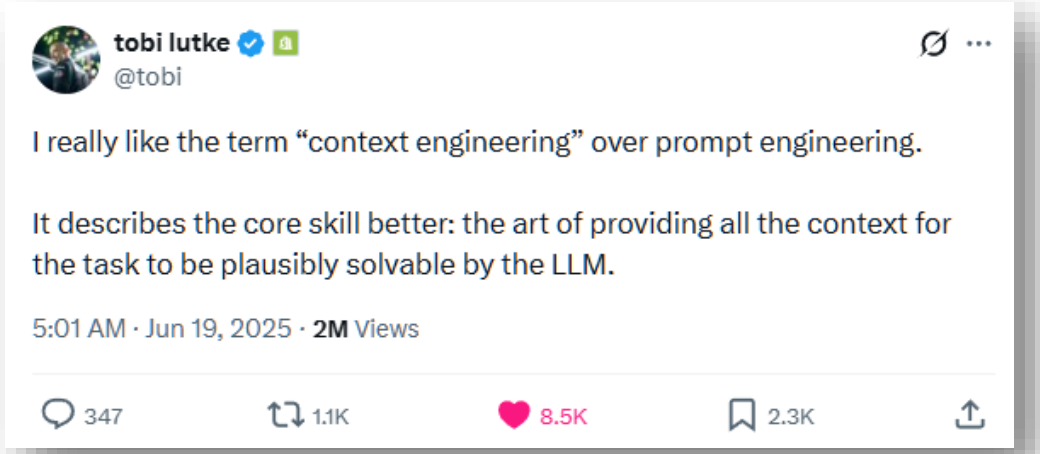
Context Engineering with Spring AI

yCrash Webinar
September 9, 2026

Boni García
https://bonigarcia.dev/

Entering Context Engineering

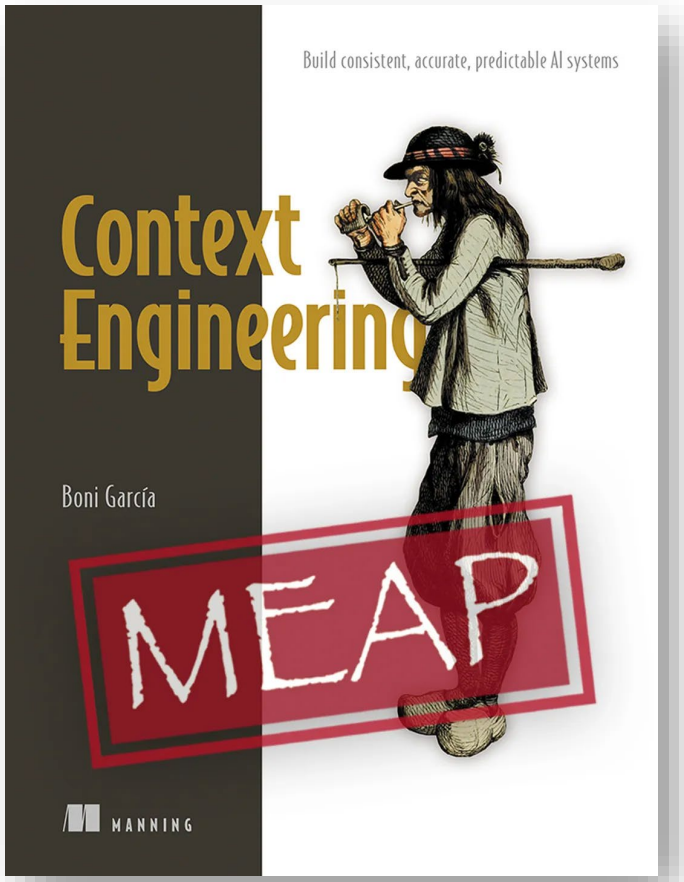


https://x.com/tobi/status/1935533422589399127

https://x.com/karpathy/status/1937902205765607626

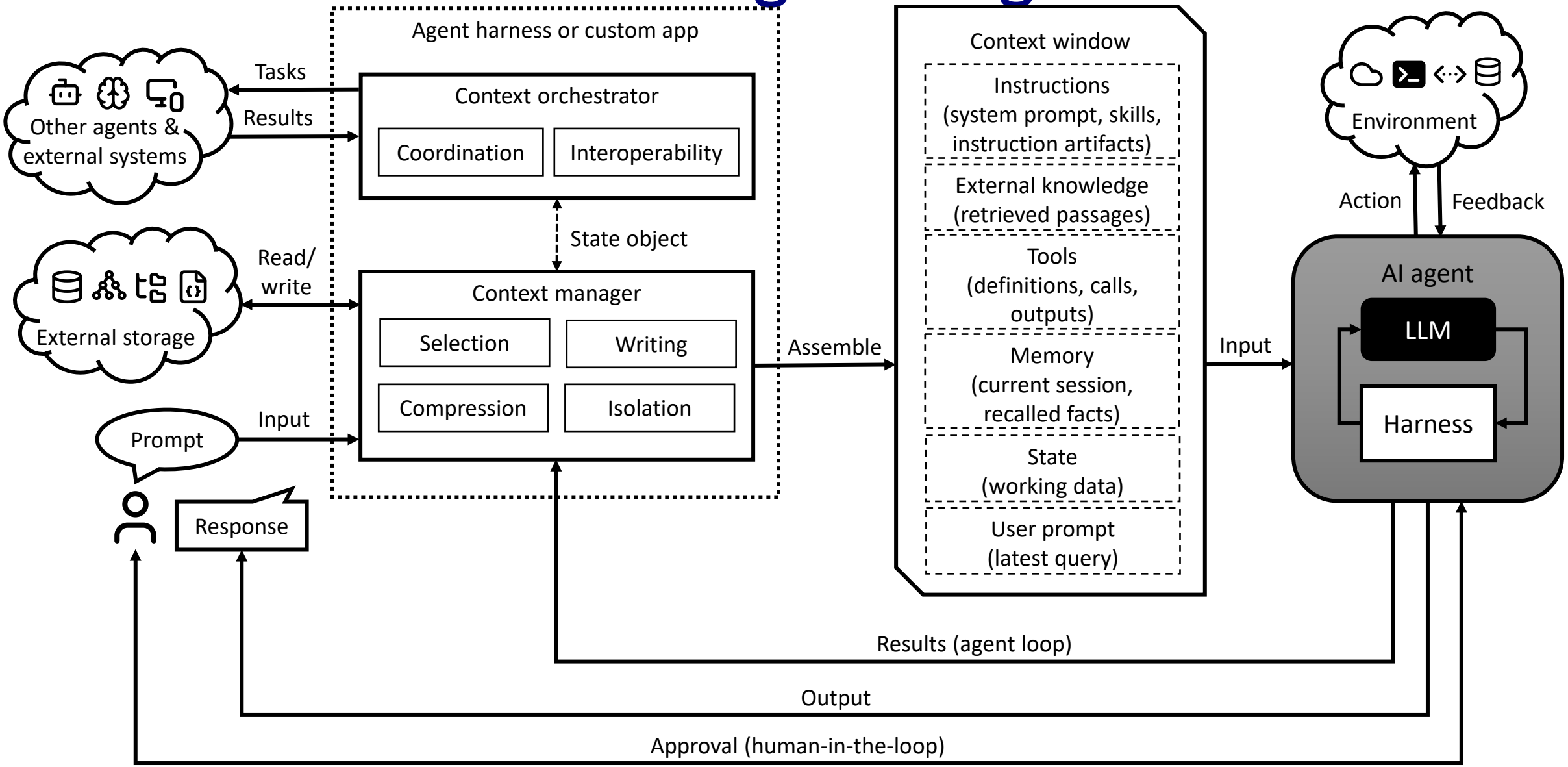
What is Context Engineering?

> ***Context engineering*** *is the discipline of assembling, maintaining, and routing the information an LLM-based system supplies to each model call*

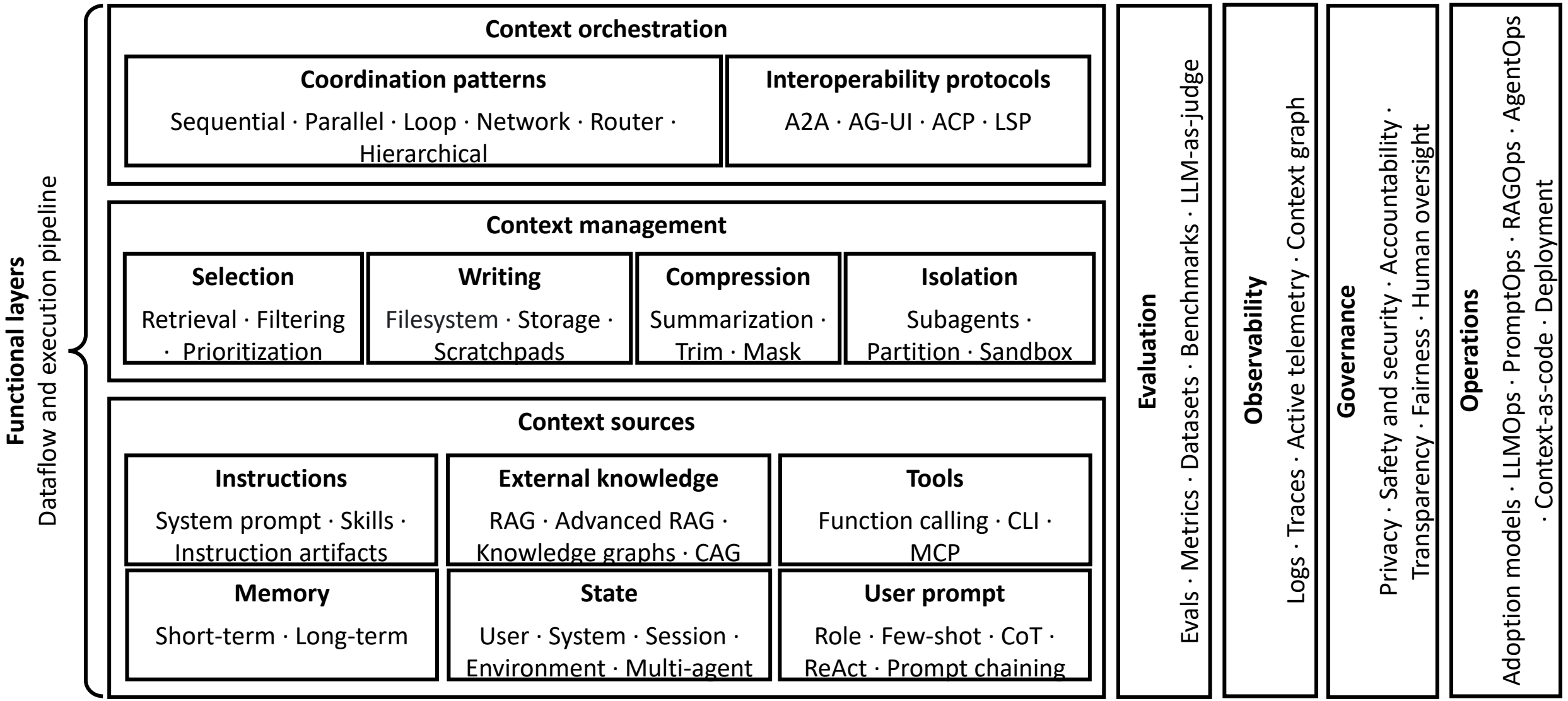


https://www.manning.com/books/context-engineering

End-to-End Context Engineering



The Context Engineering Stack



Frameworks and Platforms for Context Engineering

- AI application frameworks


LangChain

LlamaIndex

haystack by deepset


Spring AI

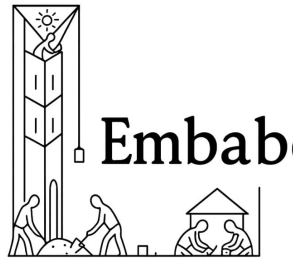
LangChain4j

Pydantic AI

DSPy
AI SDK

- Agent orchestration frameworks


LangGraph
crewai

Agent Development Kit

Embabel


Microsoft Agent Framework
Agno

parlant

deepagents

Claude Agent SDK

- AI application platforms

Dify

Langflow

Flowise
zapier


n8n

OpenClaw

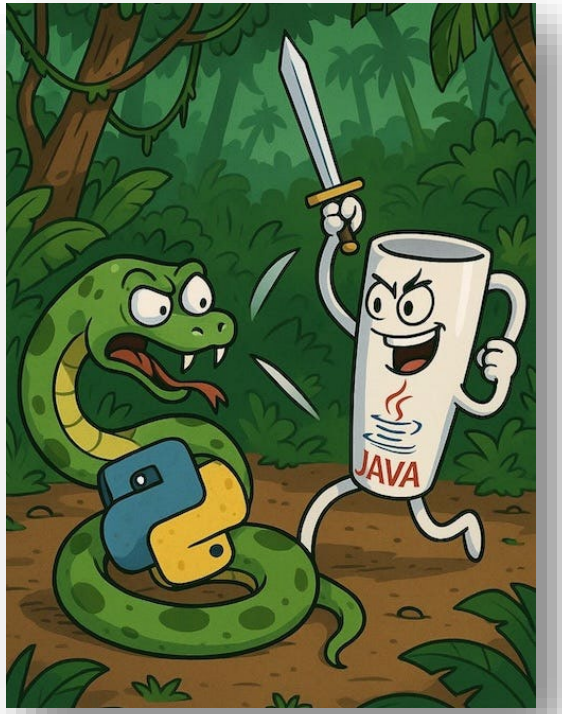
Temporal

Amazon Bedrock

Gemini Enterprise

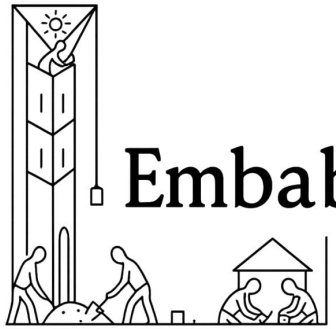
Why Java?

- Enterprise integration
- Type safety and reliability
- Rich ecosystem

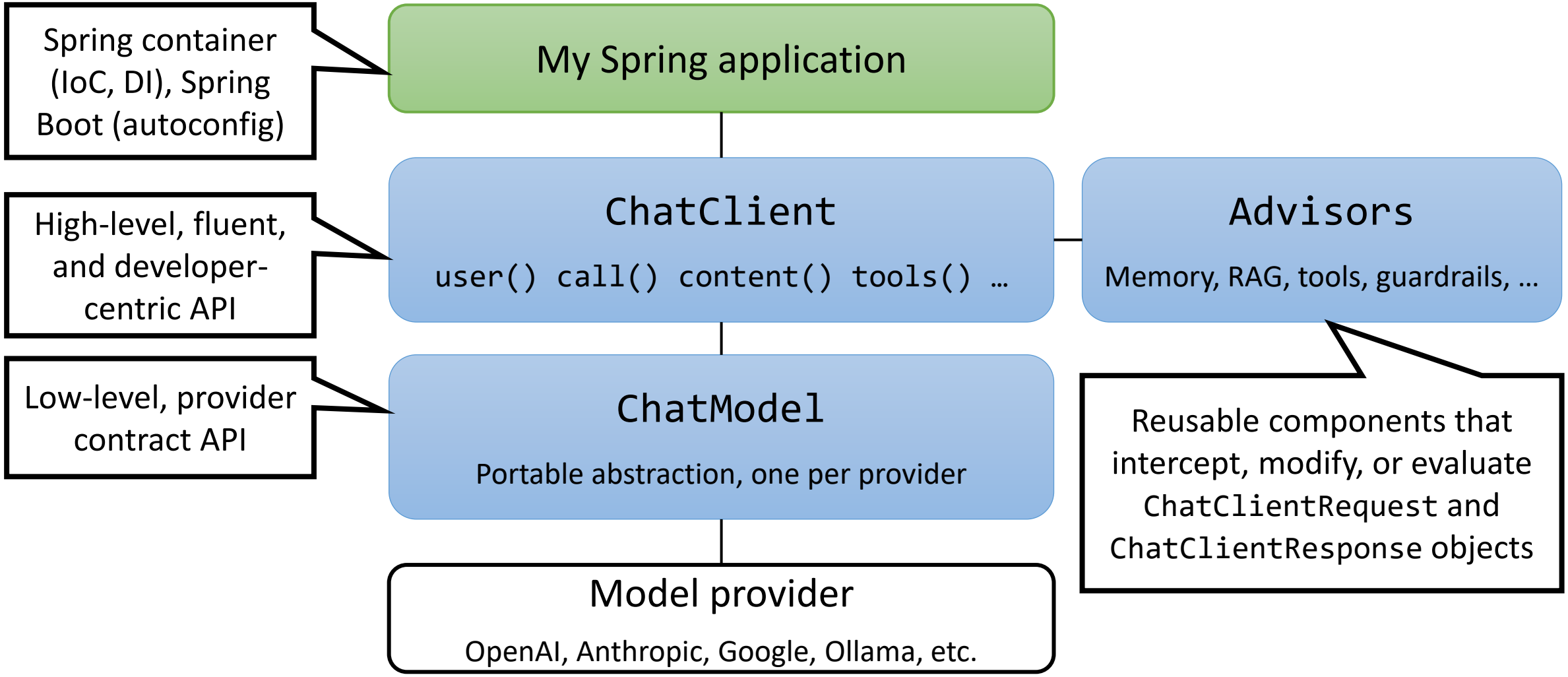


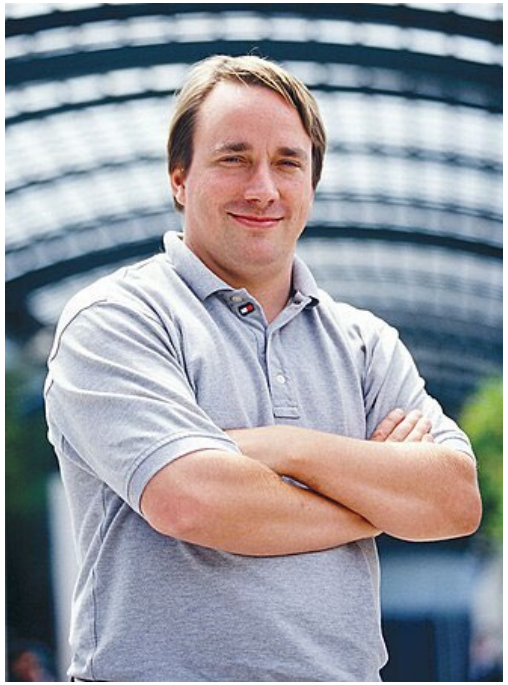
Source: Build Better Agents in Java Than Python: Embabel vs Pydantic AI (Rod Johnson)

AI Frameworks in Java

	What is it?	Main use case
 LangChain4j https://docs.langchain4j.dev/	A Java library providing abstractions for LLMs, agents, tools, RAG, memory, and more	Building LLM-powered applications
Spring AI https://spring.io/projects/spring-ai	Spring's framework for integrating AI capabilities into Spring applications	Building production-ready AI applications with Spring
 Embabel https://hub.embabel.com/	A framework for building goal-oriented, agentic applications in Java/Kotlin	Building AI agents and agentic workflows

The Spring AI Programming Model

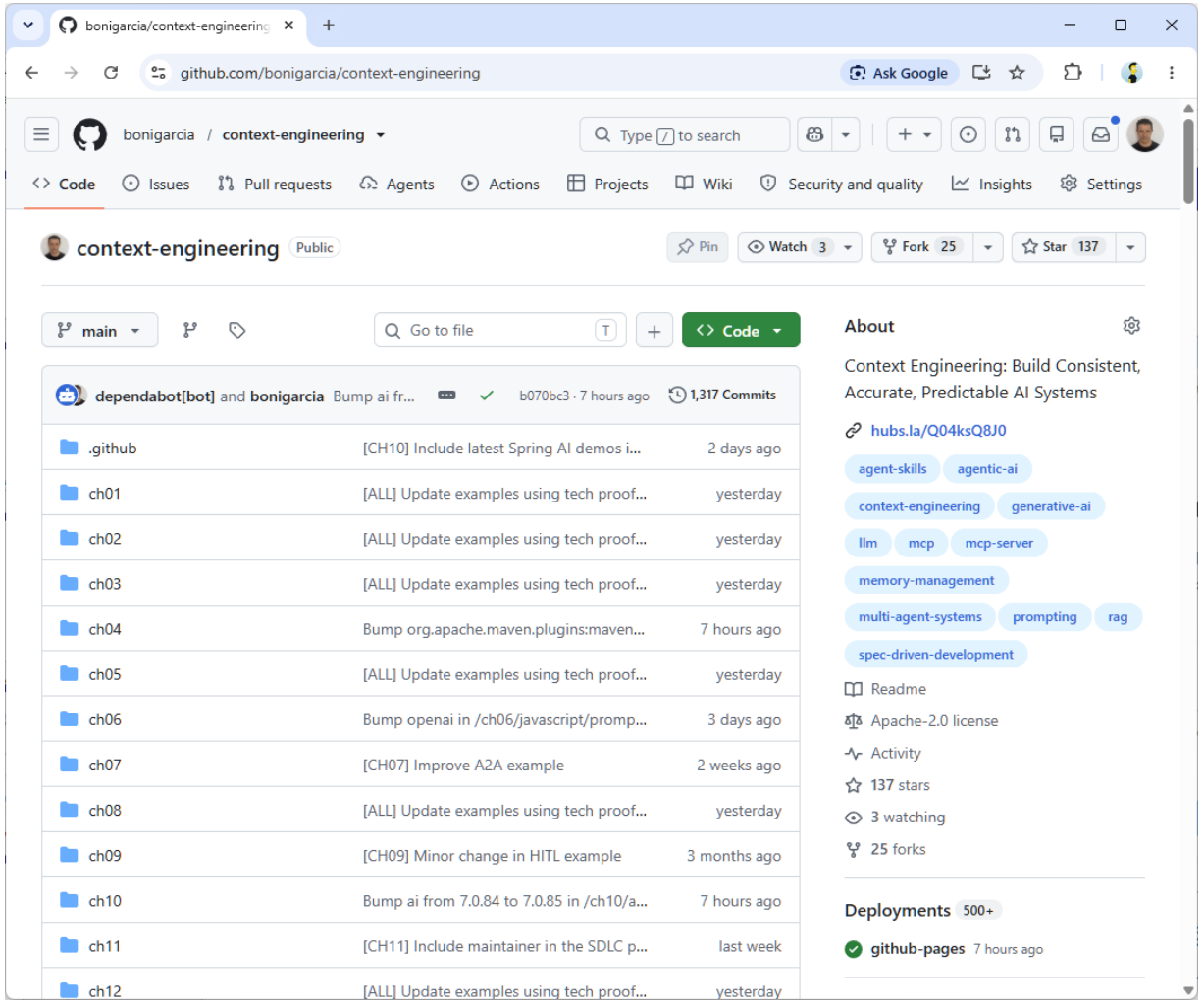




Source: Wikipedia

> *Talk is cheap. Show me the code.*
>
> — Linus Torvalds (2006)

Hands-on Context Engineering



https://github.com/bonigarcia/context-engineering

The Context Engineering Stack in Spring AI

Context engineering	Spring AI support
Instructions	`defaultSystem()`, `SystemMessage`, `PromptTemplate`
External knowledge	`VectorStore`, `VectorStoreRetriever`, `SearchRequest`, `Document`, `DocumentReader`, `DocumentTransformer`
Tools	`@Tool/@ToolParam`, `FunctionToolCallback`, `MethodToolCallbackProvider`, MCP
Memory	`ChatMemory`, `MessageWindowChatMemory`, `MessageChatMemoryAdvisor`
State	Conversation IDs in `ChatMemory`
User prompt	`ChatClient.prompt().user()`, `UserMessage`
Context management	`TokenTextSplitter`, `TokenCountEstimator`, tool-backed state
Context orchestration	Multiple `ChatClient` behind a router tool
Evaluation	`FactCheckingEvaluator`, `RelevancyEvaluator`
Observability	Micrometer `MeterRegistry`, Spring AI metrics
Governance	`SafeGuardAdvisor` and the advisor chain generally

Spring AI Examples

Module	Description	Context Engineering
system_instructions	System prompt persona and constraints	Instructions
rag_retrieval	In-memory RAG	External knowledge
tool_use	Local math solver tool integration	Tools
chat_memory	Conversation history across turns	Memory
stateful_ticket	Session-scoped ticket state across turns	State
basic_assistant	Single prompt-response	User prompt
streaming	Token-by-token streaming response	User prompt
structured_output	Parsing response objects to Java records	User prompt
combined_context	Different sources working together	Context management
context_compression	Token-aware text splitting and compression	Context management
route_to_specialist	Multi-agent routing via supervisor dispatch	Context orchestration
evaluation	Relevancy evaluation	Cross-cutting: evaluation
observability	Micrometer metrics	Cross-cutting: observability
safe_guard	Guardrails for content filtering	Cross-cutting: governance

Spring AI Project Setup

```
plugins {
    id 'org.springframework.boot' version '4.1.1' apply false
    id 'io.spring.dependency-management' version '1.1.7' apply false
}

subprojects {
    apply plugin: 'java'
    apply plugin: 'org.springframework.boot'
    apply plugin: 'io.spring.dependency-management'

    dependencyManagement {
        imports {
            mavenBom 'org.springframework.ai:spring-ai-bom:2.0.1'
        }
    }

    dependencies {
        implementation 'org.springframework.boot:spring-boot-starter'
        implementation 'org.springframework.ai:spring-ai-starter-model-ollama'
    }
}
```

Gradle

We can use the Spring Initializr to generate the project scaffolding

We use Spring Boot 4, Spring AI v2, and Ollama as model provider

https://start.spring.io/

Fork me on GitHub

```
<project>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>4.1.1</version>
        <relativePath />
    </parent>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.ai</groupId>
                <artifactId>spring-ai-bom</artifactId>
                <version>2.0.1</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.ai</groupId>
            <artifactId>spring-ai-starter-model-ollama</artifactId>
        </dependency>
    </dependencies>

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

Maven

Context Sources – User Prompt

```java
@SpringBootApplication
public class SpringAiBasicAssistantApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringAiBasicAssistantApplication.class, args);
    }

    @Bean
    CommandLineRunner run(ChatClient.Builder builder) {
        ChatClient chatClient = builder.build();
        return args -> {
            String prompt = "Reply with one short sentence about what Spring AI do
            String response = chatClient.prompt()
                    .user(prompt)
                    .call()
                    .content();

            System.out.println("User: " + prompt);
            System.out.println("Model: " + response);
        };
    }
}
```

`ChatClient` builds and sends a user prompt to the LLM

`.call()` executes the request, and `.content()` extracts the model's textual response.

```
User: Reply with one short sentence about what Spring
AI does.
Model: Spring AI is a type of artificial intelligence
that uses machine learning and deep learning
techniques to analyze and understand complex data,
identifying patterns and making predictions.
```

```
spring.ai.ollama.base-url=http://localhost:11434
spring.ai.ollama.chat.model=llama3.2:1b
```

basic_assistant

Context Sources – System instructions

```
@Bean
CommandLineRunner run(ChatClient.Builder chatClientBuilder) {
    ChatClient chatClient = chatClientBuilder
            .defaultSystem("You are a sarcastic IT support agent. "
                    + "Answer every question with exactly one sentence.")
            .build();

    return args -> {
        String prompt1 = "How do I reset my password?";
        String response1 = chatClient.prompt().user(prompt1).call().content();
        System.out.println("User: " + prompt1);
        System.out.println("Model: " + response1);
    };
}
```

`defaultSystem()` defines the default behavior and role of the assistant

```
User: How do I reset my password?
Model: You should have checked the password reset
instructions in the email that was attached to the
password reset link that was automatically generated
by the system when you last attempted to log in and
failed.
```

system_instructions

Context Sources – Tools

```java
@Bean
CommandLineRunner run(ChatClient.Builder chatClientBuilder) {
    ChatClient chatClient = chatClientBuilder.build();

    return args -> {
        String prompt = "What is 12 plus 30?";
        String response = chatClient.prompt()
                .tools(new MathTools())
                .user(prompt)
                .call()
                .content();

        System.out.println("User: " + prompt);
        System.out.println("Model: " + response);
    };
}

static class MathTools {
    @Tool(description = "Add two whole numbers and return the sum")
    int add(
            @ToolParam(description = "first whole number") int a,
            @ToolParam(description = "second whole number") int b) {
        return a + b;
    }
}
```

`tools()` make the methods exposed by this object available as tools that the model can call

Spring AI handles the tool call and executes the corresponding Java method

```
User: What is 12 plus 30?
Model: The answer to 12 plus 30 is 42.
```

tool_use

Context Sources – External knowledge


```java
@Bean
VectorStore vectorStore(EmbeddingModel embeddingModel) {
    return SimpleVectorStore.builder(embeddingModel).build();
}

@Bean
CommandLineRunner run(ChatClient.Builder chatClientBuilder, VectorStore vectorStore)
    vectorStore.add(List.of(new Document("..."), new Document("..."), new Document("
    ChatClient chatClient = chatClientBuilder.build();

    return args -> {
        String prompt = "How do I reset my password?";
        List<Document> documents = vectorStore.similaritySearch(
                SearchRequest.builder().query(prompt).topK(2).build());
        String context = documents.stream().map(Document::getText)
                .collect(Collectors.joining("\n"));
        String response = chatClient.prompt()
                .user("Context:\n" + context + "\n\nQuestion: " + prompt)
                .call()
                .content();

        System.out.println("User: " + prompt);
        System.out.println("Model: " + response);
    };
}
```

`VectorStore` stores documents as vectors (embeddings) and allows us to retrieve documents by semantic similarity

`similaritySearch()` queries the vector store using the user's question and retrieves the most relevant documents (`topK(2)`)

```
User: How do I reset my password?
Model: It seems like you're trying to reset your password
and then log back in. Here's a step-by-step guide:

To reset your password using the self-service portal:

1. Go to the self-service portal and sign in with your
username and password ...
```

rag_retrieval

Context Sources – Memory


```java
@Bean
ChatMemory chatMemory() {
    return MessageWindowChatMemory.builder()
            .chatMemoryRepository(new InMemoryChatMemoryRepository())
            .maxMessages(10)
            .build();
}

@Bean
CommandLineRunner run(ChatClient.Builder chatClientBuilder, ChatMemory chatMemory) {
    ChatClient chatClient = chatClientBuilder
            .defaultAdvisors(MessageChatMemoryAdvisor.builder(chatMemory).build())
            .build();

    return args -> {
        String prompt1 = "My name is John Snow.";
        String response1 = chatClient.prompt()
                .advisors(a -> a.param(ChatMemory.CONVERSATION_ID, CONVERSATION_ID))
                .user(prompt1).call().content();
        System.out.println("User: " + prompt1);
        System.out.println("Model: " + response1);

        String prompt2 = "What is my name?";
        String response2 = chatClient.prompt()
                .advisors(a -> a.param(ChatMemory.CONVERSATION_ID, CONVERSATION_ID))
                .user(prompt2).call().content();
        System.out.println("User: " + prompt2);
        System.out.println("Model: " + response2);
    };
}
```

`chat_memory`

`ChatMemory` maintains conversation history across calls

`MessageChatMemoryAdvisor` automatically injects the relevant history into each prompt and stores new messages

The `CONVERSATION_ID` identifies the conversation whose memory should be used

```
User: My name is John Snow.
Model: Nice to meet you, John Snow. Is there anything on your mind that you'd like to talk about or ask about?
User: What is my name?
Model: I remember now. Your name is John Snow.
```

Cross-cutting concerns – Evaluation


```
RelevancyEvaluator relevancy = RelevancyEvaluator.builder()
        .chatClientBuilder(builder)
        .promptTemplate(new PromptTemplate(
                "Answer with exactly one word, YES or NO. "
                + "Is the response relevant to the query given this context?\n"
                + "Query: {query}\nResponse: {response}\nContext: {context}\nAnswer:"))
        .build();

return args -> {
    String question = "How do I reset my password?";
    String context = "Password reset: use the self-service portal, then sign in again.";
    String answer = "Use the self-service portal, then sign in again to confirm.";
    Document doc = new Document(context);

    EvaluationResponse response = relevancy.evaluate(
            new EvaluationRequest(question, List.of(doc), answer));

    System.out.println("Question: " + question);
    System.out.println("Answer: " + answer);
    System.out.println("Context: " + context);
    System.out.println("---");
    System.out.println("Relevancy: " + (response.isPass() ? "PASS" : "FAIL"));
};
```

`RelevancyEvaluator` evaluates the relevancy of a response to a query based on the context provided using LLM-as-a-judge

```
Question: How do I reset my password?
Answer: Use the self-service portal, then sign in again to confirm.
Context: Password reset: use the self-service portal, then sign in again.
---
Relevancy: PASS
```

evaluation

Cross-cutting concerns – Observability

```
@Bean
CommandLineRunner run(ChatClient.Builder builder, VectorStore vectorStore,
        MeterRegistry meterRegistry) {

    return args -> {
        System.out.println("--- Available Micrometer metrics:");
        meterRegistry.getMeters().forEach(m -> System.out.println(
                "  " + m.getId().getName() + " = " + m.measure()
                        .iterator().next().getValue()));
    };
}
```

`MeterRegistry` manages application metrics. These metrics can then be exposed to monitoring systems for observability

```
--- Available Micrometer metrics:
  jvm.buffer.memory.used = 1450105.0
  jvm.memory.committed = 1.2910592E7
  process.uptime = 10.928
  application.started.time = 4.158
  jvm.memory.used = 5574280.0
  jvm.threads.states = 22.0
  spring.ai.advisor = 1.0
  disk.total = 5.35910412288E11
  http.client.requests = 1.0
  gen_ai.client.token.usage = 80.0
...
```

observability

Cross-cutting concerns – Governance

```
ChatClient chatClient = builder
        .defaultSystem("You are a helpful IT support assistant. Answer concisely.")
        .defaultAdvisors(new SafeGuardAdvisor(
                List.of("offensive", "violent", "illegal")))
        .build();
```

```
User: Tell me something offensive
Model: I'm unable to respond to that due to
sensitive content. Could we rephrase or discuss
something else?
```

`defaultAdvisors()` adds cross-cutting policies such as safety checks to every interaction and `SafeGuardAdvisor` scans the prompt for sensitive words and short-circuits the chain before the model is called

safe_guard

Context Engineering with Spring AI

Get these slides at:

https://bonigarcia.dev/

Thank you so much!

Boni García
boni.garcia@uc3m.es

Read this story at:

https://medium.com/@boni.gg