Lecture 18: Embedding Using Spring AI

Definition

Embeddings are vector representations of text that capture meaning and similarity. With Spring AI, embeddings can be generated directly from a Java application.

Implementation Steps:

- **Controller Setup**
  - Create a RestController.
  - Define @PostMapping("/api/embedding").
  - Accept input text using @RequestParam String text.

```
@PostMapping("/api/embedding")
public float[] embedding(@RequestParam String text) {
    return embeddingModel.embed(text);
}
```

- **Embedding Model**
  - Use the EmbeddingModel provided by Spring AI.
  - Inject it with @Autowired.
  - If multiple embeddings exist (OpenAI, Ollama), use @Qualifier("openAiEmbeddingModel").

```
@Autowired
@Qualifier("openAiEmbeddingModel")
private EmbeddingModel embeddingModel;
```

- Application Properties:Set API key and model
  - spring.application.name=SpringAICode
  - spring.ai.openai.api-key=<your-openai-api-key>
  - spring.ai.openai.embedding.options.model=text-embedding-3-large

Code Implementation:

```java
@RestController
public class OpenAIController {

    private ChatClient chatClient;

    @Autowired
    private EmbeddingModel embeddingModel;

    public OpenAIController(OpenAiChatModel chatModel) {
        this.chatClient = ChatClient.create(chatModel);
    }

    @PostMapping("/api/embedding")
    public float[] embedding(@RequestParam String text) {
        return embeddingModel.embed(text);
    }
}
```

Working:

- When a request is sent to /api/embedding?text=Bottle, Spring AI converts the text into embeddings.
- Returns an array of float values (vector representation).
- The model can be configured (e.g., text-embedding-3-large)

Key Takeaways:

- Qualifier resolves conflicts if multiple embedded beans exist.
- Embeddings are high-dimensional vectors (floats).
- Used for similarity search, clustering, and semantic understanding.