Easy RAG with LangChain4J

Julien Dubois
@juliendubois
Principal Manager, Java Developer Relations

What is LangChain4J?

- A Java version of LangChain, a JavaScript + Python framework
- Aims to simplify the creation of applications using Large Language Models
- Fully Open Source

Why would you use LangChain4J?

- A layer of abstraction above many APIs
- Easy integration of those APIs to build more complex applications
- Nice tooling to get work done faster

Demo code

The application used in this demo is available here:

https://github.com/jdubois/jdubois-langchain4j-demo

RAG = Retrieval-Augmented Generation

- 2 main limits with LLMs
  - They have a cut-off date
  - They don’t have your company internal data
- How do we typically provide this data to a LLM?
  - Get new/private data and transform them into vectors
  - Store those into a vector database
  - Request the vector database and send that data to the LLM
- This is where LangChain4J excels
  - It allows to easily use vector databases and LLMs
  - It gives you access to advanced features like Hybrid Search and Semantic re-ranking (with Azure AI Search)

Let’s do this!

Ingestion (1/3) – getting the data

- Document Loader
  - Fetch the data from a URL, Blob storage, GitHub repository…
- Document Transformer
  - Transforms the document to get only relevant data
- Document Splitter
  - Splits the document into segments
  - Segments can be paragraphs, lines, sentences, words… Or “recursive” to get the best one
  - Maximum segment size
  - Maximum overlap size
  - https://langchain-text-splitter.streamlit.app

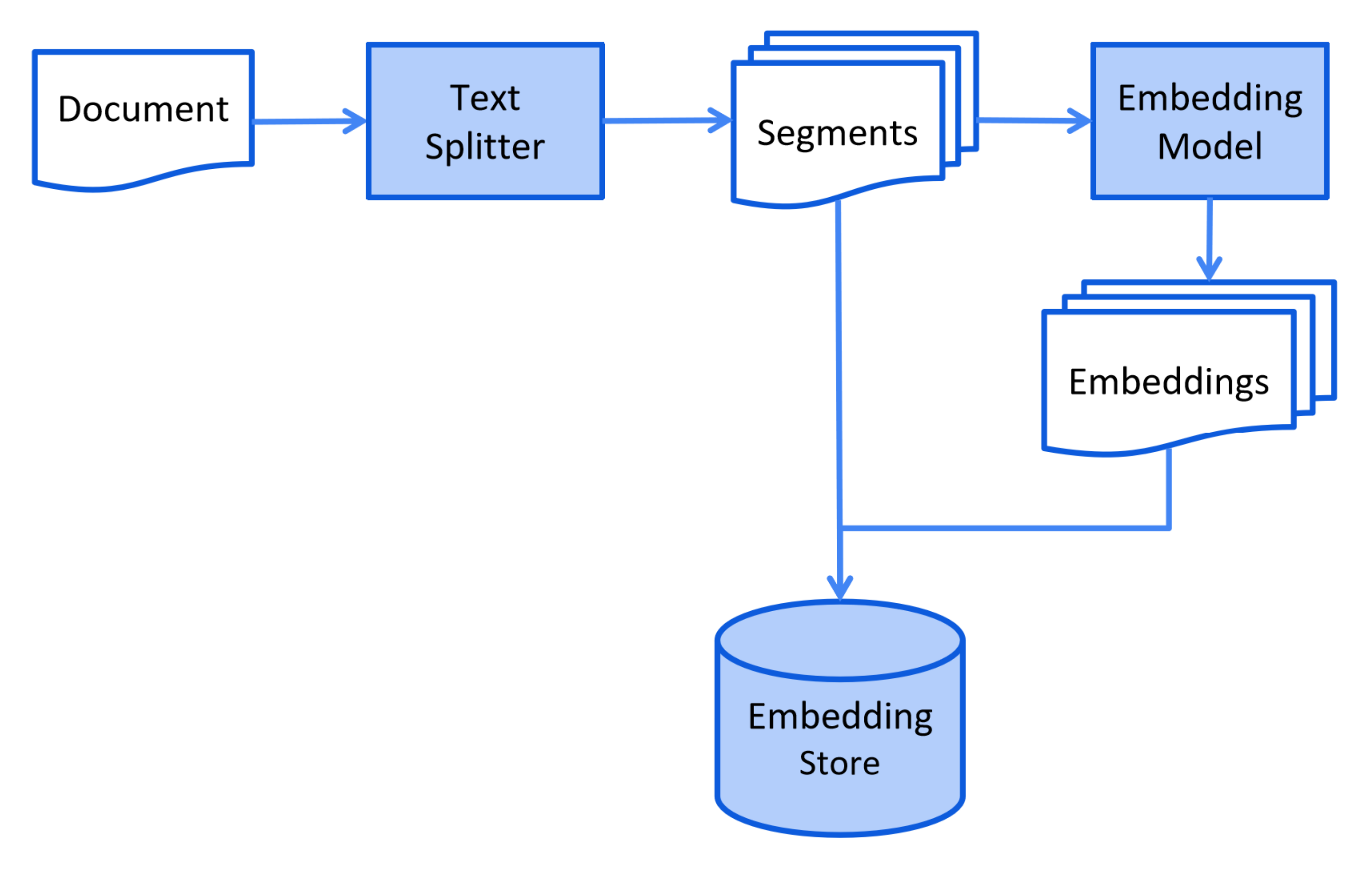
Ingestion (2/3) – transforming the text into a vector

- Embedding Model
    - Transforms the segment into a vector: an array of floats in the form [-0.021544598,-0.0643939,-0.08963279,-0.002898384,0.07627905,-0.02540857,-0.031227088,-0.0407423,-0.000047008543,-0.02396834]
    - This vector has a dimension: the size of the array (the biggest = the better & the slowest)
- There are many embedding models: "text-embedding-ada”, "text-embedding-3-small”, "nomic-embed-text"

Ingestion (3/3) – storing the vectors

- Embedding Store
  - Usually a vector store
  - More complex/advanced stores exist, for example adding full text search (=Lucene), and being able to mix those search mechanisms for finding better segments
- Stores all the vectors, as well as metadata
  - Metadata is very important: it should include the segment text, and usually references like file names (so sources can be quoted in the retrieval phase)
- Can find quickly vectors that point in the same direction, usually using an algorithm called "cosine similarity"

Graphical representation



Source: LangChain4J documentation at https://docs.langchain4j.dev/tutorials/rag/

Putting it all together with Easy RAG

```java
Document document =
    UrlDocumentLoader.load("https://www.microsoft.com/investor/reports/ar23/index.html",
    new TextDocumentParser());

EmbeddingStoreIngestor ingestor = EmbeddingStoreIngestor.builder()
        .documentTransformer(new HtmlToTextDocumentTransformer(".annual-report"))
        .documentSplitter(DocumentSplitters.recursive(300, 30))
        .embeddingModel(embeddingModel)
        .embeddingStore(embeddingStore)
        .build();

ingestor.ingest(document);
```

Best practices

- Clean up the text
- Test with different splitters, segment size, overlap size
- Test with different embedding models and different dimensions
- Use metadata, at least for quoting sources
- Be ready to re-index everything!
  - Have a smaller set of data for testing
  - Keep all the original data available
- Plan to index new data regularly

Retrieval-augmented generation (1/2)

- Get the user’s question
- Transform it into an embedding
  - This should use the same model and dimensions as when we stored the documents
- Query the vector store to find which vectors are similar to the user’s question
  - There are more advanced solutions, to help find better vectors
  - Getting too many vectors will lead to worse generation: the goal is to have the best vectors, not many vectors

Retrieval-augmented generation (2/2)

- Augment the user’s question using the text segments
- Query the LLM with the augmented prompt
  - Many Chat Models are available
  - This calculation benefits more from having a GPU
- Some interesting LLMs to test with
  - GPT-4o and GPT-4o-mini : latest and best models from OpenAI
  - Phi-3.5 : small and smart model from Microsoft
  - Llama 3.2: very small model from Meta (warning: cannot be used in Europe)
  - Tinyllama: incredibly small model compatible with Llama, good for testing on a laptop

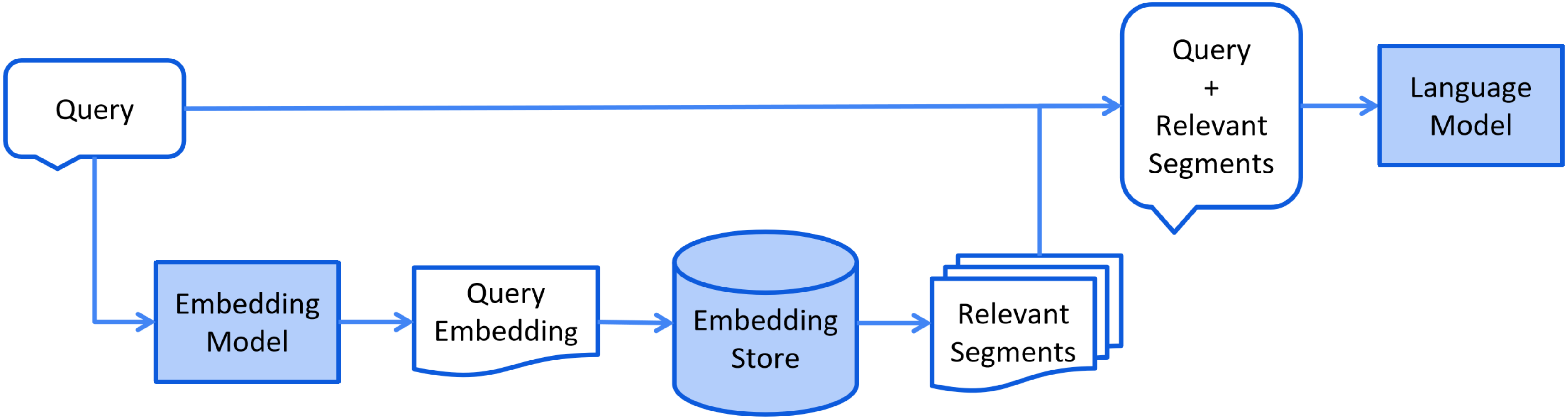
Default prompt used by EasyRAG

```
{{user question}}

Answer using the following information:
{{text segment 1}}
{{text segment 2}}
…
{{text segment n}}
```

- Possible improvements
  - Given the following information and not prior knowledge, answer the question above.
  - Keep your answer grounded in the facts of the information below.
  - If the information below doesn’t contain the facts to answer, answer that you don’t know.

Graphical representation



Source: LangChain4J documentation at https://docs.langchain4j.dev/tutorials/rag/

Putting it all together with Easy RAG

```
String question = "How many people are employed by Microsoft in the US?";

Assistant assistant = AiServices.builder(Assistant.class)
        .chatLanguageModel(chatLanguageModel)
        .contentRetriever(
            new EmbeddingStoreContentRetriever(embeddingStore, embeddingModel, 3))
        .build();

String answer = assistant.chat(question);
```

Best practices

- Do not send too much data in the prompt
    - This can lead to very bad answers
    - It costs tokens (=money)
- Test the Chat Models
    - Depending on the task, some models are better than others
    - Bigger models cost a lot more, so use them only if they are useful
- Improve the prompt
    - The prompt’s efficiency depends on the model
- Add integration tests, for example with TestContainers
- Use GPUs in production

Putting it all together

- EasyRAG makes it easy to use the RAG pattern
  - Sensible defaults
  - Tooling
  - Easy to extend
- You’ll have your RAG-based application available in no time
  - Then you’ll need to improve the ingestion and test various LLMs
  - This will cost you time & money, as AI isn’t replacing your hard work (yet)

Do you want the full experience?

This does not use EasyRAG on purpose: it’s a complete 3-hour workshop to build everything the hard way.
https://aka.ms/ws/openai-rag-quarkus

Microsoft 💙 developers