Spring Boot + vLLM + Chroma DB (PDF RAG System)

> **Note:** This guide shows how to create a Spring Boot application that integrates vLLM for LLM inference and Chroma DB for vector storage, with PDF document ingestion.

Prerequisites

- Java 17+
- Python 3.9+ (for vLLM)
- Docker (for Chroma DB)
- GPU recommended (for vLLM)
- Maven

1. Project Setup

1.1 Create a new Spring Boot project

Use start.spring.io or your IDE to create a new project with these dependencies:

- Spring Web
- Lombok (optional)

1.2 pom.xml

Add these dependencies to your `pom.xml`:

```
 <?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.o
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.0</version>
    </parent>
    <groupId>com.example</groupId>
    <artifactId>vllm-chroma-pdf</artifactId>
    <version>0.0.1-SNAPSHOT</version>
```

```
    <properties>
        <java.version>17</java.version>
    </properties>

    <dependencies>
        <!-- Spring AI -->
        <dependency>
            <groupId>org.springframework.ai</groupId>
            <artifactId>spring-ai-openai-spring-boot-starter</artifactId>
            <version>0.8.0</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.ai</groupId>
            <artifactId>spring-ai-chroma-store-spring-boot-starter</artifactId>
            <version>0.8.0</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.ai</groupId>
            <artifactId>spring-ai-pdf-document-reader</artifactId>
            <version>0.8.0</version>
        </dependency>

        <!-- Spring Web -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <!-- Lombok (optional) -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2. Infrastructure Setup

2.1 Start Chroma DB (Vector Store)


```
docker run -d -p 8000:8000 chromadb/chroma
```

2.2 Start vLLM Server

```
 pip install vllm
python -m vllm.entrypoints.openai.api_server \
  --model mistralai/Mistral-7B-Instruct-v0.1 \
  --port 8001
```

> **Note:** You can replace the model with any HuggingFace model. For CPU-only machines, consider using a quantized model like `TheBloke/Mistral-7B-Instruct-v0.1-GGUF` with LocalAI instead.

3. Application Configuration

3.1 application.yml

```
spring:
  ai:
    openai:
      base-url: http://localhost:8001/v1  # vLLM OpenAI-compatible API
      api-key: dummy-key  # vLLM doesn't need a real key
      chat:
        model: mistralai/Mistral-7B-Instruct-v0.1  # Your vLLM model

    chroma:
      client:
        host: localhost
        port: 8000  # Chroma DB port

    vectorstore:
      chroma:
        collection-name: pdf-documents
        embedding-dimensions: 1536
```

4. PDF Loading Implementation

4.1 PDF Loader Service

```
package com.example.service;

import org.springframework.ai.document.Document;
import org.springframework.ai.reader.ExtractedTextFormatter;
import org.springframework.ai.reader.pdf.PagePdfDocumentReader;
import org.springframework.ai.reader.pdf.config.PdfDocumentReaderConfig;
import org.springframework.ai.transformer.splitter.TokenTextSplitter;
import org.springframework.ai.vectorstore.VectorStore;
import org.springframework.core.io.Resource;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class PdfLoaderService {

    private final VectorStore vectorStore;
    private final TokenTextSplitter textSplitter;

    public PdfLoaderService(VectorStore vectorStore) {
        this.vectorStore = vectorStore;
        this.textSplitter = new TokenTextSplitter();
    }

    public void loadPdf(Resource pdfResource) {
        try {
            // Configure PDF reader
            var pdfReader = new PagePdfDocumentReader(
                pdfResource,
                PdfDocumentReaderConfig.builder()
                    .withPageExtractedTextFormatter(new ExtractedTextFormatter.Builde
                        .withNumberOfBottomTextLinesToDelete(3)  // Remove footer
                        .withNumberOfTopTextLinesToDelete(1)    // Remove header
                        .build())
                    .build()
            );

            // Split into chunks
            List<Document> documents = textSplitter.apply(pdfReader.get());

            if (documents.isEmpty()) {
                throw new RuntimeException("No text extracted from PDF");
            }

            // Store in Chroma DB
            vectorStore.add(documents);

            System.out.println("PDF loaded successfully. Chunks: " + documents.size()
        } catch (Exception e) {
```

```
            throw new RuntimeException("Failed to load PDF: " + e.getMessage(), e);
        }
    }
}
```


4.2 Data Loader (Initial PDF Loading)

```
package com.example.config;

import com.example.service.PdfLoaderService;
import org.springframework.boot.CommandLineRunner;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;

@Configuration
public class DataLoaderConfig {

    @Bean
    public CommandLineRunner initialPdfLoader(PdfLoaderService pdfLoaderService) {
        return args -> {
            // Load a PDF from classpath (src/main/resources)
            pdfLoaderService.loadPdf(new ClassPathResource("sample.pdf"));

            // Or from a URL:
            // pdfLoaderService.loadPdf(new UrlResource("https://example.com/document
        };
    }
}
```

5. RAG Controller

```
package com.example.controller;

import org.springframework.ai.chat.ChatClient;
import org.springframework.ai.vectorstore.VectorStore;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;
import java.util.stream.Collectors;

@RestController
public class RagController {
```

```
    private final ChatClient chatClient;
    private final VectorStore vectorStore;

    public RagController(ChatClient chatClient, VectorStore vectorStore) {
        this.chatClient = chatClient;
        this.vectorStore = vectorStore;
    }

    @GetMapping("/ask")
    public String ask(@RequestParam String question) {
        // 1. Retrieve relevant documents from Chroma DB
        List<String> relevantDocs = vectorStore.similaritySearch(question)
                .stream()
                .map(doc -> doc.getContent())
                .collect(Collectors.toList());

        if (relevantDocs.isEmpty()) {
            return "No relevant information found";
        }

        // 2. Build RAG prompt
        String context = String.join("\n\n", relevantDocs);
        String prompt = """
            Answer the question based only on the following context:
            ---
            Context:
            %s
            ---
            Question: %s
            """.formatted(context, question);

        // 3. Get AI response from vLLM
        return chatClient.call(prompt);
    }
}
```

6. Running the Application

6.1 Start the services

1. Start Chroma DB: `docker run -d -p 8000:8000 chromadb/chroma`
2. Start vLLM: `python -m vllm.entrypoints.openai.api_server --model mistralai/Mistral-7B-Instruct-v0.1 --port 8001`

6.2 Run Spring Boot

```
mvn spring-boot:run
```


6.3 Test the API

```
curl "http://localhost:8080/ask?question=What%20is%20the%20main%20topic%20of%20the%2(
```

7. Project Structure

```
src/main/
├── java/
│   └── com/example/
│       ├── config/          # Configuration classes
│       │   └── DataLoaderConfig.java
│       ├── controller/      # REST controllers
│       │   └── RagController.java
│       ├── service/         # Business logic
│       │   └── PdfLoaderService.java
│       └── VllmChromaPdfApplication.java  # Main class
└── resources/
    ├── application.yml      # Configuration
    └── sample.pdf          # Example PDF to load
```

> **Tip:** For production use, consider:
>
> - Adding authentication
> - Implementing rate limiting
> - Adding logging and monitoring
> - Using a proper PDF preprocessing pipeline

1. What is Chroma?

Definition

Chroma is an **open-source vector database** specifically designed for AI applications. It stores embeddings (vector representations of data) and enables efficient similarity searches.

Key Features

- Lightweight and easy to deploy (Docker or standalone)
- Optimized for vector similarity search
- Supports filtering by metadata
- Persistent storage option

Usage in This Project

1. **Storage:** Stores vector embeddings of PDF document chunks
2. **Retrieval:** Finds relevant document sections using cosine similarity
3. **Setup:** Runs in Docker container on port 8000

Configuration Example:

```
spring:
  ai:
    chroma:
      client:
        host: localhost
        port: 8000
      collection-name: pdf-documents
```

2. What is vLLM?

Definition

vLLM is a **high-performance LLM inference engine** developed by UC Berkeley. It provides OpenAI-compatible API endpoints for local LLM execution.

Key Features

- PagedAttention for optimized memory usage
- Continuous batching for high throughput
- Supports most HuggingFace models
- 10-24x faster than naive inference

Usage in This Project

1. **Model Hosting:** Serves Mistral-7B via OpenAI-compatible API
2. **Prompt Processing:** Generates answers using RAG context
3. **Setup:** Requires GPU for optimal performance

Startup Command:

```
python -m vllm.entrypoints.openai.api_server \
  --model mistralai/Mistral-7B-Instruct-v0.1 \
  --port 8001 \
  --max-model-len 4096
```

3. What is spring-ai-pdf-document-reader?

Definition

A Spring AI module that provides **PDF parsing capabilities** using Apache PDFBox under the hood.

Key Features

- Extracts text page-by-page
- Handles headers/footers removal
- Preserves document structure
- Works with encrypted PDFs

Usage in This Project

1. **Loading:** Reads PDFs from filesystem/URL
2. **Preprocessing:** Removes headers/footers
3. **Output:** Produces Document objects for vectorization

Code Example:

```
PagePdfDocumentReader reader = new PagePdfDocumentReader(
  pdfResource,
  PdfDocumentReaderConfig.builder()
    .withPageExtractedTextFormatter(new ExtractedTextFormatter.Builder()
      .withNumberOfTopTextLinesToDelete(1)
      .build())
```

```
    .build()
);
```


4. OpenAI Spring Boot Starter

What is spring-ai-openai-spring-boot-starter? Definition

A Spring Boot starter that provides **auto-configuration** for OpenAI-compatible clients, including vLLM.

Key Features

- Auto-wires `ChatClient` and `EmbeddingClient`
- Supports both OpenAI API and local endpoints (like vLLM)
- Manages API keys and connection pooling

Usage in This Project

1. **vLLM Integration:** Connects to local vLLM server
2. **Prompt Execution:** Handles RAG prompt submissions
3. **Configuration:** Simplified via `application.yml`

Configuration Example:

```
spring:
  ai:
    openai:
      base-url: http://localhost:8001/v1  # vLLM endpoint
      api-key: dummy-key
      chat:
        model: mistralai/Mistral-7B-Instruct-v0.1
```

5. Chroma Store Starter

What is spring-ai-chroma-store-spring-boot-starter? Definition

A Spring Boot starter that **auto-configures Chroma DB** as a `VectorStore` implementation.

Key Features

- Automatically creates collections
- Handles connection pooling
- Supports metadata filtering

Usage in This Project

1. **Storage:** Auto-wires `VectorStore` bean
2. **Operations:** Simplified `add()` and `similaritySearch()`
3. **Metadata:** Optional document metadata storage

Code Example:

```
@Autowired
private VectorStore vectorStore;

// Store documents
vectorStore.add(List.of(new Document("text")));

// Retrieve similar documents
List<Document> results = vectorStore.similaritySearch("query");
```

6. VectorStore Interface

What is org.springframework.ai.vectorstore.VectorStore? Definition

A Spring AI **abstraction layer** for vector database operations.

Key Features

- Unified API for different vector databases
- Supports CRUD operations with embeddings
- Metadata-aware searching

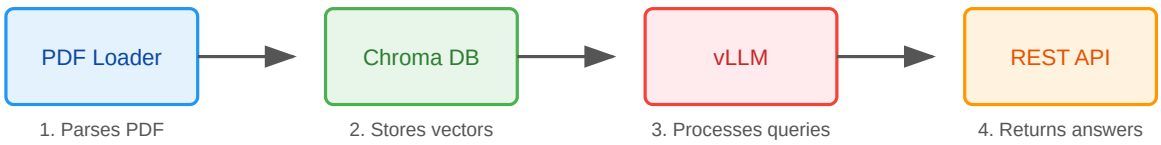
Usage in This Project

1. **Abstraction:** Works with Chroma DB seamlessly
2. **RAG Pipeline:** Handles document storage/retrieval
3. **Swappable:** Can switch to PostgreSQL/Pinecone later

Interface Methods:

```
public interface VectorStore {
  void add(List<Document> documents);
  List<Document> similaritySearch(String query);
  List<Document> similaritySearch(
      String query,
      int k,
      double threshold);
}
```

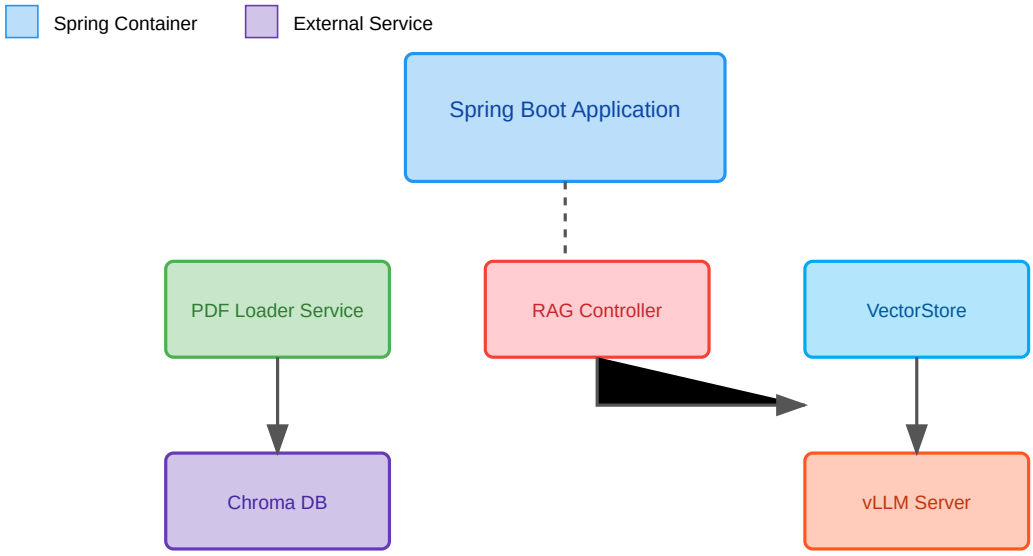
Functional Architecture



Data Flow

1. **PDF Loader:** Extracts text chunks from PDF files
2. **Chroma DB:** Stores text embeddings as vectors
3. **vLLM:** Generates answers using RAG pattern
4. **REST API:** Exposes query endpoint to clients

Technical Architecture



Component Interactions

Component	Responsibility
PDF Loader Service	Parses PDFs into text chunks with metadata
VectorStore	Manages embeddings in Chroma DB
RAG Controller	Handles HTTP requests and orchestrates RAG flow

Troubleshooting

Issue	Solution
Chroma DB not responding	`docker ps` Check if container is running

Issue	Solution
vLLM server unreachable	Verify: • Port 8001 is available • GPU drivers are properly installed • Model files are downloaded
No text extracted from PDF	⚠️ Use OCR tools for scanned documents
Slow responses	Try quantized models like: `TheBloke/Mistral-7B-Instruct-v0.1-GGUF`