LLM Integration in Python and Java: A Developer's Journey with LangChain and Spring AI

Niccolò Caselli

I. Introduction

The advent of large language models has revolutionized natural language processing tasks. Frameworks like LangChain and Spring AI have emerged to streamline the incorporation of LLMs into applications. Understanding their differences is crucial for developers aiming to leverage LLMs effectively.

II. LangChain Overview

LangChain is an open-source framework developed by Harrison Chase in October 2022, designed to simplify the integration of large language models (LLMs) into diverse application domains. The framework offers a flexible and modular architecture, empowering developers to build complex AI-driven products. LangChain has grown into a comprehensive ecosystem with complementary tools such as LangGraph and LangSmith, which extend its capabilities for building sophisticated, production-ready applications and experimental projects. LangChain is maintained by an active community of contributors and is widely adopted across startups and big-tech enterprises.

A. Key Features

- **Modular Components:** Built around core abstractions such as chains, tools, agents, memory, and document loaders, enabling flexible, reusable, and composable AI pipelines.
- **Extensive Integrations:** Provides seamless integration with a variety of cloud providers, vector stores (e.g., FAISS, Pinecone, Weaviate), language model APIs (e.g., OpenAI, Anthropic, Cohere, Hugging Face), and file loaders for working with structured and unstructured data.
- **Agents and Planning:** Supports advanced agent architectures such as `ReAct`, `MRKL`, `Plan-and-Execute`, and `BabyAGI`, allowing for out-of-the-box decision-making and task decomposition.
- **LangGraph Integration:** `LangGraph` is an low-level orchestration framework for complex agentic systems. It enables the construction of stateful, multi-step LLM applications using a directed graph model.
- **LangSmith Integration:** `LangSmith` is a unified observability and evaluation platform for developers. It provides tools to trace execution, benchmark performance, and analyze system behavior.
- **Multi-language Support:** While LangChain is primarily written in Python, it also supports JavaScript and Java implementations.

III. Spring AI Overview

Spring AI is an open-source initiative within the Spring Framework that facilitates the integration of generative AI capabilities into enterprise-level Java applications. As LangChain, Spring AI is open-source, allowing developers to contribute actively to the project.

Although Spring AI was officially introduced in August 2023 during the *SpringOne* conference, the first milestone release, version 1.0.0-M1, was made available on May 30, 2024. The stable General Availability release of Spring AI version 1.0.0 is anticipated between late April and early May 2025.

A. Key Features

- **Spring Ecosystem Integration:** Seamlessly integrates with the existing Spring Framework, enabling developers to leverage AI capabilities within the Spring ecosystem.
- **Enterprise Scalability:** Designed to support enterprise-level applications, enabling the development of AI-powered solutions that scale effectively across diverse environments. Spring provides robust scalability through its multi-threading capabilities and efficient use of the JVM, making Java a strong choice for handling complex, concurrent workloads.
- **Seamless Deployment:** The Spring ecosystem provides a comprehensive framework that facilitates the efficient deployment of AI models across heterogeneous environments. Components such as Spring Boot and Spring Cloud support the scalable orchestration of microservices, making Spring particularly well-suited for production-grade, distributed AI systems.

IV. Comparative Analysis

Among the differences highlighted in Table I, the most notable likely lies in the conceptual approach to supporting agents. Spring AI and LangChain employ fundamentally different strategies. LangChain follows an agent-first approach and natively supports patterns like ReAct, MRKL, and Plan-and-Execute, and integrates a wide range of tools such as web search (allowing retrieval using the entire internet), databases, external APIs, and even external applications. Conversely, Spring AI takes a more workflow-oriented approach, prioritizing reliability. Following the insights of Anthropic, its emphasis on simplicity and composability over complex frameworks. At the time of writing, Spring AI does not directly integrate tools for creating agents but instead offers documentation of **agentic patterns** (e.g., Chain Workflow, Parallelization

TABLE I
FEATURE COMPARISON BETWEEN LANGCHAIN AND SPRING AI

Feature	LangChain	Spring AI
Language Support	Python (JavaScript/TypeScript secondary)	Java/Kotlin
Agents	Native framework support	Not supported
Design Approach	Modular chains/tools/memory components	Spring-style abstractions (templates, dependency injection)
Model Providers	50+ integrations	15+ supported providers
State Management	Conversation memory classes + LangGraph for stateful workflows	Chat memory support
Observability	LangSmith integration	Builds upon the observability features in the Spring ecosystem
Deployment	Docker/K8s optional	Spring Boot/Cloud-native ready
REST API Support	Requires Flask/FastAPI add-ons	Native Spring Web support
Primary Focus	Rapid prototyping/research/startups	Enterprise backend integration

Workflow, Routing Work, and Orchestrator-Workers) that can be implemented via the framework interfaces. Therefore, for now, there is no autonomous decision loop like LangChain's AgentExecutor or a native mechanism for automatic routing or re-planning (like Plan-and-Execute). But this is likely to change as the project develops.

V. DEVELOPMENT EXPERIMENT: IELTS ESSAY EVALUATOR WITH RAG

To explore the development experience with LangChain and Spring AI, I implemented an IELTS essay evaluation system using Retrieval-Augmented Generation (RAG). This experiment focuses on understanding how each framework supports developer workflow and analysing the integration challenges.

A. Dataset and Methodology

I used a publicly available dataset from Kaggle containing 1200 IELTS essays from Task 1 and Task 2, each annotated with band scores and evaluation criteria. The RAG pipeline was constructed to retrieve relevant evaluation examples from the dataset, and then pass the enriched context to a language model for evaluation. A similarity threshold of 0.7 was employed for the retrieval of the top 5 documents.

Two systems were developed:

- **Python-based RAG with LangChain:** Utilized `FAISS` for vector search, OpenAI GPT-4 as the foundation model, and LangChain to orchestrate the retrieval and generation pipeline, as well as manage embedding creation. A REST API was provided using `FastAPI`.
- **Java-based RAG with Spring AI:** Employed Spring AI for retrieval and embedding support, with OpenAI GPT-4 as the foundation model. Used Spring's `SimpleVectorStore` as an in-memory vector database.

B. Development Insights

TABLE II
DEVELOPMENT EXPERIENCE: LANGCHAIN VS SPRING AI

Aspect	LangChain (Python)	Spring AI (Java)
Ease of Setup	High	Moderate
Modularity	High	High
Integration with Backend	Low	High
Documentation	Excellent	Growing
Community Support	Large and active	Emerging, backed by Spring ecosystem

A key observation concerns the documentation and community support for the two frameworks. While LangChain offers excellent documentation, it can be somewhat scattered and difficult to navigate, especially due to frequent updates that often render older guides and examples obsolete. However, this is compensated by an active community that produces high-quality tutorials and articles. Spring AI, on the other hand, benefits from the well-established and structured nature of the broader Spring ecosystem. Although its documentation is still maturing and online resources are currently limited, its integration with Spring Initializr and the more focused scope of its API can make it easier for newcomers to get started.

Another relevant factor is Java's verbosity relative to Python, which can hinder development speed in tasks like string manipulation and prompt construction. In this context, Spring AI is at a disadvantage: operations that are concise in Python often require boilerplate in Java. This extends to tasks such as JSON parsing, where Python's dynamic data structures, like dictionaries, facilitate simpler data access and manipulation. Spring AI addresses these challenges by offering higher-level abstractions, including classes like `PromptTemplate` and predefined prompt structures, which reduce manual string handling and streamline prompt creation. Moreover, Java's strong typing, robust IDE support and mature build tools, such as Maven and Gradle, enhance maintainability and reliability, particularly in large-scale systems.

VI. CONCLUSION

LangChain and Spring AI represent two distinct approaches to LLM integration: one prioritizing flexibility and cutting-edge agentic capabilities, the other emphasizing enterprise-grade reliability and alignment with established engineering practices. My hands-on implementation revealed that:

- LangChain delivers a fast and flexible developer experience, suitable not only for rapid prototyping and experimentation but also for building advanced systems. However, its expansive and evolving ecosystem can lead to a more dispersed development experience.

- Spring AI fits naturally within the Java ecosystem, offering strong support for modularity and production-readiness. While it currently provides fewer features, this results in a more straightforward integration process.

These findings highlight that framework selection should be guided by project priorities — whether the focus is on innovation and agentic capabilities or on stability and maintainability in a production context.

VII. Project Availability and Disclaimer

The code for this project is publicly available at: https://github.com/NiccoloCase/ielts-evaluator-langchain-springai.

Disclaimer: This project is intended solely as a development testbed for experimenting with the development lifecycle with LangChain and Spring AI in the context of Retrieval-Augmented Generation (RAG). It is not production-ready. Several improvements are necessary to move towards production-quality product, including:

1) **Prompt Engineering:**
   - Reduce the number of in-context examples (currently five) to lower API usage costs.
   - Standardize the format of examples to align with the desired JSON output structure. This is currently complicated by the dataset's lack of per-category evaluation scores.
2) **Hyperparameter Tuning:** Optimize the temperature setting to balance between creativity and consistency in output. Different temperature values can be used for evaluation (which should tend to be deterministic), while human-like feedback can be used for more subjective assessment.
3) **Structured Output:** Utilize the structured output features of both Spring AI and LangChain to ensure that model responses strictly conform to the predefined JSON schema.
4) **Deployment Considerations:** Replace the in-memory vector store used in prototypes with a production-ready, deployed vector database to enable scalable retrieval-augmented generation.