Criando um Projeto de IA Generativa em Java

Guia Completo

A Inteligência Artificial Generativa representa um dos avanços mais impactantes da computação moderna.

Seu uso abrange desde assistentes virtuais até sistemas de geração de conteúdo automatizado.

Neste guia, desenvolveremos um chatbot baseado na API da OpenAI utilizando **Java 17+**, **Maven** e a biblioteca openai-gpt3-java.

O projeto incluirá um sistema de cache para otimização de chamadas à API e um modelo alternativo para flexibilizar a geração de respostas.

1. Configuração do Ambiente

Antes de iniciar o desenvolvimento, configure seu ambiente com os seguintes requisitos:

1. **Instale o Java JDK 17+** para garantir compatibilidade com bibliotecas recentes.
2. **Configure o Maven** para gerenciar dependências e compilações do projeto.
3. **Escolha uma IDE** como IntelliJ IDEA, Eclipse ou NetBeans para um desenvolvimento eficiente.
4. **Obtenha uma chave de API da OpenAI** para autenticação nas chamadas à API.

2. Criando o Projeto Maven

1. **Crie um novo projeto Maven** na sua IDE.
2. **Adicione as seguintes dependências** no arquivo pom.xml:

```
<dependencies>
    <dependency>
        <groupId>com.theokanning.openai-gpt3-java</groupId>
        <artifactId>service</artifactId>
        <version>0.18.2</version>
    </dependency>
```

```
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>1.18.30</version>
        <scope>provided</scope>
    </dependency>
</dependencies>
```

3. Estrutura do Projeto

Organizamos o código em pacotes para facilitar a manutenção e escalabilidade:

```
src/main/java/com/example/genai/
├── config/
│   └── OpenAIConfig.java
├── service/
│   ├── AIService.java
│   ├── TextSummarizer.java
│   ├── AlternativeAIService.java
│   └── CacheService.java
└── Main.java
```

3. Implementação Passo a Passo

4.1. Configuração do OpenAI

```
package com.example.genai.config;

import com.theokanning.openai.service.OpenAiService;

public class OpenAIConfig {
    private static final String API_KEY = "sua-chave-api-aqui";

    public static OpenAiService getOpenAiService() {
        return new OpenAiService(API_KEY);
    }
}
```

4.2. Serviço de IA com Cache

```
package com.example.genai.service;

import com.theokanning.openai.completion.chat.ChatCompletionRequest;
import com.theokanning.openai.completion.chat.ChatMessage;
import com.theokanning.openai.service.OpenAiService;
import java.util.ArrayList;
import java.util.List;

public class AIService {
    private final OpenAiService service;
    private final CacheService cacheService;

    public AIService(OpenAiService service, CacheService cacheService) {
        this.service = service;
```

```
        this.cacheService = cacheService;
    }

    public String generateResponse(String prompt) {
        if (cacheService.hasCachedResponse(prompt)) {
            return cacheService.getCachedResponse(prompt);
        }

        List<ChatMessage> messages = new ArrayList<>();
        messages.add(new ChatMessage("user", prompt));

        ChatCompletionRequest request = ChatCompletionRequest.builder()
                .model("gpt-3.5-turbo")
                .messages(messages)
                .maxTokens(500)
                .temperature(0.7)
                .build();

        String response = service.createChatCompletion(request)
                .getChoices().get(0)
                .getMessage().getContent();

        cacheService.storeResponse(prompt, response);
        return response;
    }
}
```

4.3. Serviço de Cache

```
package com.example.genai.service;

import java.util.HashMap;
import java.util.Map;

public class CacheService {
    private final Map<String, String> cache = new HashMap<>();

    public boolean hasCachedResponse(String prompt) {
        return cache.containsKey(prompt);
    }

    public String getCachedResponse(String prompt) {
        return cache.get(prompt);
    }

    public void storeResponse(String prompt, String response) {
        cache.put(prompt, response);
    }
}
```

4.4. Classe Principal

```
package com.example.genai;

import com.example.genai.config.OpenAIConfig;
import com.example.genai.service.AIService;
import com.example.genai.service.CacheService;
import java.util.Scanner;
```

```
public class Main {
    public static void main(String[] args) {
        CacheService cacheService = new CacheService();
        AIService aiService = new AIService(OpenAIConfig.getOpenAiService(), cacheService);
        Scanner scanner = new Scanner(System.in);

        System.out.println("Bem-vindo ao Assistente IA!");
        System.out.println("Digite 'sair' para encerrar o programa.");

        while (true) {
            System.out.print("\nSua pergunta: ");
            String input = scanner.nextLine();

            if (input.equalsIgnoreCase("sair")) {
                break;
            }

            try {
                String response = aiService.generateResponse(input);
                System.out.println("\nResposta da IA: " + response);
            } catch (Exception e) {
                System.out.println("Erro ao processar sua solicitação: " + e.getMessage());
            }
        }

        scanner.close();
        System.out.println("Programa encerrado!");
    }
}
```

5. Exemplo de Uso com Entrada e Saída (Input e Output)

Entrada:

Bem-vindo ao Assistente IA!
Digite 'sair' para encerrar o programa.

Sua pergunta: Como funciona a IA Generativa?

Saída:

Resposta da IA: A IA generativa funciona através de modelos treinados que utilizam aprendizado profundo para gerar texto, imagens e outros tipos de conteúdo baseados em padrões aprendidos.

Conclusão Técnica

Este projeto demonstra como integrar a API da OpenAI em um sistema Java eficiente e escalável.

- **Implementação de um sistema de cache**, reduzindo chamadas redundantes e otimizando custos e desempenho.
- **Criação de um serviço alternativo**, permitindo maior flexibilidade e controle sobre as respostas geradas.
- **Estruturação modular do código**, separando funcionalidades para melhor manutenção e expansão.
- **Gerenciamento eficiente de chamadas API**, garantindo escalabilidade e tempo de resposta otimizado.

Bons estudos

EducaCiência FastCode para a comunidade