is a community version of LangChain for Java called LangChain4j**. It works with Java 8 or higher and supports Spring Boot 2 and 3.

The various dependencies of LangChain are available at Maven Central. We may **need to add one or more dependencies** in our application, depending on the features we use: