



ICT Training Center



Il tuo partner per la Formazione e la Trasformazione digitale della tua azienda





SPRING AI

GENERATIVE ARTIFICIAL INTELLIGENCE CON JAVA

Simone Scannapieco

Corso avanzato per Venis S.p.A, Venezia, Italia

Novembre 2025



RETRIEVAL AUGMENTED GENERATION

PARTE 1

- ➡ Chatbot Ollama-CV e Gemini-Venis
- ⚠ Approccio *naive* (informazione testuale estratta tramite GENAI-assisted web scraping/summarization)
- 1 Iscrizione al portale HuggingFace e creazione access token
- 2 Verifica e modifica variabili di ambiente
- 3 Creazione configurazione ambiente Docker Qdrant
- 4 Modifica dipendenze di progetto
- 5 Modifica configurazione applicativo
- 6 Creazione *prompt templates* per strategia RAG
- 7 Configurazione vector store per Ollama e Gemini embeddings
- 8 Creazione component per popolamento vector store (dati Venis e CV)
- 9 Creazione interfaccia e implementazione del servizio
- 10 Modifica controllore MVC
- 11 Test delle funzionalità con Postman/Insomnia

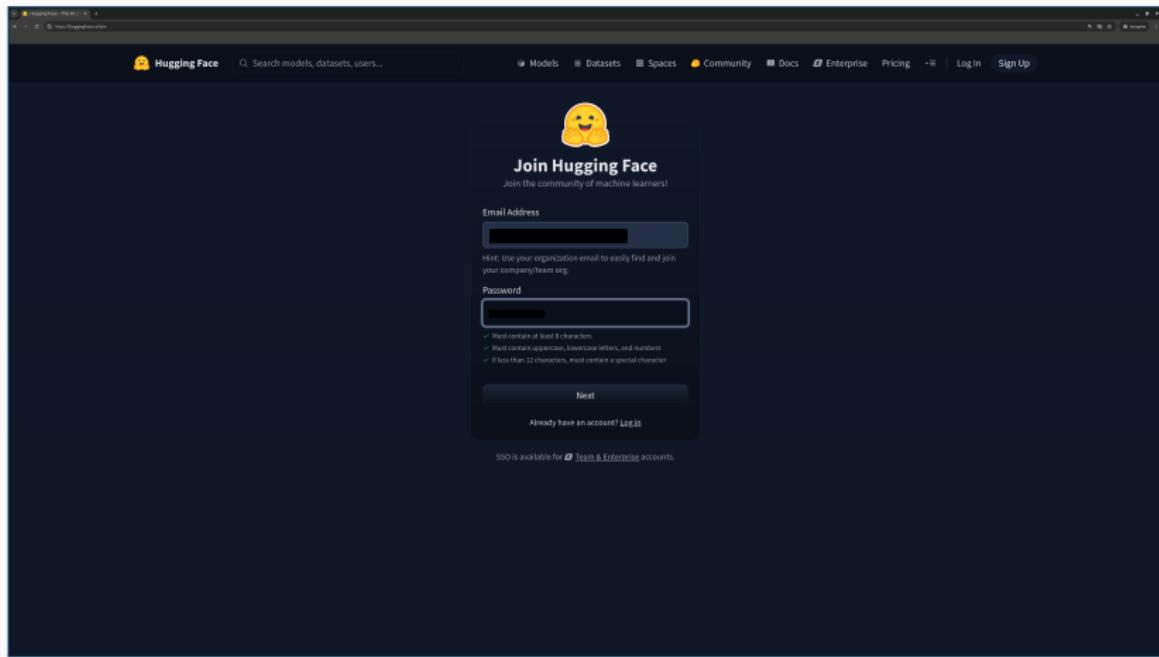
AMBIENTE DI SVILUPPO

CREAZIONE ACCOUNT HUGGINGFACE

- 1 Accedere al portale <https://huggingface.co/> e cliccare il pulsante Sign up

The screenshot shows the Hugging Face homepage. At the top right, there is a red-bordered "Sign Up" button. The main content area features a large emoji of a smiling face with hands raised. Below it, the text "The AI community building the future." is displayed. A sidebar on the left lists various AI model categories like NLP, Computer Vision, and Audio. The right side shows a grid of trending models, each with a thumbnail, name, and a brief description. At the bottom, there's a "Trending on 🤖 this week" section.

2 Creare una nuova utenza



AMBIENTE DI SVILUPPO

CREAZIONE ACCOUNT HUGGINGFACE

3 Accedere al portale e selezionare Access Tokens nel menu in alto a destra

The screenshot shows the Hugging Face platform interface. On the left, there's a sidebar with options like 'Profile', 'Inbox (0)', 'Settings', 'Billing', and 'Get PRO'. Below that are sections for 'Organizations' and 'Resources'. The main area shows a 'Following' list with items from 'essential' (They might open source something soon), 'yogito' (Pioneering in AI video generation technology), and 'Qwen' (Developing advanced large language and vision models). On the right, there's a 'Profile' section for 'Simone.Scannapieco' and a 'Trending' section with various AI projects. A prominent red box highlights the 'Access Tokens' option in the top right corner of the main menu bar.

AMBIENTE DI SVILUPPO

CREAZIONE ACCOUNT HUGGINGFACE

- 4 Premere sul pulsante Create new token

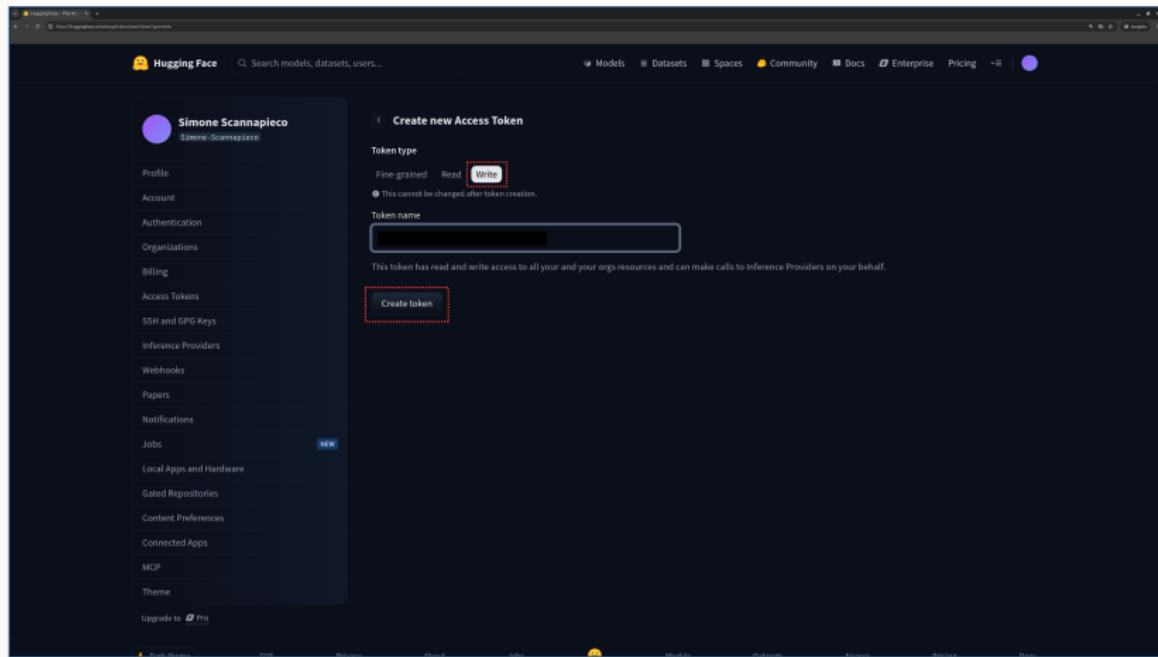
The screenshot shows the Hugging Face Hub interface. On the left, there's a sidebar with various account settings like Profile, Account, Authentication, Organizations, Billing, and several sections under Access Tokens. The 'Access Tokens' section is currently selected. On the right, there's a table titled 'User Access Tokens' with columns for Name, Value, Last Refreshed Date, Last Used Date, and Permissions. A red dashed box highlights the 'Create new token' button at the top right of this table area. The overall theme is dark.

Name	Value	Last Refreshed Date	Last Used Date	Permissions
0	0	0	0	WRITE
0	0	0	0	READ
0	0	0	0	READ
0	0	0	0	READ
0	0	0	0	READ

AMBIENTE DI SVILUPPO

CREAZIONE ACCOUNT HUGGINGFACE

- 5 Selezionare token di tipo Write, denominare il token e premere Create token



AMBIENTE DI SVILUPPO

CREAZIONE ACCOUNT HUGGINGFACE

6 Cercare il modello di *embedding*, leggendone la card

The screenshot shows the Hugging Face platform interface. At the top, there is a search bar with the placeholder "Simone Scannapieco". Below the search bar, the results for "Simone Scannapieco" are displayed. The first result is "Simone-Scannapieco/sentence-bert-base-italian-v2", which is described as "See 1 model results for 'Simone Scannapieco'".

The main content area features a large image of a smiling emoji. Below it, the text "The AI community building the future." is displayed in a large, bold font. A subtitle below reads: "The platform where the machine learning community collaborates on models, datasets, and applications." There are two buttons at the bottom left: "Explore AI Apps" and "Browse 1M+ models".

On the right side of the screen, there is a sidebar with various categories: Models, Datasets, Spaces, Community, Docs, Enterprise, Pricing, Log In, and Sign Up. Below these, there is a "Trending on 🤖 this week" section.

The main search results area has tabs for Libraries, Datasets, Languages, Licenses, and Other. It lists several models:

- meta-llama/Llama-2-7B
- stabilityai/stable-diffusion-xl-base-0.9
- openchat/openchat
- llyasizel/Greyscale-v1.1
- carlsberg/zenosocene_v2_2L
- meta-llama/Llama-2-13B
- flirtei/falcon-4B-11bit-v1
- MirroredVisionCoCo-15B-V2.0
- ComfyUI/Tableau-LLMfusion-v1.4
- stabilityai/ChatGLM-6B-v0.3
- balefire/space-70-8k-v1-test

AMBIENTE DI SVILUPPO

CREAZIONE ACCOUNT HUGGINGFACE

6 Cercare il modello di *embedding*, leggendone la card

The screenshot shows the Hugging Face Model Card interface for the model `sentence-bert-base-italian-xxl-uncased-F32-GGUF`. The card includes the following details:

- Model Card**: Shows the model's name, last updated (16 days ago), and its use of the GGUF binary format.
- Hardware compatibility**: Supports 32-bit hardware.
- Inference Providers**: Lists supported providers like Sentence Similarity.
- Base model**: The model is based on the `sickipedia/sentence-bert-base-italian-xxl-uncased` model.
- Dataset used to train**: The dataset used for training is `PhilipMay/stsb_wmt11_it`.

File launch.json

```
{  
    // Use IntelliSense to learn about possible attributes.  
    // Hover to view descriptions of existing attributes.  
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387  
    "version": "0.2.0",  
    "configurations": [  
        {  
            "type": "java",  
            "name": "Launch Current File",  
            "request": "launch",  
            "mainClass": "${file}"  
        },  
        {  
            "type": "java",  
            "name": "DemoApplication",  
            "request": "launch",  
            "mainClass": "it.venis.ai.spring.demo.DemoApplication",  
            "projectName": "demo",  
            "env": {  
                "GOOGLE_AI_API_KEY": "...",  
                "HUGGING_FACE_HUB_TOKEN": "..."  
            }  
        }  

```

File settings.json

```
{  
    "java.compile.nullAnalysis.mode": "disabled",  
    "java.configuration.updateBuildConfiguration": "interactive",  
    "java.test.config": {  
        "env": {  
            "GOOGLE_AI_API_KEY": "...",  
            "HUGGING_FACE_HUB_TOKEN": "..."  
        }  
    }  
}
```

File docker-compose.yml

```
services:  
  ...  
  spring-ai-vector-store:  
    image: qdrant/qdrant:${QDRANT_VERSION:-latest}  
    hostname: spring-ai-vector-store  
    container_name: spring_ai_vector_store  
    ports:  
      - ${QDRANT_HTTP_PORT:-6333}:6333  
      - ${QDRANT_GRPC_PORT:-6334}:6334  
    volumes:  
      - spring_ai_vector_store:/qdrant/storage  
    restart: unless-stopped  
  
volumes:  
  ...  
  spring_ai_vector_store:  
    name: spring_ai_vector_store
```

File spring-ai.env

```
...  
# qdrant configuration  
QDRANT_VERSION=v1.13.0  
QDRANT_HTTP_PORT=6333  
QDRANT_GRPC_PORT=6334  
  
# default: latest  
# default: 6333  
# default: 6334
```

Dipendenze di sistema aggiuntive

```
...
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-rag</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-advisors-vector-store</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.ai</groupId>
    <artifactId>spring-ai-starter-vector-store-qdrant</artifactId>
</dependency>
...
...
```

Configurazione applicativo

```
spring:
  ...
  autoconfigure:
    exclude:
      - org.springframework.ai.vectorstore.qdrant.autoconfigure.QdrantVectorStoreAutoConfiguration
        # We must disable Vector Store auto-configuration because of two different EmbeddingModel beans
        # (OpenAI-Gemini and Ollama). The goal is to have two different vector collections, one for each
        # family of LLM.
  ai:
    ollama:
      ...
      embedding:
        model: hf.co/Simone-Scannapieco/sentence-bert-base-italian-xxl-uncased-F32-GGUF
  openai:
    ...
    embedding:
      options:
        model: gemini-embedding-001
  vectorstore:
    qdrant:
      initialize-schema: true
      host: 172.17.0.1
      port: 6334
  ...
```

File erase_llm_volumes.sh

```
#!/bin/bash

volume_name=spring_ai_llm

docker volume rm $volume_name
```

File erase_vector_store_volumes.sh

```
#!/bin/bash

volume_name=spring_ai_vector_store

docker volume rm $volume_name
```

File get-rag-data-system-ita-prompt.st

Sei un assistente AI in grado di rispondere alle domande dell'utente solo in base al contesto fornito dalla sezione DOCUMENTI.

Se la risposta non è presente nella sezione DOCUMENTI, informa l'utente di non sapere la risposta.

DOCUMENTI: <documenti>

File get-rag-data-system-eng-prompt.st

You are a helpful assistant, answering questions based on the given context in the DOCUMENTS section and no prior knowledge.

If the answer is not in the DOCUMENTS section, then reply that you cannot answer to the question.

DOCUMENTS: <documenti>

Configurazione Vector Store - I

```
package it.venis.ai.spring.demo.config;

import org.springframework.ai.embedding.EmbeddingModel;
import org.springframework.ai.openai.OpenAiEmbeddingModel;
import org.springframework.ai.vectorstore.VectorStore;
import org.springframework.ai.vectorstore.qdrant.QdrantVectorStore;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import io.qdrant.client.QdrantClient;
import io.qdrant.client.QdrantGrpcClient;

@Configuration
public class RAGConfig {

    @Autowired
    public OpenAiEmbeddingModel openAiEmbeddingModel;

    @Autowired
    public EmbeddingModel ollamaEmbeddingModel;

    @Value("${spring.ai.vectorstore.qdrant.host:#{null}}")
    private String qdrantHost;
    @Value("${spring.ai.vectorstore.qdrant.port:#{null}}")
    private String qdrantPort;
    @Value("${spring.ai.vectorstore.qdrant.use-tls:#{null}}")
    private String useTls;

    ...
}
```

Configurazione Vector Store - II

```
...  
@Bean  
public QdrantClient qdrantClient() {  
    QdrantGrpcClient.Builder grpcClientBuilder = QdrantGrpcClient.newBuilder()  
        .qdrantHost == null ? "localhost" : qdrantHost,  
        .qdrantPort == null ? 6334 : Integer.valueOf(qdrantPort),  
        .useTls == null ? false : Boolean.valueOf(useTls);  
    return new QdrantClient(grpcClientBuilder.build());  
}  
  
@Value("${spring.ai.vectorstore.qdrant.collection-name.gemini:#{null}}")  
private String qdrantCollectionNameGemini;  
@Value("${spring.ai.vectorstore.qdrant.collection-name.ollama:#{null}}")  
private String qdrantCollectionNameOllama;  
@Value("${spring.ai.vectorstore.qdrant.initialize-schema:#{null}}")  
private String qdrantInitializeSchema;  
  
@Bean  
public VectorStore geminiVectorStore(QdrantClient qdrantClient,  
    @Qualifier("openAiEmbeddingModel") EmbeddingModel geminiEmbeddingModel) {  
    return QdrantVectorStore.builder(qdrantClient, geminiEmbeddingModel)  
        .collectionName(qdrantCollectionNameGemini == null ? "vector_store_gemini" : qdrantCollectionNameGemini)  
        .initializeSchema(qdrantInitializeSchema == null ? false : Boolean.valueOf(qdrantInitializeSchema))  
        .build();  
}  
  
@Bean  
public VectorStore ollamaVectorStore(QdrantClient qdrantClient,  
    @Qualifier("ollamaEmbeddingModel") EmbeddingModel ollamaEmbeddingModel) {  
    return QdrantVectorStore.builder(qdrantClient, ollamaEmbeddingModel)  
        .collectionName(qdrantCollectionNameOllama == null ? "vector_store_ollama" : qdrantCollectionNameOllama)  
        .initializeSchema(qdrantInitializeSchema == null ? false : Boolean.valueOf(qdrantInitializeSchema))  
        .build();  
}
```

Componente popolamento Qdrant - I

```
package it.venis.ai.spring.demo.rag;

...
@Component
public class TextDataLoader {

    private final VectorStore geminiVectorStore;
    private final VectorStore ollamaVectorStore;

    public TextDataLoader(@Qualifier("geminiVectorStore") VectorStore geminiVectorStore,
        @Qualifier("ollamaVectorStore") VectorStore ollamaVectorStore) {
        this.geminiVectorStore = geminiVectorStore;
        this.ollamaVectorStore = ollamaVectorStore;
    }

    @PostConstruct
    public void loadVenisInfoIntoVectorStore() {
        List<String> venisInfo = List.of(...);
        SearchRequest searchRequest = SearchRequest.builder()
            .query("Check")
            .similarityThresholdAll()
            .build();
        List<Document> similarDocs = geminiVectorStore.similaritySearch(searchRequest);
        if (similarDocs.size() == 0) {
            List<Document> documents =
                venisInfo.stream().map(Document::new).collect(Collectors.toList());
            this.geminiVectorStore.add(documents);
        }
    }
}
```

Componente popolamento Qdrant - II

```
...
@PostConstruct
public void loadSSCVInfoIntoVectorStore() {
    List<String> ssCVInfo = List.of(...);
    SearchRequest searchRequest = SearchRequest.builder()
        .query("Check")
        .similarityThresholdAll()
        .build();
    List<Document> similarDocs = ollamaVectorStore.similaritySearch(searchRequest);
    if (similarDocs.size() == 0) {
        List<Document> documents = ssCVInfo.stream().map(Document::new).collect(Collectors.toList());
        this.ollamaVectorStore.add(documents);
    }
}
```

Interfaccia servizio

```
package it.venis.ai.spring.demo.services;

import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.QuestionRequest;

public interface RAGService {

    public Answer getGeminiRAGTextToVectorStoreAnswer(QuestionRequest request);

    public Answer getOllamaRAGTextToVectorStoreAnswer(QuestionRequest request);

}
```

Implementazione servizio - I

```
package it.venis.ai.spring.demo.services;  
  
...  
  
@Service  
@Configuration  
public class RAGServiceImpl implements RAGService {  
  
    private final ChatClient geminiChatClient;  
    private final ChatClient ollamaChatClient;  
    private final ChatClient ollamaMemoryChatClient;  
    private VectorStore geminiVectorStore;  
    private VectorStore ollamaVectorStore;  
  
    public RAGServiceImpl(  
        @Qualifier("geminiChatClient") ChatClient geminiChatClient,  
        @Qualifier("ollamaChatClient") ChatClient ollamaChatClient,  
        @Qualifier("ollamaMemoryChatClient") ChatClient ollamaMemoryChatClient,  
        @Qualifier("geminiVectorStore") VectorStore geminiVectorStore,  
        @Qualifier("ollamaVectorStore") VectorStore ollamaVectorStore) {  
  
        this.geminiChatClient = geminiChatClient;  
        this.ollamaChatClient = ollamaChatClient;  
        this.ollamaMemoryChatClient = ollamaMemoryChatClient;  
        this.geminiVectorStore = geminiVectorStore;  
        this.ollamaVectorStore = ollamaVectorStore;  
  
    }  
  
    ...
```

Implementazione servizio - II

```
...
@Value("classpath:templates/get-rag-data-system-eng-prompt.st")
private Resource ragDataSystemEngPrompt;

@Override
public Answer getGeminiRAGTextToVectorStoreAnswer(QuestionRequest request) {

    SearchRequest searchRequest = SearchRequest.builder()
        .query(request.body().question())
        .topK(4)
        .similarityThreshold(.2)
        .build();

    List<Document> similarDocs = geminiVectorStore.similaritySearch(searchRequest);

    String similarDocsString = similarDocs.stream()
        .map(Document::getText)
        .collect(Collectors.joining(System.lineSeparator()));

    return new Answer(this.geminiChatClient.prompt()
        .system(s -> s.text(this.ragDataSystemEngPrompt)
            .params(Map.of("documenti", similarDocsString)))
        .user(request.body().question())
        .templateRenderer(StTemplateRenderer.builder().startDelimiterToken('<')
            .endDelimiterToken('>')
            .build())
        .call()
        .content());
}

...

```

Implementazione servizio - III

```
...
@Value("classpath:templates/get-rag-data-system-ita-prompt.st")
private Resource ragDataSystemItaPrompt;

@Override
public Answer getOllamaRAGTextToVectorStoreAnswer(QuestionRequest request) {

    SearchRequest searchRequest = SearchRequest.builder()
        .query(request.body().question())
        .topK(4)
        .similarityThreshold(.3)
        .build();

    List<Document> similarDocs = ollamaVectorStore.similaritySearch(searchRequest);

    String similarDocsString = similarDocs.stream()
        .map(Document::getText)
        .collect(Collectors.joining(System.lineSeparator()));

    return new Answer(this.ollamaMemoryChatClient.prompt()
        .advisors(advisorSpec -> advisorSpec.param(ChatMemory.CONVERSATION_ID, request.username()))
        .system(s -> s.text(this.ragDataSystemItaPrompt)
            .params(Map.of("documenti", similarDocsString)))
        .user(request.body().question())
        .templateRenderer(StTemplateRenderer.builder().startDelimiterToken('<')
            .endDelimiterToken('>')
            .build())
        .call()
        .content());
}

}
```

Implementazione controllore REST

```
package it.venis.ai.spring.demo.controllers;  
  
...  
  
@RestController  
public class QuestionController {  
  
    private final QuestionService service;  
    private final RAGService ragService;  
  
    public QuestionController(QuestionService service, RAGService ragService) {  
        this.service = service;  
        this.ragService = ragService;  
    }  
  
    ...  
  
    @PostMapping("/gemini/ask/rag/text-to-vs/venis")  
    public Answer getGeminiRAGTextToVectorStoreAnswer(@RequestBody QuestionRequest request) {  
        return this.ragService.getGeminiRAGTextToVectorStoreAnswer(request);  
    }  
  
    @PostMapping("/ollama/ask/rag/text-to-vs/cv")  
    public Answer getOllamaRAGTextToVectorStoreAnswer(@RequestBody QuestionRequest request) {  
        return this.ragService.getOllamaRAGTextToVectorStoreAnswer(request);  
    }  
}
```



⚠ Verificare la dashboard Qdrant (<http://172.17.0.1:6333/dashboard>)

The screenshot shows the Qdrant dashboard interface. On the left, there's a sidebar with icons for collections, embeddings, and other features. The main area is titled "Collections". It contains a search bar labeled "Search Collection". Below the search bar is a table with the following data:

Name	Status	Points (Approx)	Segments	Shards	Vectors Configuration (Name, Size, Distance)	Actions
vector_store_gemini	green	18	8	1	default 3072 Cosine	⋮
vector_store_oilama	green	17	8	1	default 768 Cosine	⋮

At the bottom left of the dashboard, it says "v1.19.0".



<https://github.com/simonescannapieco/spring-ai-advanced-dgroove-venis-code.git>

Branch: 7-spring-ai-gemini-ollama-rag-text-to-vector-store