Integration with ai

Why didn’t we use ChatGPT-3.5?

- Limited knowledge:

  if you ask ChatGPT-3.5 about a new technology that has just been released, it may not have sufficient information to answer your question accurately, because its learning sources don't include any information generated after September 2021

- Tendency to create false information:

  ChatGPT-3.5 may produce false information when asked certain questions or given certain prompts which could lead users astray if they rely too heavily on its responses without double-checking them with other sources first. ChatGPT-3.5 tends to "hallucinate" and make up information in about 15-20% of output.

- Cost:

  If each user provides five sentences (15 tokens each), a single piece of feedback would be 75 tokens long. 200 instances per day should therefore take up about 15,000 tokens. That means with the GPT-3.5 Turbo 4K model, your daily ChatGPT API cost would be about $0.0225 for input and $0.03 for output, and now paying with foreign currency has some restrictions in Egypt.

- Public LLM:

  You can’t share with it sensitive information and you can’t fine-tune information in it (Provide the model with your data or store your data in their database)

But also, it has a simple way to interact with its APIs

```
import java.io.*;
import java.net.HttpURLConnection;
import java.net.URL;

public class ChatGPTAPIExample {

   public static String chatGPT(String prompt) {
       String url = "https://api.openai.com/v1/chat/completions";
       String apiKey = "YOUR API KEY HERE";
       String model = "gpt-3.5-turbo";

       try {
           URL obj = new URL(url);
           HttpURLConnection connection = (HttpURLConnection) obj.openConnection();
           connection.setRequestMethod("POST");
           connection.setRequestProperty("Authorization", "Bearer " + apiKey);
           connection.setRequestProperty("Content-Type", "application/json");

           // The request body
           String body = "{\"model\": \"" + model + "\", \"messages\": [{\"role\": \"user\",
\"content\": \"" + prompt + "\"}]}";
           connection.setDoOutput(true);
           OutputStreamWriter writer = new OutputStreamWriter(connection.getOutputStream());
           writer.write(body);
           writer.flush();
           writer.close();

           // Response from ChatGPT
           BufferedReader br = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
           String line;

           StringBuffer response = new StringBuffer();

           while ((line = br.readLine()) != null) {
               response.append(line);
           }
           br.close();

           // calls the method to extract the message.
           return extractMessageFromJSONResponse(response.toString());

       } catch (IOException e) {
           throw new RuntimeException(e);
       }
   }

   public static String extractMessageFromJSONResponse(String response) {
       int start = response.indexOf("content")+ 11;

       int end = response.indexOf("\"", start);

       return response.substring(start, end);

   }

   public static void main(String[] args) {

       System.out.println(chatGPT("hello, how are you? Can you tell me what's a Fibonacci
Number?"));

   }}
```

But due to the disadvantages mentioned before we choose to work with a more powerful Large Language Model (LLM):

Llama2: is a family of pre-trained and fine-tuned large language models (LLMs) released by Meta AI in 2023. Released free of charge for research and commercial use, Llama 2 AI models are capable of a variety of natural language processing (NLP) tasks, from text generation to programming code, Llama 2 was trained with 2 trillion tokens—the numerically-represented words, word parts, phrases and other semantic fragments that transformer-based neural networks use for language processing—from publicly available sources.

Can be fine-tuned and private model, doesn't need internet and your data privacy is secured

Llama 2 does not have its own dedicated API, but it's accessible through multiple providers.

As we mentioned it doesn't have its own API, so we need a provider to interact with this model

Ollama: is a tool that helps us run large language models on our local machine and makes experimentation more accessible Ollama bundles model weights, configuration, and data into a single package, It optimizes setup and configuration details, including GPU usage.

First, we installed Ollama, it has no interface it works on terminal or Windows PowerShell

Then, Fetch for LLM model via ollama pull llama2

After that, you can work with it and ask her questions.

But in 5/2/2024 we switched to llama3 because:

Llama 3 has been trained on a **massive dataset** of text and code, including creative writing examples. This allows it to understand the patterns and structures of different text formats. When you provide a prompt or starting point, Llama 3 can use its knowledge to generate text that adheres to the format and style you specify.

Feature	Llama 3	Llama 2
Parameters	8 Billion & 70 Billion versions	Lower parameter versions (exact numbers may not be public)
Performance (Benchmarks)	Improved in ARC, DROP, MMLU	Likely lower scores on same benchmarks
Capabilities	Strong reasoning, creative text generation	Potentially offered similar capabilities, but likely to a lesser extent
Availability	Openly available for research	Availability details unclear, likely less open

Fine-Tuning the Model

In fine-tuning this model, we used python Language with package that take the pdf of the book convert it to text then make data manipulated to be able to fine-tune the model

But the question is: how can I interact with it using Java?

We found a package that could serve us called: Langchain4j which is

Easy interaction with LLMs and Vector Stores

All major commercial and open-source LLMs and Vector Stores are easily accessible through a unified API, enabling you to build chatbots, assistants, and more.

Tailored for Java

Smooth integration into your Java applications is made possible thanks to Quarkus and Spring Boot integrations. There is two-way integration between LLMs and Java: you can call LLMs from Java and allow LLMs to call your Java code in return.

Then from its document, we learned to interact with it

https://docs.langchain4j.dev/get-started/

it's not that easy because lack of videos or tutorials about AI with Java especially integration with langchain4j