LangChain4j in Action: A Walkthrough from Basic Chaining to Agentic Workflows

Phase 1: Context & Use Case

Phase 2: LangChain4j Capabilities

Phase 3: Problems with Naive Pipelines

Into Agentic Workflows

whoami

Iqbal Aissaoui, SWE

- **SWE @ Mindera (3+ years)**
- **Java Ecosystem + DevOps (AWS Cloud)**
- **Based in Casablanca**
- **Avid Runner**
- **Animal Lover**

Phase 1: Context & Use Case

Phase 1: Context & Use Case

Main Drivers:

- **Autonomy (Built by you, for you)**: The frustration of relying on rigid, black-box SaaS tools.

You want a system where you control the logic, the data, and the execution.

Phase 1: Context & Use Case

Main Drivers:

- **Automation (Set once, run on schedule):** Building a path for repeatable, codified workflows. The AI CLI engine allows you to take a complex mental workflow (e.g., research -> filter -> format) and lock it into an immutable, version-controlled pipeline.

You architect the sequence once, hand the keys over to a CI/CD cron schedule, and let it execute autonomously

Phase 1: Context & Use Case

Main Drivers:

- **Empowering Individuals:** Social media platforms optimize their feeds for attention grabbing. Their algorithms actively, filter, suppress, or bury the exact niche information you actually care about (like the registration date for a local 10k race) in favor of engagement bait.

By orchestrating LLM Calls, you systematically bypass the platform's biased feed. This is a powerful usage of AI to do **curation directly on your behalf**

Phase 1: Context & Use Case

Main Drivers:

Vendor Agnostic & Programmatic Interface: The AI landscape is incredibly volatile. If you hardcode specific SDKs or write custom API calls for OpenAI or Google, your entire application becomes tightly coupled to that provider's pricing, downtime, and ecosystem.

LangChain4j acts as a **universal adapter**. By developing against its programmatic, dynamic interfaces, you build your Java logic once and make the underlying LLM completely interchangeable.

Phase 1: Context & Use Case

Use Case:

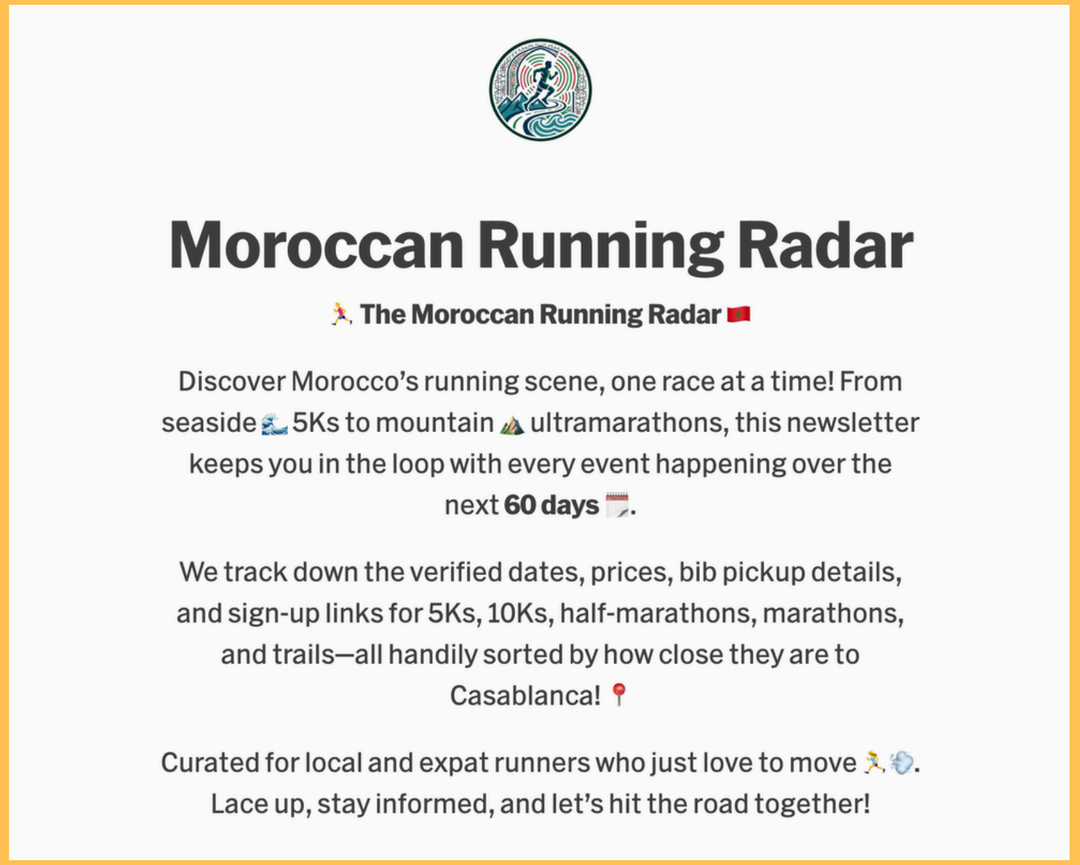
- **The Problem with "Chat"** It is manual, conversational, and disconnected from our systems. **Chat is great for brainstorming, but it is terrible for repeatable engineering.**
- **The AI CLI Paradigm (The "Engine")** We stop treating LLMs as conversational partners. Instead, we embed them directly into the terminal as **programmable utility functions**.

The True Value

The goal is to build autonomous pipelines that can:

- Extract chaotic, unstructured data from the web.
- Force that data through strict, deterministic validation schemas.
- Output highly curated, high-signal intelligence on a perfectly reliable schedule.

Phase 1: Context & Use Case



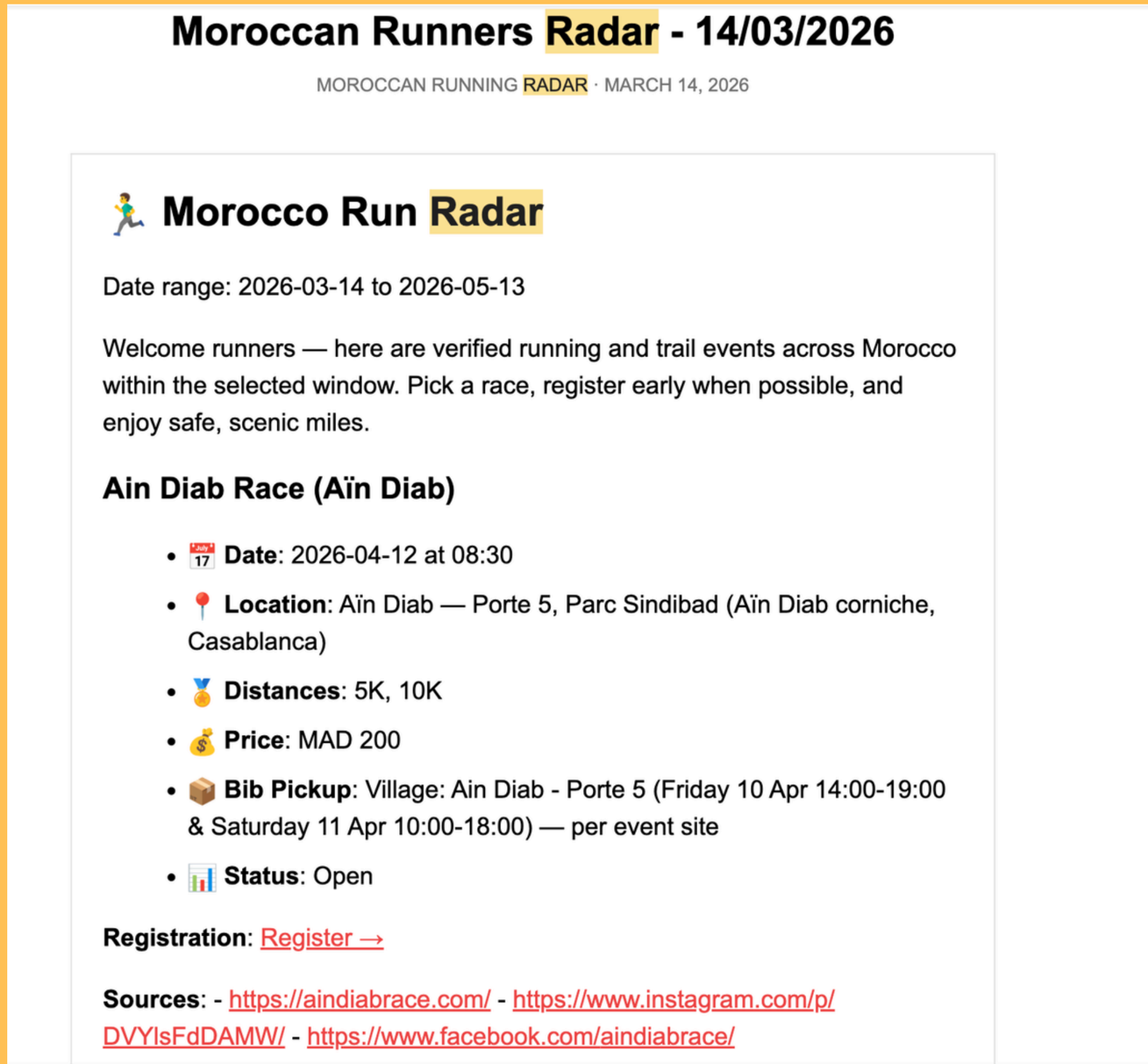
Moroccan Running Radar

🏃‍♂️ **The Moroccan Running Radar** 🇲🇦

Discover Morocco's running scene, one race at a time! From seaside 🌊 5Ks to mountain ⛰️ ultramarathons, this newsletter keeps you in the loop with every event happening over the next **60 days** 🗓️.

We track down the verified dates, prices, bib pickup details, and sign-up links for 5Ks, 10Ks, half-marathons, marathons, and trails—all handily sorted by how close they are to Casablanca! 📍

Curated for local and expat runners who just love to move 🏃‍♀️💨. Lace up, stay informed, and let's hit the road together!



Moroccan Runners Radar - 14/03/2026

MOROCCAN RUNNING RADAR · MARCH 14, 2026

🏃‍♂️ Morocco Run Radar

Date range: 2026-03-14 to 2026-05-13

Welcome runners — here are verified running and trail events across Morocco within the selected window. Pick a race, register early when possible, and enjoy safe, scenic miles.

Ain Diab Race (Aïn Diab)

- 📅 **Date**: 2026-04-12 at 08:30
- 📍 **Location**: Aïn Diab — Porte 5, Parc Sindibad (Aïn Diab corniche, Casablanca)
- 🏅 **Distances**: 5K, 10K
- 💰 **Price**: MAD 200
- 📦 **Bib Pickup**: Village: Ain Diab - Porte 5 (Friday 10 Apr 14:00-19:00 & Saturday 11 Apr 10:00-18:00) — per event site
- 📊 **Status**: Open

Registration: Register →

Sources: - https://aindiabrace.com/ - https://www.instagram.com/p/DVYIsFdDAMW/ - https://www.facebook.com/aindiabrace/

Phase 1: Context & Use Case

Moroccan IT Events AI <MS_A1xSrg@iqbalaissaoui.com> Thu, Feb 26, 11:01 PM

to me

💻 Tech Radar: March 2026 Events in Morocco

Casablanca • Rabat • National

📍 Casablanca

[HACKATHON] Rab'Hacks 2026 - Student Hackathon (EIGSI Casablanca)

- 📅 **Date**: 2026-03-13 at 00:00
- 📍 **Venue**: EIGSI Casablanca / Remote
- 📝 **Description**: Rab'Hacks 2026 (student track) is a month-long hackathon (13 Feb – 13 Mar 2026) focused on prototyping with AI and no-code tools. EIGSI Casablanca hosted the local launch and runs region activities and mentorship for students.
- 🔗 Register / Details →

[CONFERENCE] India-Africa ICT Expo 2026

- 📅 **Date**: 2026-03-27 at 09:00
- 📍 **Venue**: Hyatt Regency Casablanca
- 📝 **Description**: Two-day ICT expo (27–28 Mar 2026) connecting Indian and African telecom & ICT companies with panels, exhibitors and B2B matchmaking.
- 🔗 Register / Details →

📍 Rabat & Others

- No events from the provided list are scheduled in Rabat or other cities within the next 30 days.

Keep coding, keep building.

Generated by AI.

Phase 1: Context & Use Case

🎓 Academic Medicine Job Radar

2026-03-16 → 2026-05-15 • Medicine/public health focus • Outside Morocco • UK and South Africa prioritized • Assistant → Professor-equivalent levels

Metric	Count
Total roles	3
UK	3
South Africa	0
Other	0

Level distribution:

- Assistant: 1
- Associate: 1
- Senior Lecturer/Reader: 1
- Professor-equivalent: 0

🇬🇧 United Kingdom

Associate Professorship (clinical) of Primary Care Health Sciences

- **Level**: Associate
- 🏛️ **Institution**: University of Oxford — Nuffield Department of Primary Care Health Sciences
- 📍 **Location**: Oxford, United Kingdom
- 💼 **Type**: Full time
- 💰 **Salary**: Combined University and College salary from £57,986 to £77,366 (where stated)
- ⏰ **Deadline**: **2026-03-16**
- 📝 **Summary**: Clinical Associate Professorship in primary care/public health with leadership in clinical research and teaching; fits clinicians/researchers working at the interface of medicine and public health
- 🔗 Apply →
- Sources:
  - https://www.phc.ox.ac.uk/about/work-with-us
  - https://data.ox.ac.uk/doc/vacancy/173887.naturejobs-xml
  - https://my.corehr.com/pls/uoxrecruit/erq_search_version_4.start_search_with_params?p_display_in_irish=N&p_internal_external=E&p_force_type=E&p_company=10&p_refresh_search=Y

Reply Forward

Phase 2: LangChain4j Capabilities

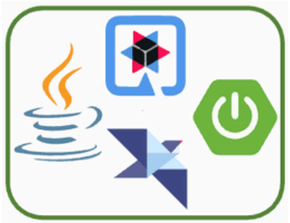
LangChain4j

Supercharge your Java application with the power of LLMs

Introduction

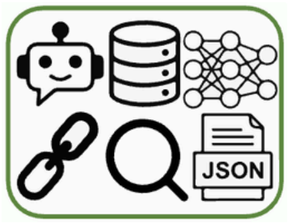
Easy interaction with LLMs and Vector Stores

All major commercial and open-source LLMs and Vector Stores are easily accessible through a unified API, enabling you to build chatbots, assistants and more.



Tailored for Java

Smooth integration into your Java applications is made possible thanks to Quarkus, Spring Boot and Helidon integrations. There is two-way integration between LLMs and Java: you can call LLMs from Java and allow LLMs to call your Java code in return.

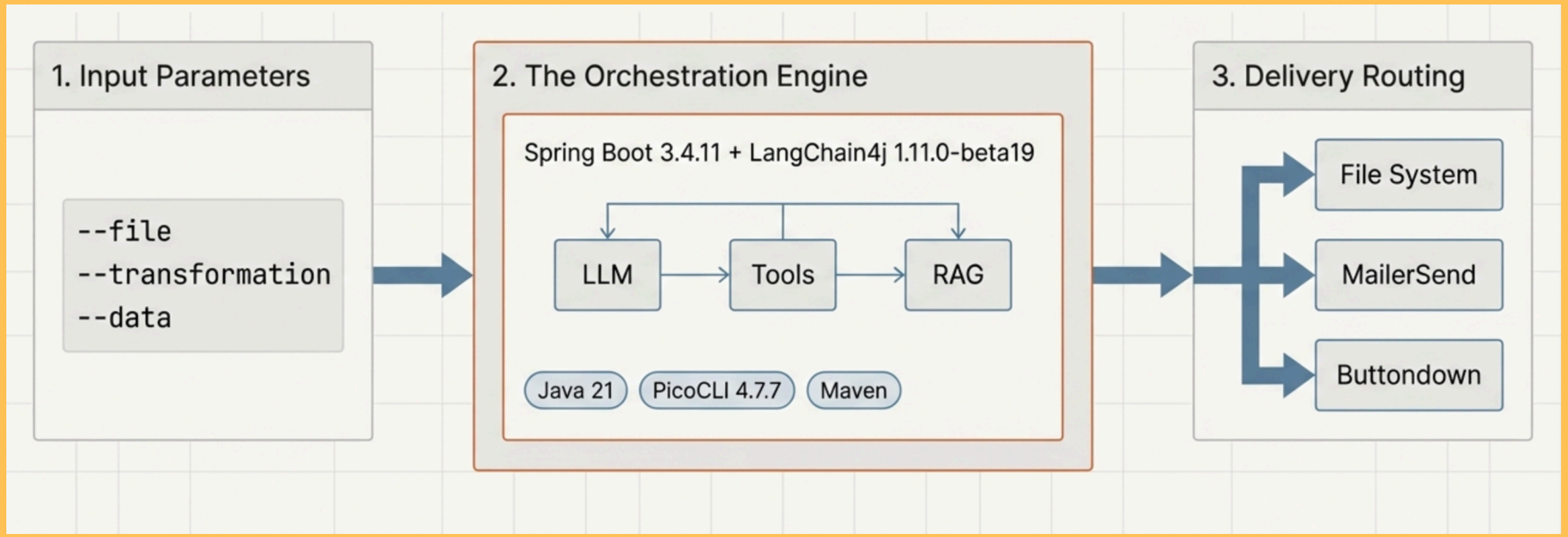


Agents, Tools, RAG

Our extensive toolbox provides a wide range of tools for common LLM operations, from low-level prompt templating, chat memory management, and output parsing, to high-level patterns like Agents and RAG.

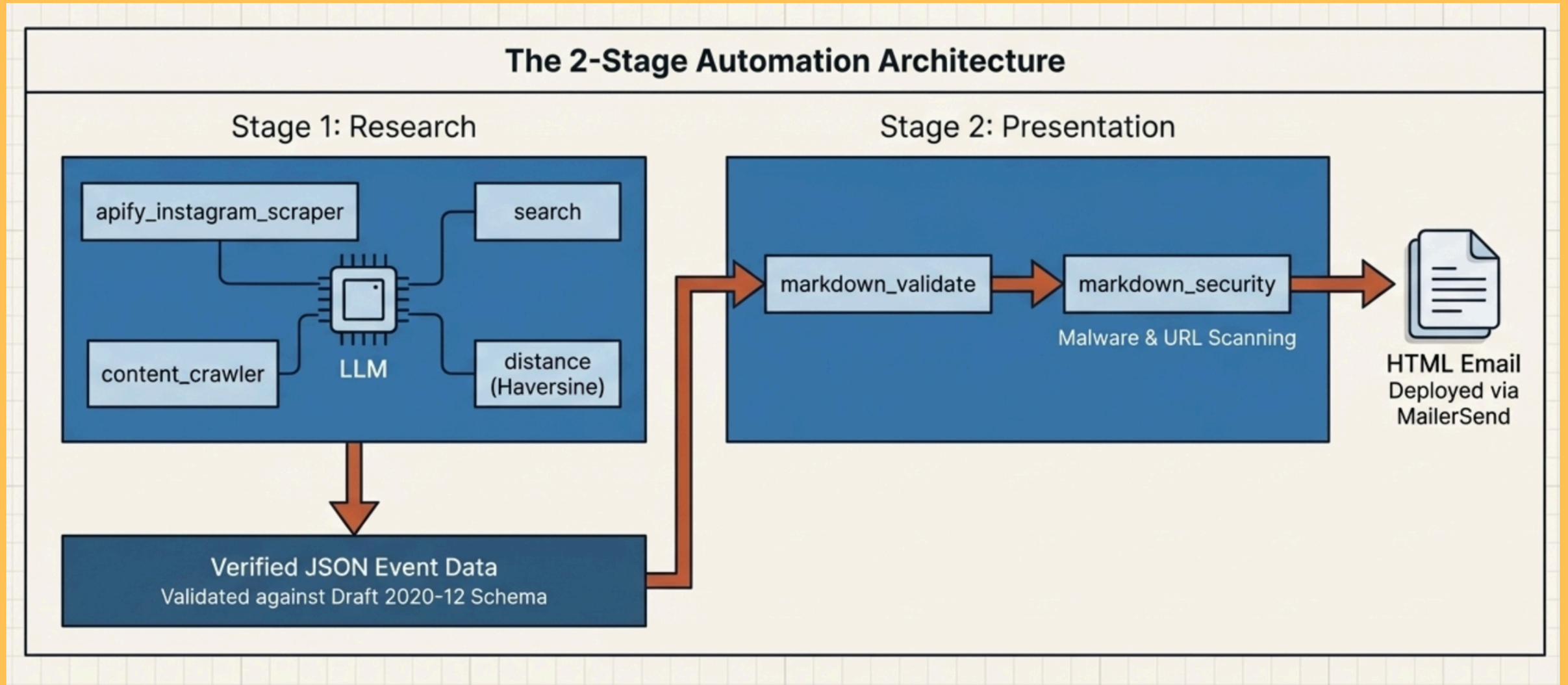
Phase 2: LangChain4j Capabilities

The Architecture



Phase 2: LangChain4j Capabilities

The Architecture



Phase 2: LangChain4j Capabilities

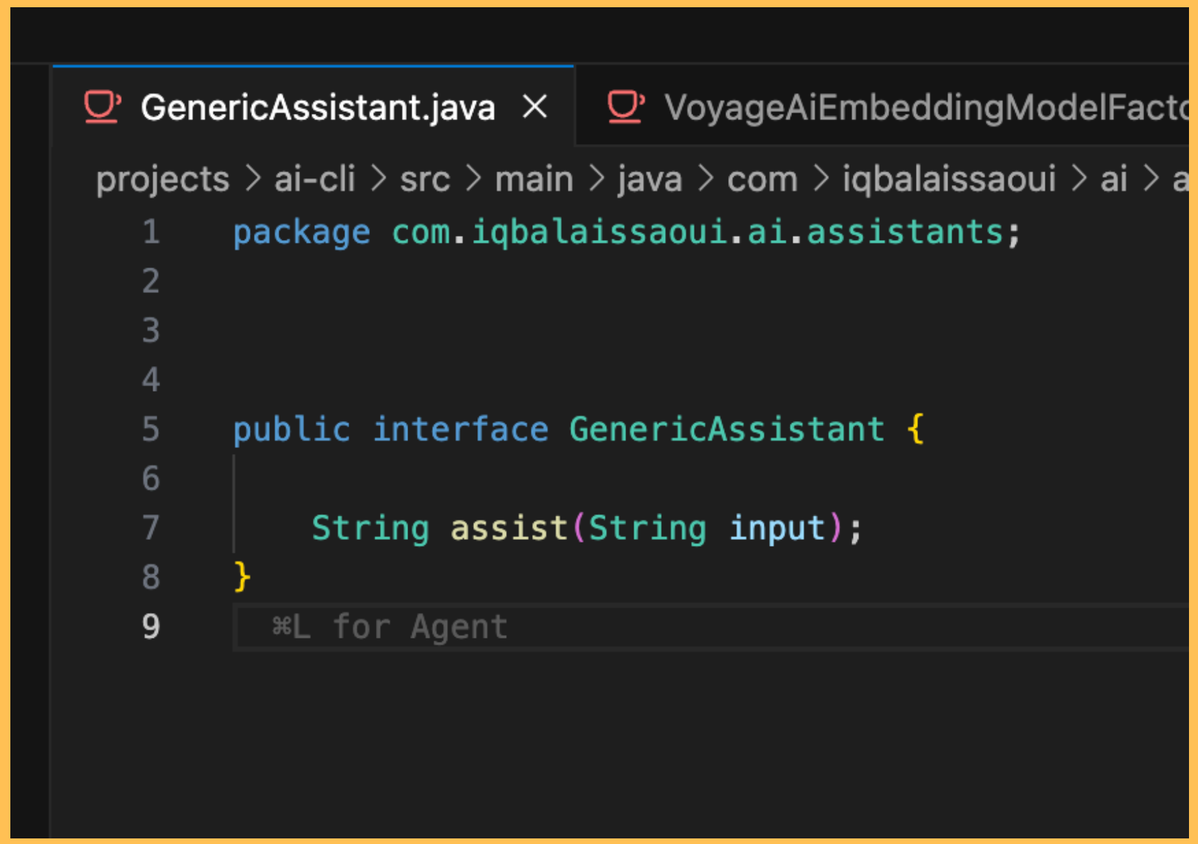
The 60-Second Glossary:

- **Chat Model:** It takes text in, and computes text out.
- **Embedding Model:** "The Translator." It converts a string of text into a giant array of floats (numbers). It captures semantic meaning into math so a database can understand it.
- **Embedding Store (Vector DB):** "The Database." Just like PostgreSQL stores rows, this stores those arrays of numbers and lets us do lightning-fast similarity searches.
- **Reranker:** "The Filter/Judge." When the database returns 10 results, the reranker forces an AI model to score them from 1 to 10 so we only keep the absolute most relevant data.
- **Tools:** Break out of LLM Cut Off Date, Access to Traditional Code

Load A ChatModel Bean

```java
import java.time.Duration;

@Component
public class OpenAiChatModelFactory implements ChatModelFactory {

    private static final Logger log = LoggerFactory.getLogger(OpenAiChatModelFactory.class);

    @Override
    public boolean supports(String modelName, ProviderProperties properties) {
        if (modelName == null || modelName.isBlank())
            return false;
        if (properties.getOpenai() == null || properties.getOpenai().getSupportedModels() == null) {
            return false;
        }
        return properties.getOpenai().getSupportedModels().stream()
                .anyMatch(model -> model.equalsIgnoreCase(modelName));
    }

    @Override
    public ChatModel create(String modelName, ProviderProperties properties) {
        log.info("Instantiating OpenAI Chat Model from PicoCLI: {}", modelName);
        return OpenAiChatModel.builder()
                .apiKey(properties.getOpenai().getApiKey())
                .baseUrl(properties.getOpenai().getApiBaseUrl())
                .maxRetries(1000)
                .logRequests(true)
                .logResponses(true)
                .timeout(Duration.ofSeconds(properties.getOpenai().getTimeoutSeconds()))
                .maxTokens(properties.getOpenai().getMaxTokens() != null ? properties.getOpenai().getMaxTokens()
                .modelName(modelName)
                .build();
    }
}
```

Declare an Intercace

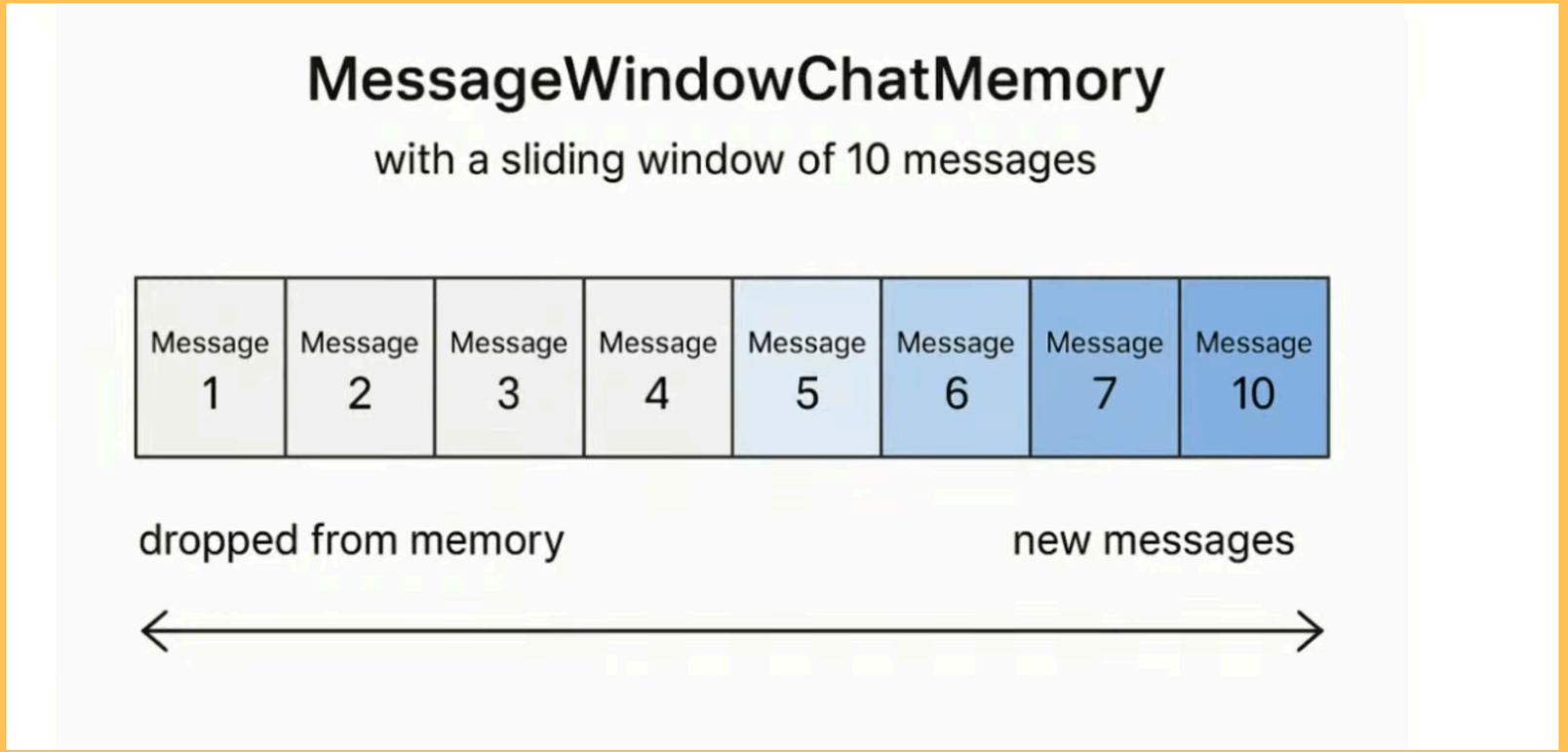


Build an AI Service

```
7
8   @Service
9   public class SimpleChatService {
0
1       private final GenericAssistant assistant;
2
3       public SimpleChatService(ChatModel chatModel) {     Pin selection to
4           this.assistant = AiServices.builder(GenericAssistant.class)
5                   .chatModel(chatModel)
6                   .build();
7       }
8
9       public String chat(String input) {
0           return assistant.assist(input);
1       }
2   }
3
```

Give it Short Term Memory

- LLMs are Stateless By Design
- Multi-turn conversations
- Makes tool calls possible
- Prevents token explosion.



Short Term Memory

```java

@Service
public class SimpleChatService {

    private final GenericAssistant assistant;

    // Single shared memory store — fine for a simple chatbot.
    // But in a multi-stage pipeline, each stage needs its OWN isolated store
    // to prevent context pollution (Stage 2 seeing Stage 1's tool interactions).
    // See AssistantService.build() for the per-stage isolation pattern.
    private final InMemoryChatMemoryStore memoryStore = new InMemoryChatMemoryStore();

    public SimpleChatService(ChatModel chatModel) {
        this.assistant = AiServices.builder(GenericAssistant.class)
                .chatModel(chatModel)
                .chatMemory(MessageWindowChatMemory.builder()
                        .maxMessages(20)
                        .chatMemoryStore(memoryStore)
                        .build())
                .build();
    }

    public String chat(String input) {
        return assistant.assist(input);
    }
}

```

Phase 2: LangChain4j Capabilities

The 60-Second Glossary:

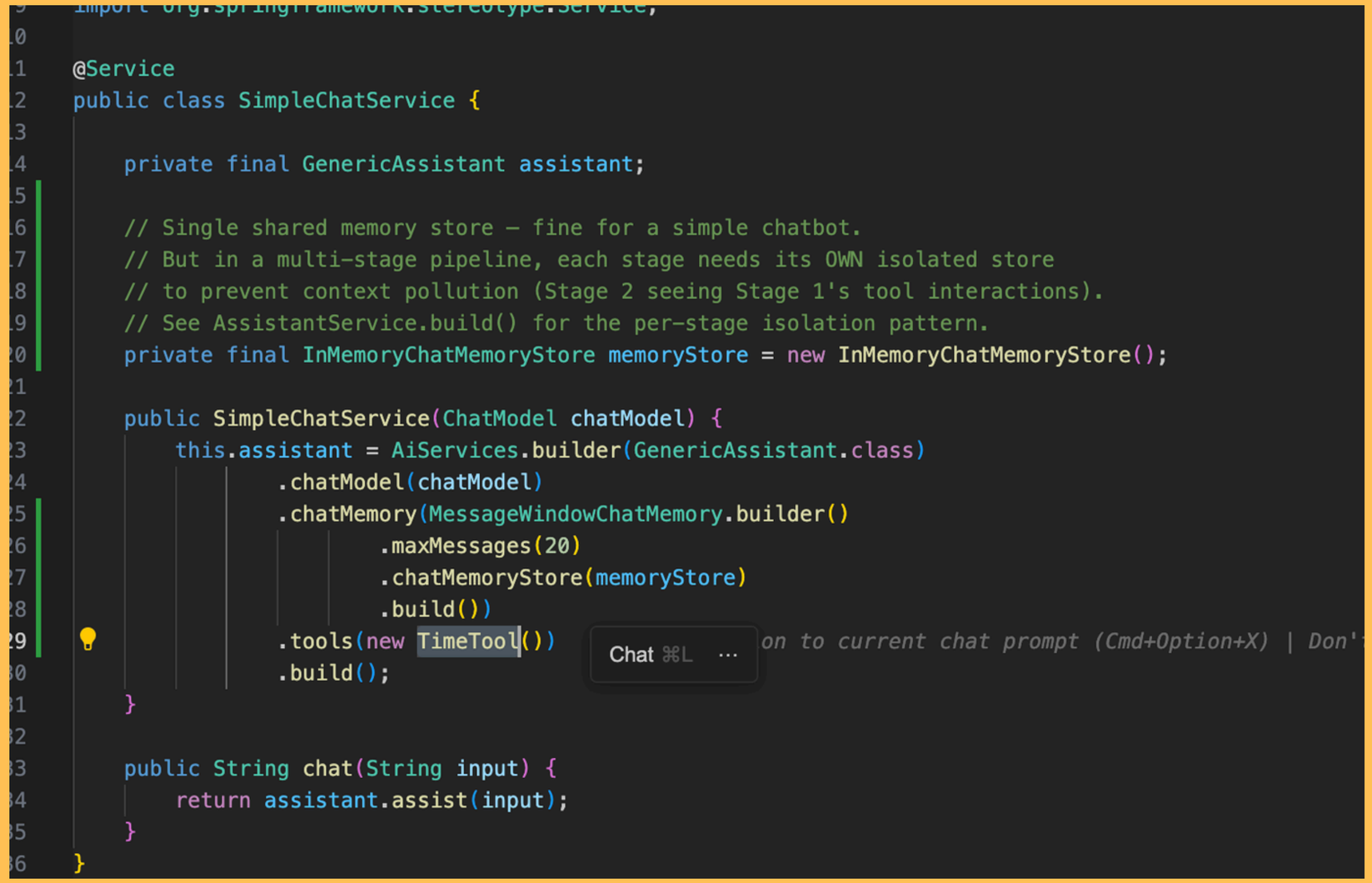
- **Chat Model:** It takes text in, and computes text out. Period.
- **Embedding Model:** "The Translator." It converts a string of text into a giant array of floats (numbers). It captures semantic meaning into math so a database can understand it.
- **Embedding Store (Vector DB):** "The Database." Just like PostgreSQL stores rows, this stores those arrays of numbers and lets us do lightning-fast similarity searches.
- **Reranker:** "The Filter/Judge." When the database returns 10 results, the reranker forces an AI model to score them from 1 to 10 so we only keep the absolute most relevant data.
- **Tools:** Break out of LLM Cut Off Date, Access to Traditional Code

Give it Tools

```java
@Conditional(NowEnabledCondition.class)
@Component
public class TimeTool {

    private static final Logger log = LoggerFactory.getLogger(TimeTool.class);

    @Tool("Provides the current search window range (today -> today + 60 days), formatted as ISO 8601 strings. Use t
    public String time() {
        log.info("TimeTool.time() called to get the search window range.");
        String today = ZonedDateTime.now().toInstant().toString();
        String todayPlus60 = ZonedDateTime.now().plusDays(60).toInstant().toString();
        return today + " -> " + todayPlus60;
    }
}
```

LLMs can't check the clock

Give it Tools



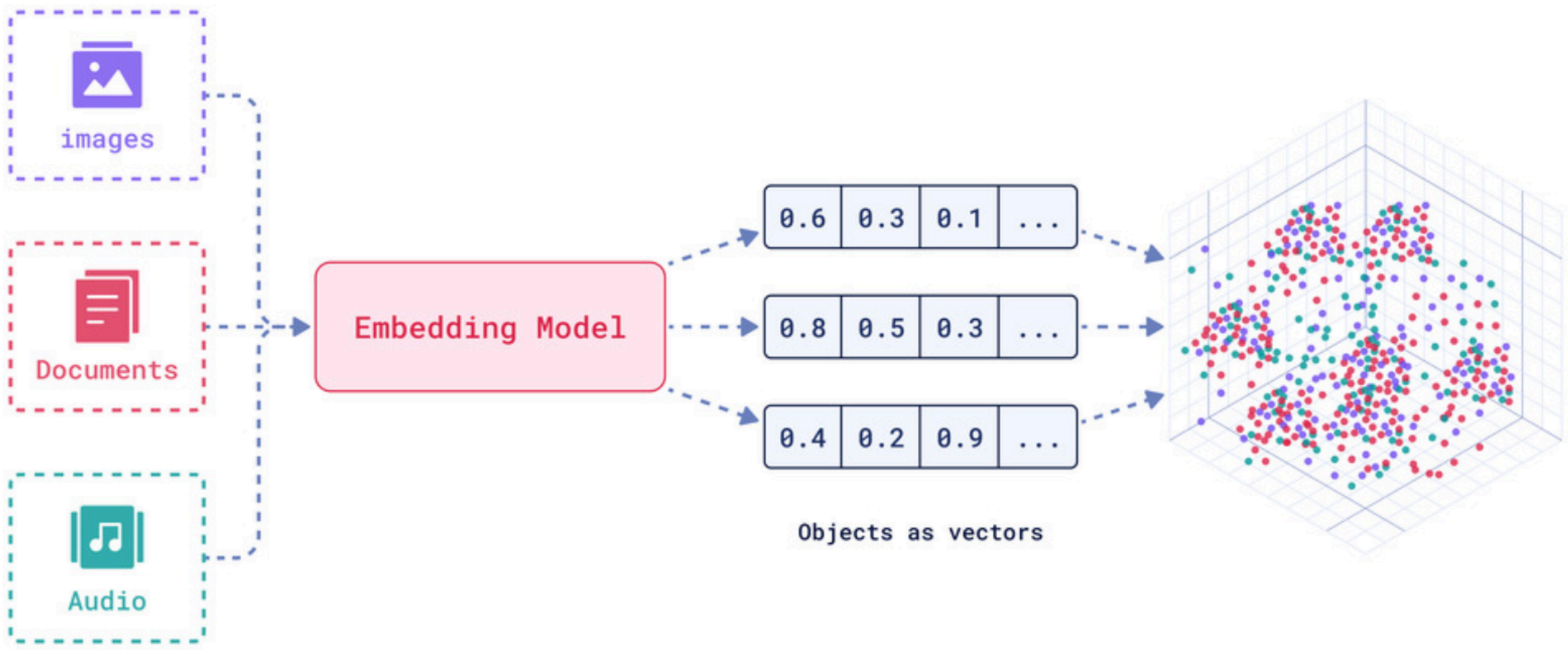
Phase 2: LangChain4j Capabilities
RAG

Phase 2: LangChain4j Capabilities

The 60-Second Glossary:

- **Chat Model:** It takes text in, and computes text out. Period.
- **Embedding Model:** "The Translator." It converts a string of text into a giant array of floats (numbers). It captures semantic meaning into math so a database can understand it.
- **Embedding Store (Vector DB):** "The Database." Just like PostgreSQL stores rows, this stores those arrays of numbers and lets us do lightning-fast similarity searches.
- **Reranker:** "The Filter/Judge." When the database returns 10 results, the reranker forces an AI model to score them from 1 to 10 so we only keep the absolute most relevant data.
- **Tools:** Break out of LLM Cut Off Date, Access to Traditional Code

Phase 2: LangChain4j Capabilities



Load An Embedding Model Bean

```java
@Component
public class OpenAiEmbeddingModelFactory implements EmbeddingModelFactory {

    private static final Logger log = LoggerFactory.getLogger(OpenAiEmbeddingModelFactory.class);

    @Override
    public boolean supports(String modelName, ProviderProperties properties) {
        if (modelName == null || modelName.isBlank())
            return false;
        if (properties.getOpenai() == null || properties.getOpenai().getSupportedEmbeddingModels() == nul
            return false;
        }
        return properties.getOpenai().getSupportedEmbeddingModels().stream()
                .anyMatch(model -> model.equalsIgnoreCase(modelName));
    }

    @Override
    public EmbeddingModel create(String modelName, ProviderProperties properties) {
        log.info("Instantiating OpenAI Embedding Model: {}", modelName);
        return OpenAiEmbeddingModel.builder()
                .apiKey(properties.getOpenai().getApiKey())
                .modelName(modelName)
                .logRequests(true)
                .logResponses(true)
                .timeout(Duration.ofHours(1))
                .build();
    }
}
```

Phase 2: LangChain4j Capabilities

The 60-Second Glossary:

- **Chat Model:** It takes text in, and computes text out. Period.
- **Embedding Model:** "The Translator." It converts a string of text into a giant array of floats (numbers). It captures semantic meaning into math so a database can understand it.
- **Embedding Store (Vector DB):** "The Database." Just like PostgreSQL stores rows, this stores those arrays of numbers and lets us do lightning-fast similarity searches.
- **Reranker:** "The Filter/Judge." When the database returns 10 results, the reranker forces an AI model to score them from 1 to 10 so we only keep the absolute most relevant data.
- **Tools:** Break out of LLM Cut Off Date, Access to Traditional Code

Load An Embedding Store Bean

```java
import org.springframework.stereotype.Component;

@Component
public class QdrantEmbeddingStoreFactory implements EmbeddingStoreFactory {

    private static final Logger log = LoggerFactory.getLogger(QdrantEmbeddingStoreFactory.class);

    @Override
    public boolean supports(String storeName) {
        return ProviderConstants.EMBEDDING_STORE_QDRANT.equalsIgnoreCase(storeName);
    }

    @Override
    public EmbeddingStore<TextSegment> create(String storeName, ProviderProperties properties) {
        log.info("Creating Qdrant embedding store bean");
        var qdrantProps = properties.getStore().getQdrant();
        log.info("Connecting to Qdrant Collection: {}", qdrantProps.getCollectionName());

        if (qdrantProps == null || qdrantProps.getHost() == null || qdrantProps.getHost().isBlank()) {
            throw new IllegalArgumentException(
                    "Qdrant host must be configured in provider.store.qdrant.host when using qdrant store");
        }
        if (qdrantProps.getApiKey() == null || qdrantProps.getApiKey().isBlank()) {
            throw new IllegalArgumentException(
                    "Qdrant API Key must be configured in provider.store.qdrant.api-key when using qdrant store");
        }

        io.qdrant.client.QdrantClient client = new io.qdrant.client.QdrantClient(
                io.qdrant.client.QdrantGrpcClient.newBuilder(qdrantProps.getHost(), qdrantProps.getPort(),
                        qdrantProps.getUseTls())
                        .withApiKey(qdrantProps.getApiKey())
                        .build());

        return QdrantEmbeddingStore.builder()
                .client(client)
                .collectionName(qdrantProps.getCollectionName())
                .build();
    }
}
```

Ingesting Data

```java
// Step 1: Load documents → split into chunks → embed → store
public void ingest(List<Path> files) {
    List<Document> documents = files.stream()
            .map(FileSystemDocumentLoader::loadDocument)
            .toList();

    EmbeddingStoreIngestor.builder()
            .embeddingModel(embeddingModel)
            .embeddingStore(embeddingStore)
            .build()
            .ingest(documents);
}
```

Enable Retrieval

```
    // Step 2: Chat with RAG — retrieves relevant chunks automatically
    public String chat(String input) {
        GenericAssistant assistant = AiServices.builder(GenericAssistant.class)
                .chatModel(chatModel)
                .chatMemory(MessageWindowChatMemory.builder()
                        .maxMessages(20)
                        .chatMemoryStore(new InMemoryChatMemoryStore())
                        .build())
                .retrievalAugmentor(DefaultRetrievalAugmentor.builder()
                        .contentRetriever(EmbeddingStoreContentRetriever.builder()
                                .embeddingModel(embeddingModel)
                                .embeddingStore(embeddingStore)
                                .maxResults(5)
                                .build())
                        .build())
                .build();

        return assistant.assist(input);
    }
}
```

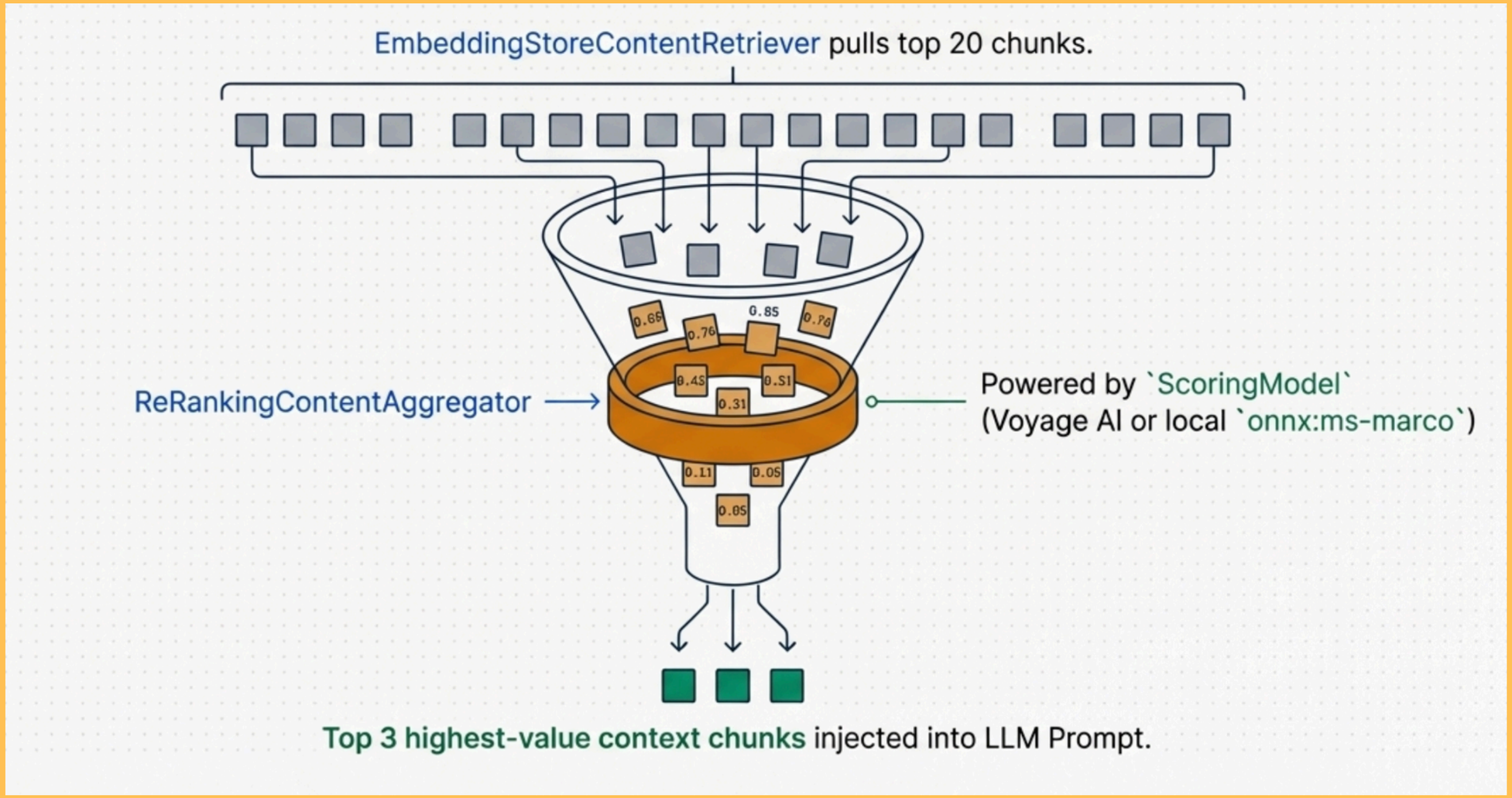
Phase 2: LangChain4j Capabilities
Advanced RAG

Phase 2: LangChain4j Capabilities

The 60-Second Glossary:

- **Chat Model:** It takes text in, and computes text out. Period.
- **Embedding Model:** "The Translator." It converts a string of text into a giant array of floats (numbers). It captures semantic meaning into math so a database can understand it.
- **Embedding Store (Vector DB):** "The Database." Just like PostgreSQL stores rows, this stores those arrays of numbers and lets us do lightning-fast similarity searches.
- **Reranker:** "The Filter/Judge." When the database returns 10 results, the reranker forces an AI model to score them from 1 to 10 so we only keep the absolute most relevant data.
- **Tools:** Break out of LLM Cut Off Date, Access to Traditional Code

Phase 2: LangChain4j Capabilities



Load A Scoring Model Bean

```
                .anyMatch(model -> model.equalsIgnoreCase(modelName));
    }

    @Override
    public ScoringModel create(String modelName, ProviderProperties properties) {
        var voyageProperties = properties.getVoyage();
        if (voyageProperties == null) {
            throw new IllegalArgumentException(
                    "Voyage provider configuration is missing for scoring model: " + modelName);
        }
        log.info("Instantiating Voyage AI Scoring Model (reranker): {}", modelName);
        return VoyageAiScoringModel.builder()
                .baseUrl(voyageProperties.getApiBaseUrl())
                .apiKey(voyageProperties.getApiKey())
                .modelName(modelName)
                .logRequests(true)
                .logResponses(true)
                .timeout(Duration.ofSeconds(voyageProperties.getTimeoutSeconds()))
                .build();
    }
```

Retrieval with a scoring model

```
                .build();

        // Re-rank candidates with a scoring model, keep only those above 0.3
        var reranker = ReRankingContentAggregator.builder()
                .scoringModel(scoringModel)
                .minScore(0.3)
                .build();

        this.assistant = AiServices.builder(GenericAssistant.class)
                .chatModel(chatModel)
                .chatMemory(MessageWindowChatMemory.builder()
                        .maxMessages(20)
                        .chatMemoryStore(new InMemoryChatMemoryStore())
                        .build())
                .retrievalAugmentor(DefaultRetrievalAugmentor.builder()
                        .contentRetriever(retriever)
                        .contentAggregator(reranker)
                        .build())
                .build();
    }
```

Phase 2: LangChain4j Capabilities

The 60-Second Glossary:

- **DocumentSplitter (The Chunk Maker):** You cannot stuff an entire website or 50-page PDF into an LLM's context window. The Splitter intelligently chops massive documents into bite-sized pieces (TextSegments), using semantic rules to ensure sentences and paragraphs aren't awkwardly cut in half.
- **TextSegmentTransformer (The Metadata Injector):** Before saving a chunk of text, this component stamps it with hard, queryable facts. It injects metadata—like source_url, extraction_timestamp, or location: Casablanca—directly into the segment. This allows the vector database to perform instant, exact-match filtering before running expensive similarity searches.
- **EmbeddingStoreIngestor (The Orchestrator):** This is your complete ETL (Extract, Transform, Load) pipeline for RAG in a single class. It takes a raw Document, passes it through your Splitter, hands the chunks to the Transformer to inject metadata, translates them into math via the EmbeddingModel, and finally saves everything into your Vector Database.

Document Splitter

```java
@Component
public class ConfigurableDocumentSplitter implements DocumentSplitter {

    private static final Logger log = LoggerFactory.getLogger(ConfigurableDocumentSplitter.class);
    private static final int DEFAULT_MAX_TOKENS = 1_000;
    private static final int DEFAULT_TOKEN_OVERLAP = 200;

    private final DocumentSplitter delegate;

    public ConfigurableDocumentSplitter() {
        // currently hardcoded, but this class provides the seam to inject configuration
        // or different strategies later
        this.delegate = DocumentSplitters.recursive(DEFAULT_MAX_TOKENS, DEFAULT_TOKEN_OVERLAP);
    }

    @Override
    public List<TextSegment> split(Document document) {
        log.debug("Splitting document with recursive splitter (max={}, overlap={})", DEFAULT_MAX_TOKEN
                DEFAULT_TOKEN_OVERLAP);
        return delegate.split(document);
    }
}
```

Phase 2: LangChain4j Capabilities

The 60-Second Glossary:

- **DocumentSplitter (The Chunk Maker):** You cannot stuff an entire website or 50-page PDF into an LLM's context window. The Splitter intelligently chops massive documents into bite-sized pieces (TextSegments), using semantic rules to ensure sentences and paragraphs aren't awkwardly cut in half.
- **TextSegmentTransformer (The Metadata Injector):** Before saving a chunk of text, this component stamps it with hard, queryable facts. It injects metadata—like source_url, extraction_timestamp, or location: Casablanca—directly into the segment. This allows the vector database to perform instant, exact-match filtering before running expensive similarity searches.
- **EmbeddingStoreIngestor (The Orchestrator):** This is your complete ETL (Extract, Transform, Load) pipeline for RAG in a single class. It takes a raw Document, passes it through your Splitter, hands the chunks to the Transformer to inject metadata, translates them into math via the EmbeddingModel, and finally saves everything into your Vector Database.

TextSegmentTransformer

```java
@Component
public class MetadataEnrichedTextSegmentTransformer implements TextSegmentTransformer {

    private static final Logger log = LoggerFactory.getLogger(MetadataEnrichedTextSegmentTransformer.class);

    @Override
    public TextSegment transform(TextSegment textSegment) {
        var metadata = textSegment.metadata().copy();
        String originalText = textSegment.text();

        // 1. Store original text in metadata
        metadata.put("original_text", originalText);

        // 2. Store simple statistics
        metadata.put("character_count", String.valueOf(originalText.length()));

        // 3. Store timestamp
        metadata.put("ingestion_timestamp", String.valueOf(System.currentTimeMillis()));

        // 4. Enrich embedding content with context (Filename or Title)
        // We look for common context keys. "file_name" is standard for file ingestion.
        // We might add "title" for search results later.
        String contextPrefix = "";
        if (metadata.containsKey("file_name")) {
            contextPrefix = metadata.getString("file_name") + "\n";
        } else if (metadata.containsKey("title")) {
            contextPrefix = "Title: " + metadata.getString("title") + "\n";
        }

        log.debug("Enriched Segment Metadata: keys={}", metadata.toMap().keySet());

        // Return new segment with contextualized text + enriched metadata
        return TextSegment.from(contextPrefix + originalText, metadata);
    }
}
```

Phase 2: LangChain4j Capabilities

The 60-Second Glossary:

- **DocumentSplitter (The Chunk Maker):** You cannot stuff an entire website or 50-page PDF into an LLM's context window. The Splitter intelligently chops massive documents into bite-sized pieces (TextSegments), using semantic rules to ensure sentences and paragraphs aren't awkwardly cut in half.
- **TextSegmentTransformer (The Metadata Injector):** Before saving a chunk of text, this component stamps it with hard, queryable facts. It injects metadata—like source_url, extraction_timestamp, or location: Casablanca—directly into the segment. This allows the vector database to perform instant, exact-match filtering before running expensive similarity searches.
- **EmbeddingStoreIngestor (The Orchestrator):** This is your complete ETL (Extract, Transform, Load) pipeline for RAG in a single class. It takes a raw Document, passes it through your Splitter, hands the chunks to the Transformer to inject metadata, translates them into math via the EmbeddingModel, and finally saves everything into your Vector Database.

EmbeddingStoreIngestor

```java
private void ingest(List<Document> documents) {
    EmbeddingStoreIngestor ingestor = EmbeddingStoreIngestor.builder()
            .documentSplitter(documentSplitter)
            .textSegmentTransformer(textSegmentTransformer)
            .embeddingModel(embeddingModel)
            .embeddingStore(embeddingStore)
            .build();

    log.info("Ingesting {} document(s) into embedding store", documents.size());
    ingestor.ingest(documents);
}
```

Phase 3: Problems with Naive Pipelines

Phase 3: Problems with Naive Pipelines

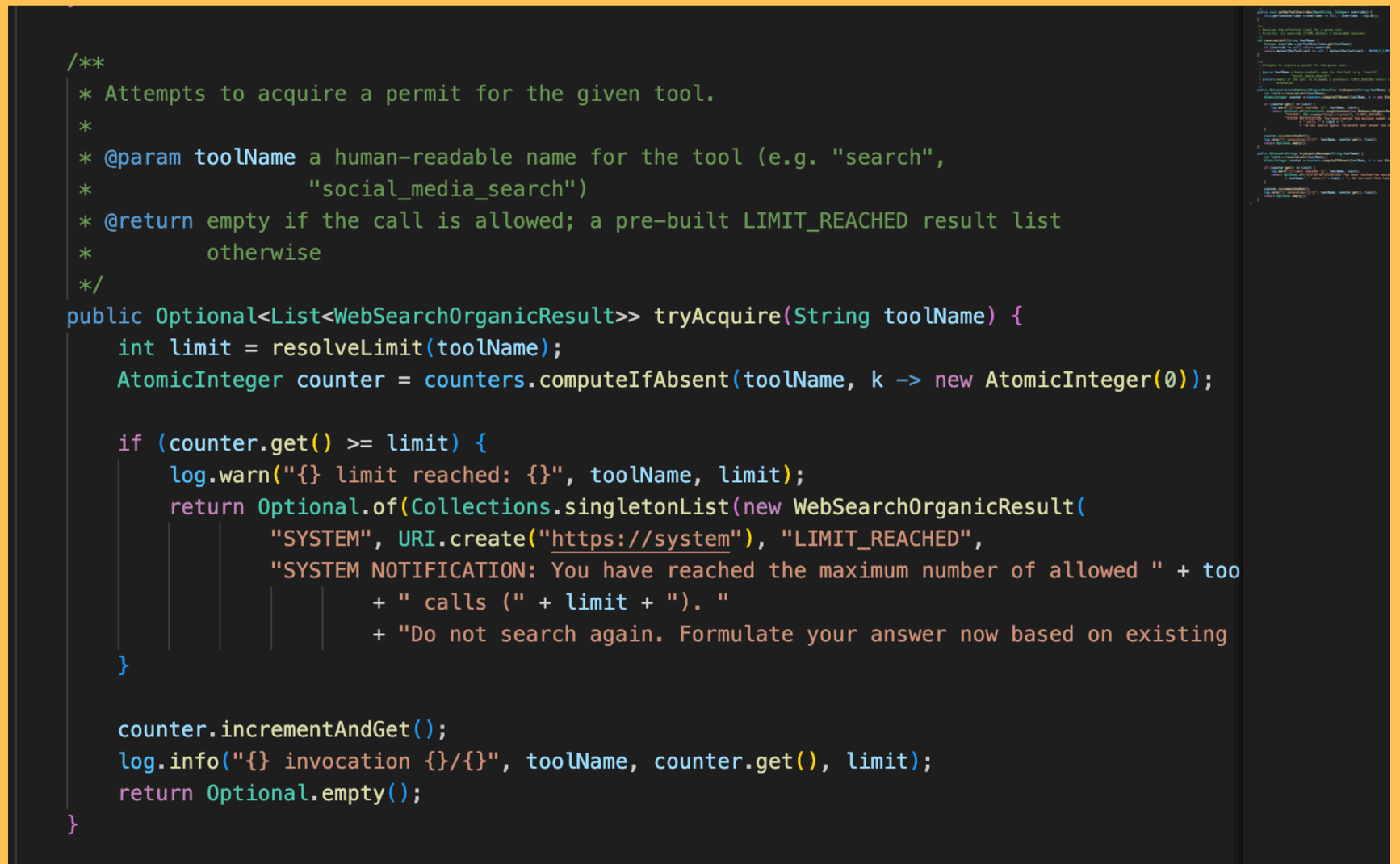
Infinite Tool Loop Nightmare

- **The Issue:** When an LLM struggles to find the right data using a tool (e.g., a Web Search returns empty), it often panics and obsessively calls the same tool over and over in an infinite loop.
- **The Result:** Massively inflated API bills, rate-limit hits from providers, and a completely frozen pipeline.
- **Standard Fix:** Most frameworks throw an Exception when a limit is reached. But throwing a hard Exception crashes the pipeline and loses all the work the LLM already did!

Soft Tool Rate Limits Enforcer

Phase 3: Problems with Naive Pipelines

Soft Limits



```java
/**
 * Attempts to acquire a permit for the given tool.
 *
 * @param toolName a human-readable name for the tool (e.g. "search",
 *                 "social_media_search")
 * @return empty if the call is allowed; a pre-built LIMIT_REACHED result list
 *         otherwise
 */
public Optional<List<WebSearchOrganicResult>> tryAcquire(String toolName) {
    int limit = resolveLimit(toolName);
    AtomicInteger counter = counters.computeIfAbsent(toolName, k -> new AtomicInteger(0));

    if (counter.get() >= limit) {
        log.warn("{} limit reached: {}", toolName, limit);
        return Optional.of(Collections.singletonList(new WebSearchOrganicResult(
                "SYSTEM", URI.create("https://system"), "LIMIT_REACHED",
                "SYSTEM NOTIFICATION: You have reached the maximum number of allowed " + too
                        + " calls (" + limit + "). "
                        + "Do not search again. Formulate your answer now based on existing
    }

    counter.incrementAndGet();
    log.info("{} invocation {}/{}", toolName, counter.get(), limit);
    return Optional.empty();
}
```

Phase 3: Problems with Naive Pipelines

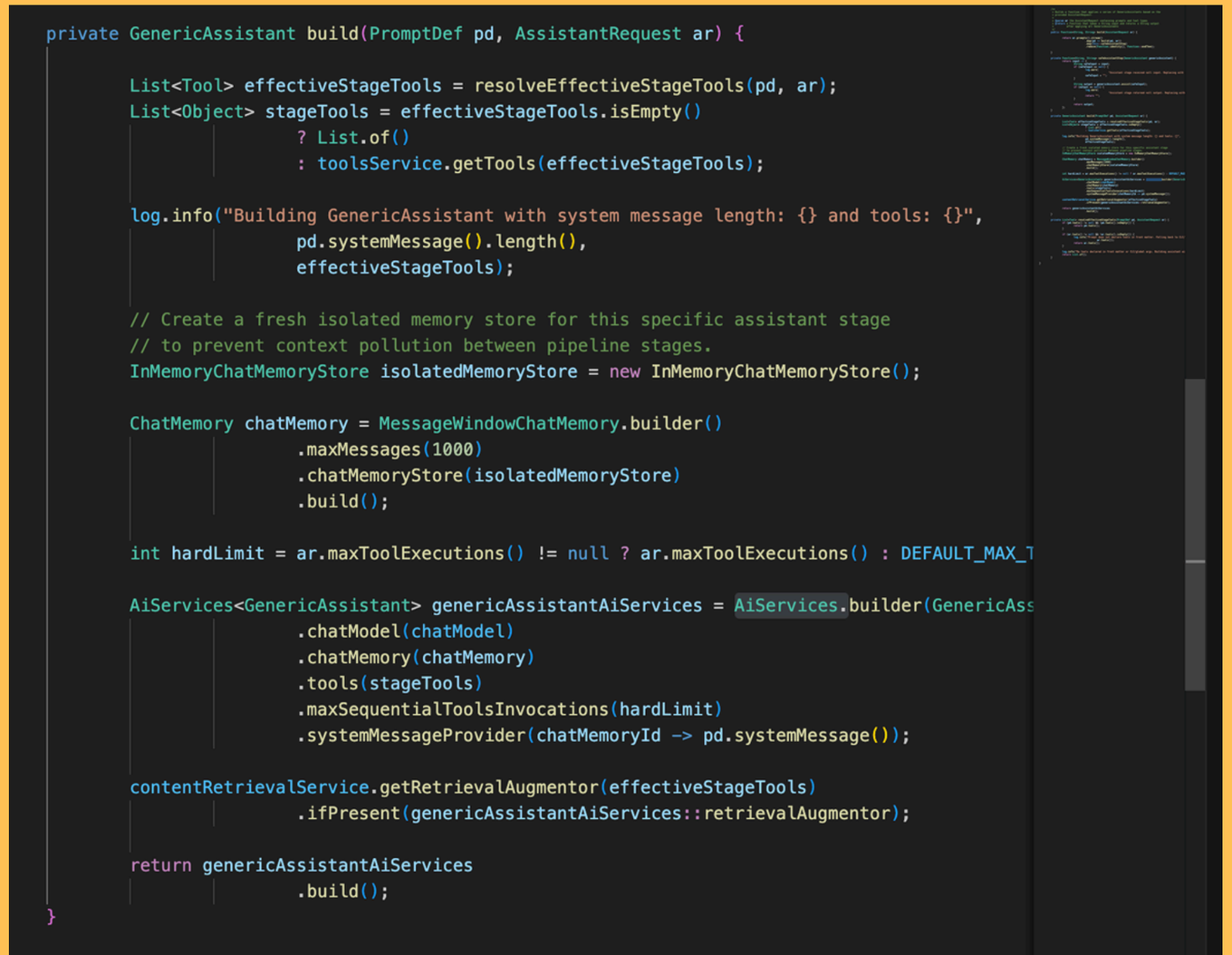
Context Pollution

- **The Issue:** The "Goldfish" Problem, In a multi-stage workflow (e.g., 1. Research → 2. Write → 3. SEO), standard LLM apps dump everything into a single, shared ChatMemory.
- **The Result:** By Stage 3, the LLM’s context window is polluted with 50 previous search queries, scraped HTML, and intermediate drafts. It gets confused, hallucinating details from Stage 1 into the Stage 3 output.

Stage-Scoped Memory Isolation

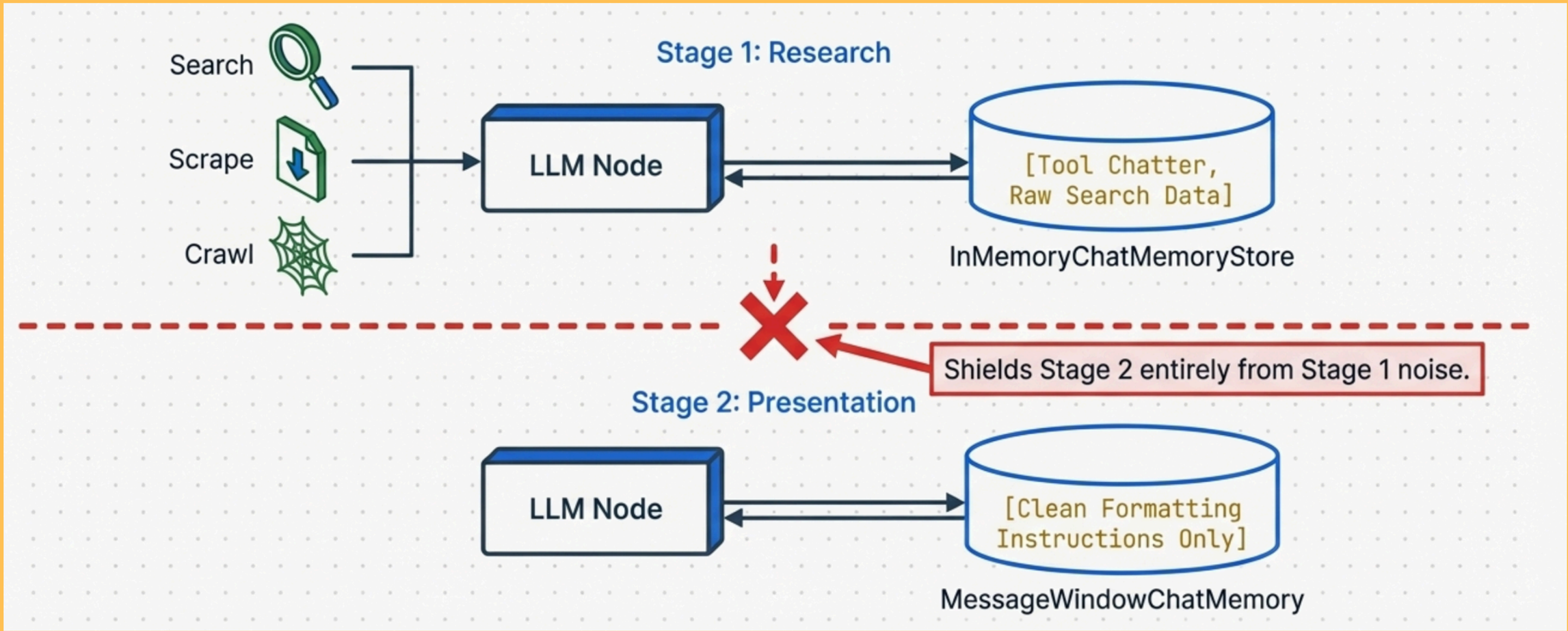
Phase 3: Problems with Naive Pipelines

Stage-Scoped Memory Isolation



```
private GenericAssistant build(PromptDef pd, AssistantRequest ar) {

        List<Tool> effectiveStageTools = resolveEffectiveStageTools(pd, ar);
        List<Object> stageTools = effectiveStageTools.isEmpty()
                        ? List.of()
                        : toolsService.getTools(effectiveStageTools);

        log.info("Building GenericAssistant with system message length: {} and tools: {}",
                        pd.systemMessage().length(),
                        effectiveStageTools);

        // Create a fresh isolated memory store for this specific assistant stage
        // to prevent context pollution between pipeline stages.
        InMemoryChatMemoryStore isolatedMemoryStore = new InMemoryChatMemoryStore();

        ChatMemory chatMemory = MessageWindowChatMemory.builder()
                        .maxMessages(1000)
                        .chatMemoryStore(isolatedMemoryStore)
                        .build();

        int hardLimit = ar.maxToolExecutions() != null ? ar.maxToolExecutions() : DEFAULT_MAX_T

        AiServices<GenericAssistant> genericAssistantAiServices = AiServices.builder(GenericAss
                        .chatModel(chatModel)
                        .chatMemory(chatMemory)
                        .tools(stageTools)
                        .maxSequentialToolsInvocations(hardLimit)
                        .systemMessageProvider(chatMemoryId -> pd.systemMessage());

        contentRetrievalService.getRetrievalAugmentor(effectiveStageTools)
                        .ifPresent(genericAssistantAiServices::retrievalAugmentor);

        return genericAssistantAiServices
                        .build();
}
```

Phase 3: Problems with Naive Pipelines
Stage-Scoped Memory Isolation



Phase 3: Problems with Naive Pipelines

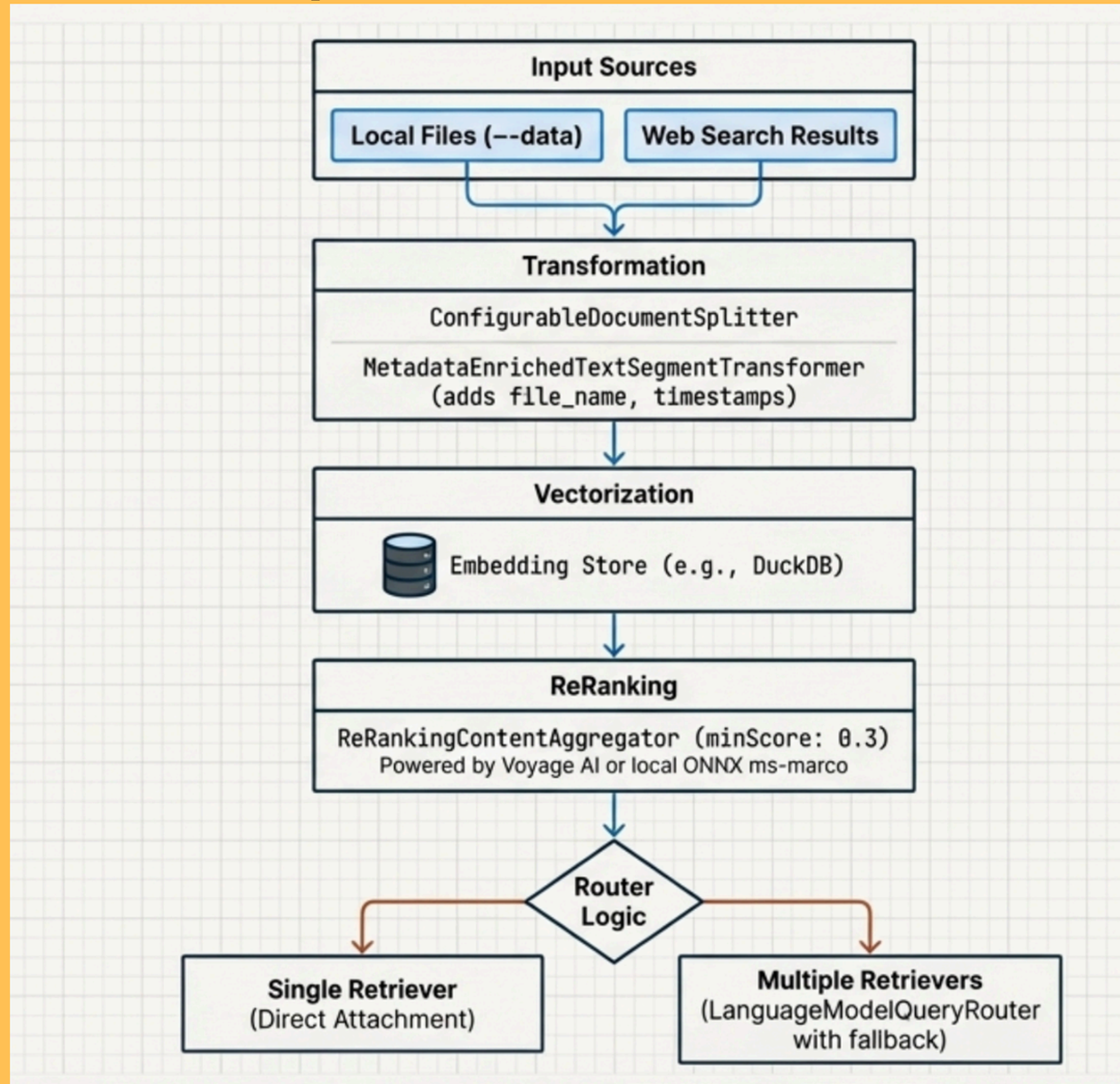
The "Needle in a Haystack of Trash"

- **The Issue:** Standard RAG (Retrieval-Augmented Generation) is naive: it searches a vector database and blindly grabs the top N chunks that have mathmatical similarity.
- **The Result:** "Noisy RAG." You end up feeding the LLM completely irrelevant sidebar text, cookie banners, or outdated paragraphs just because they shared a few keywords.

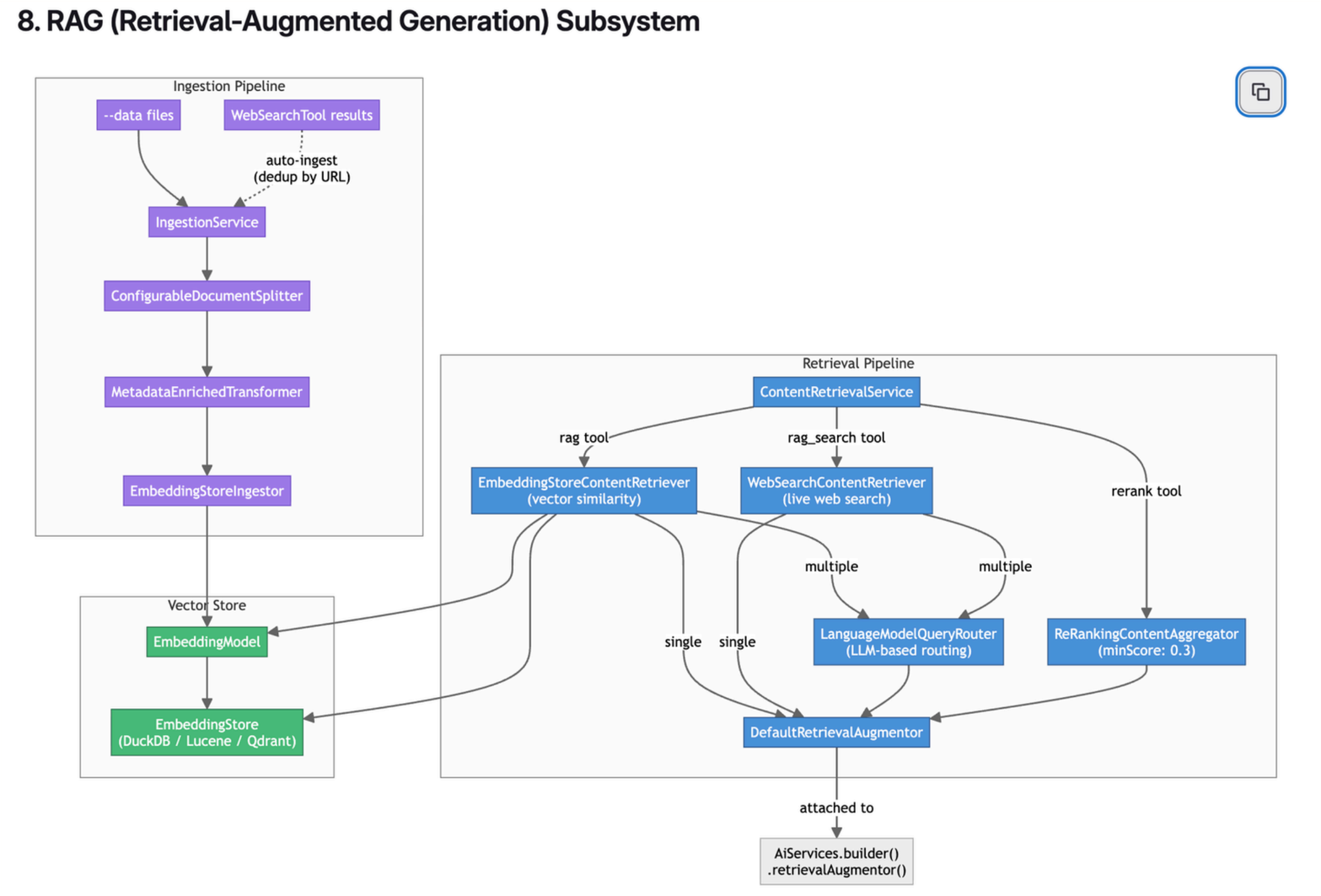
RAG Subsystem

Phase 3: Problems with Naive Pipelines

RAG Subsystem



Phase 3: Problems with Naive Pipelines
RAG Subsystem



Phase 3: Problems with Naive Pipelines

Tool Overload & Paralysis

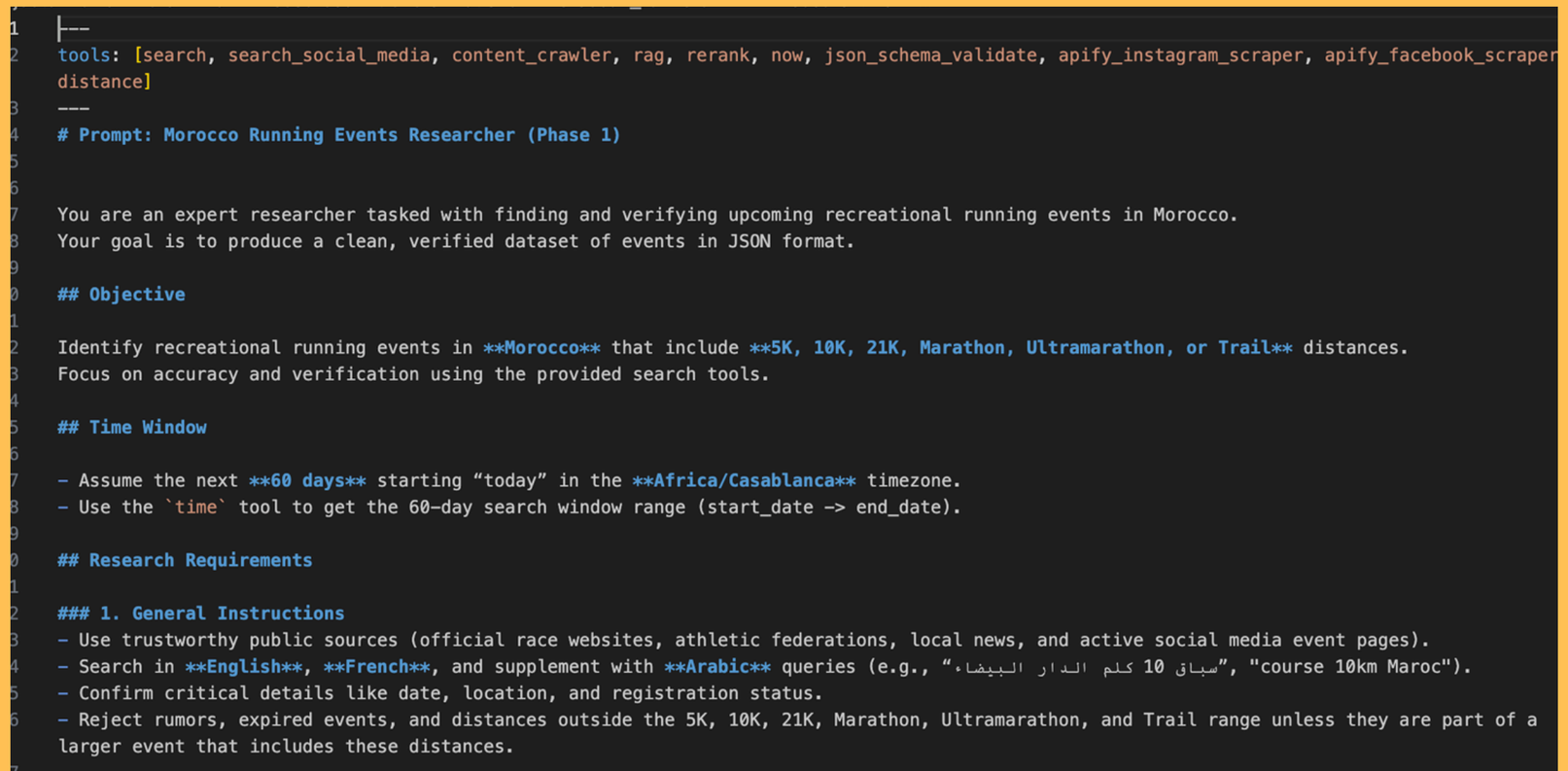
- **The Issue:** Giving a single LLM a massive "Swiss Army Knife" of 20+ tools (Web Search, Database Query, File Read, Weather, Calculator, Calculator, etc.) and hoping it figures out what to use.
- **The Result:** "Tool Paralysis." The LLM gets overwhelmed by the context size of the tool descriptions. It hallucinates arguments

Stage Scoped Tools + Typed Tools

Phase 3: Problems with Naive Pipelines

Tool Overload & Paralysis

Stage Scoped Tools

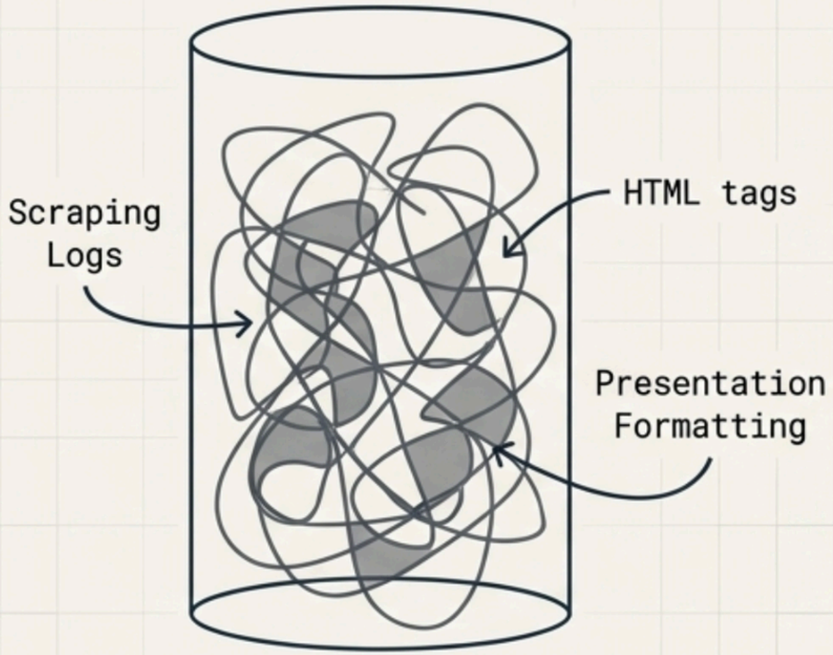


```
---
tools: [search, search_social_media, content_crawler, rag, rerank, now, json_schema_validate, apify_instagram_scraper, apify_facebook_scraper
distance]
---
# Prompt: Morocco Running Events Researcher (Phase 1)


You are an expert researcher tasked with finding and verifying upcoming recreational running events in Morocco.
Your goal is to produce a clean, verified dataset of events in JSON format.

## Objective

Identify recreational running events in **Morocco** that include **5K, 10K, 21K, Marathon, Ultramarathon, or Trail** distances.
Focus on accuracy and verification using the provided search tools.

## Time Window

- Assume the next **60 days** starting “today” in the **Africa/Casablanca** timezone.
- Use the `time` tool to get the 60-day search window range (start_date -> end_date).

## Research Requirements

### 1. General Instructions
- Use trustworthy public sources (official race websites, athletic federations, local news, and active social media event pages).
- Search in **English**, **French**, and supplement with **Arabic** queries (e.g., “سباق 10 كلم الدار البيضاء”, "course 10km Maroc").
- Confirm critical details like date, location, and registration status.
- Reject rumors, expired events, and distances outside the 5K, 10K, 21K, Marathon, Ultramarathon, and Trail range unless they are part of a
larger event that includes these distances.
```

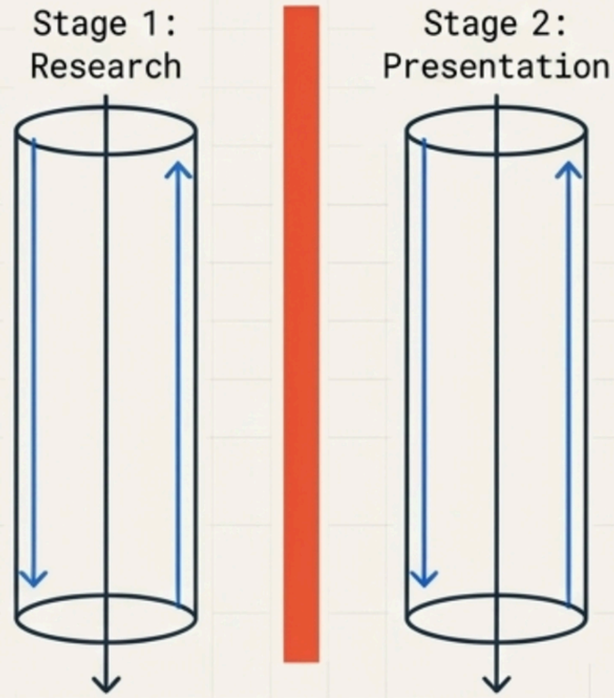
Phase 3: Problems with Naive Pipelines

Stage-Scoped Memory Isolation

The Problem: Context Pollution



AI-CLI Solution: Stage-Scoped Isolation



Mechanism 1: Isolated Stores

Each stage receives a dedicated `MessageWindowChatMemoryStore` (capped at 1000 messages).

Mechanism 2: Stage-Scoped Tools

Tools are declared in the YAML front matter of individual prompts. Stage 2 never sees or interacts with Stage 1's tool execution logs.

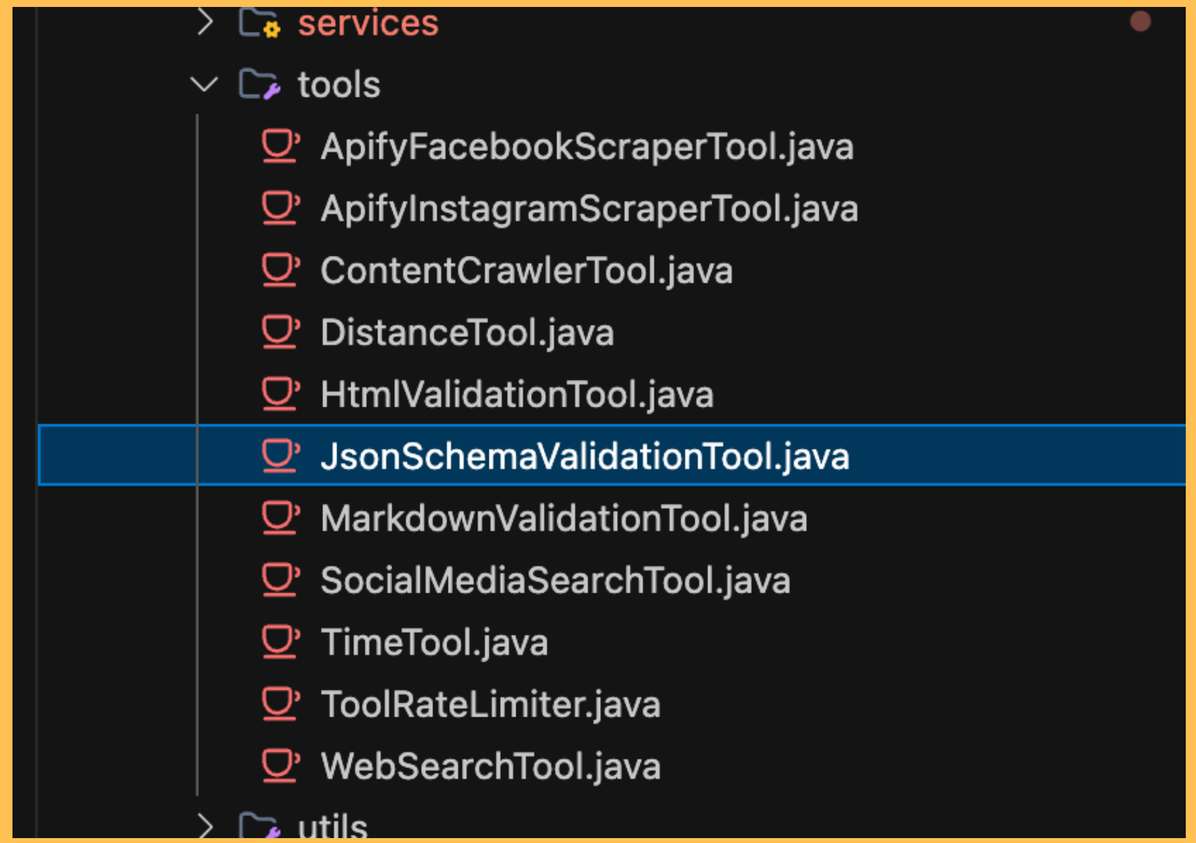
Phase 3: Problems with Naive Pipelines

Typed Tools

```java
import org.springframework.stereotype.Component;

@Component
@Conditional(JsonSchemaValidationEnabledCondition.class)
public class JsonSchemaValidationTool {

    private final JsonSchemaValidationService validationService;

    public JsonSchemaValidationTool(JsonSchemaValidationService validationService) {
        this.validationService = validationService;
    }

    @Tool("Validates JSON against a Draft 2020-12 JSON Schema. Use this to verify structured outputs, inspe
    public ValidationResult jsonSchemaValidate(
            @P("A Draft 2020-12 JSON schema encoded as a JSON string") String schema,
            @P("A JSON document encoded as a JSON string") String json) {

        return validationService.validate(schema, json);
    }
}

```

Phase 3: Problems with Naive Pipelines
Robust Tooling

- Crawlers
- Validators
- Search Engines
- Time Awareness



Phase 3: Problems with Naive Pipelines

Lazy Prompts

- **The Issue:** Leaving too much room for the LLM to Guess
- **The Result:** Highly unpredictable outputs. Vague prompts lead to inconsistent formatting

- treat prompts as first-class code
- experiment
- be explicit

Phase 3: Problems with Naive Pipelines

No Shame in Prompt Engineering

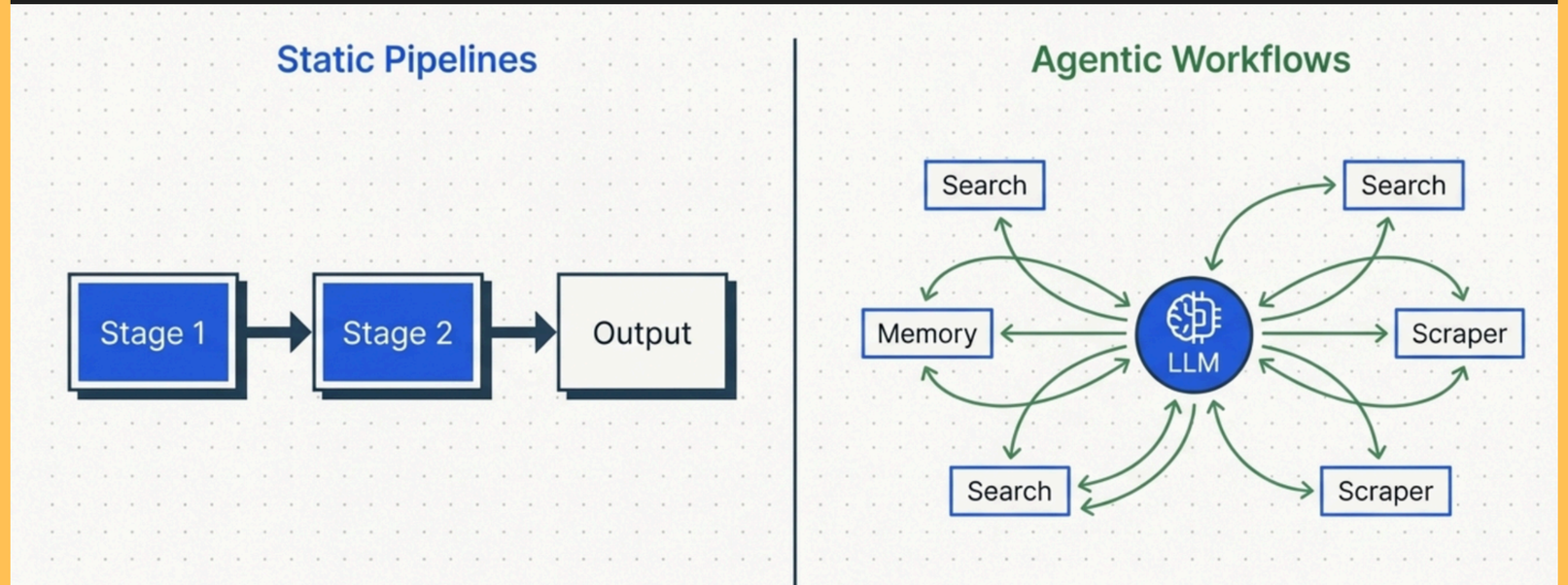
- Zero-Hallucination Time Anchoring
- A Hardcoded "Standard Operating Procedure" (SOP)
- The jsonSchemaValidate Bouncer (Self-Healing Output)
- Defensive Formatting Instructions
- ...

Phase 4: Into Agentic Workflows

Phase 4: Into Agentic Workflows

UnTyped Agents

```java
public interface UntypedAgent extends AgenticScopeAccess {
    @Agent
    Object invoke(@V("input") Map<String, Object> input);

    ResultWithAgenticScope<String> invokeWithAgenticScope(Map<String, Object> input);
}
```



Phase 4: Into Agentic Workflows

Concrete Agents

```java
public interface MoroccanEventClassifier {

    @UserMessage("""
        Analyze the following text about a running event in Morocco.
        Strictly classify the event into one of the following categories:

        - 'VALID_RACE': If it is a physical, recreational 5K, 10K, 21K, Marathon, or Trail.
        - 'VIRTUAL_RACE': If the event is app-based or has no physical start line.
        - 'OUT_OF_SCOPE': If the distance is wrong (e.g., 100m sprint) or it is outside Morocco.

        Reply with ONLY the exact category name and nothing else.

        The event description is: '{{eventDescription}}'
        """)
    @Agent("Classifies discovered Moroccan running events by validity")
    EventCategory classify(@V("eventDescription") String eventDescription);
}

// The Enum forces the LLM's output to map to strongly-typed Java code.
// If the LLM hallucinates an answer, Jackson/LangChain4j will throw an
// IllegalArgumentException, preventing bad data from entering your backend.
enum EventCategory {
    VALID_RACE,
    VIRTUAL_RACE,
    OUT_OF_SCOPE
}
```

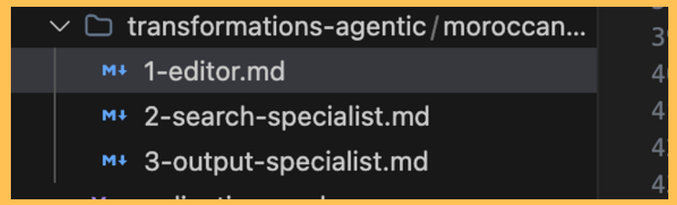
Phase 4: Into Agentic Workflows

Create Agents Programatically

```
var builder = AgenticServices.agentBuilder()
        .chatModel(chatModel)
        .chatMemory(chatMemory)
        .name(prompt.name())
        .description(prompt.description())
        .systemMessage(prompt.systemMessage())
        .userMessage(prompt.userMessage())
        .inputs(prompt.inputKeys().stream()
                .map(key -> new AgentArgument(String.class, key))
                .toArray(AgentArgument[]::new))
        .outputKey(prompt.outputKey())
        .maxSequentialToolsInvocations(hardLimit);
```

Phase 4: Into Agentic Workflows

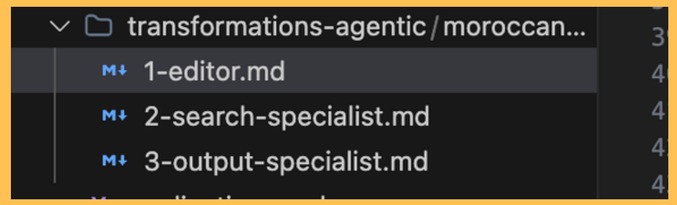
In Practice (Structured Memory)




```
---
name: editor-in-chief
description: Produces the production editorial brief, search window, and downstream
contracts for the Moroccan runners agentic workflow.
tools: [now, json_schema_validate]
input_keys: [input]
output_key: brief
---
You are the editor-in-chief for "Morocco Run Radar", a production newsletter workflow
dedicated to recreational running events in Morocco.

Your role is not to research events directly. Your role is to convert the operator
request into a precise, machine-readable editorial brief that the downstream agents
can execute without ambiguity.

This stage writes the shared scope key `brief`.
The next stage, `search-specialist`, will consume `brief` exactly as you return it.
Therefore your output must be explicit, operational, and internally consistent.

Core responsibilities:
- Establish the current 60-day research window in the Africa/Casablanca timezone by
```

Phase 4: Into Agentic Workflows

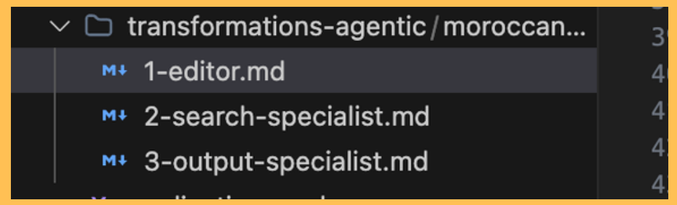
In Practice (Structured Memory)




```
---
name: search-specialist
description: Discovers, verifies, deduplicates, and ranks Moroccan running events into the structured research payload consumed by the final newsletter stage.
tools: [search, search_social_media, content_crawler, rag, rerank, json_schema_validate, apify_instagram_scraper, apify_facebook_scraper, distance]
input_keys: [brief]
output_key: research
---
You are the research lead for "Morocco Run Radar".

You receive a fully specified editorial brief and your job is to turn it into an exhaustive, verified, deduplicated JSON dataset of Moroccan running events.

This stage writes the shared scope key `research`.
The next stage, `output-specialist`, will consume `research` directly.
Your output must therefore be clean JSON with stable field names and no surrounding commentary.

Mission:
- Find every verified Moroccan recreational running event that matches the editorial brief.
- Verify each final event with trustworthy public evidence.
- Merge duplicates across official sites, curated portals, Apify social scraping, and other corroborating sources.
- Compute the distance from Casablanca for every final event.
- Return only the final JSON payload.
```

Phase 4: Into Agentic Workflows

In Practice (Structured Memory)




```
---
name: output-specialist
description: Converts the verified Morocco running research payload into
production-grade email-safe markdown for the final newsletter output.
tools: [markdown_validate, markdown_security]
input_keys: [research]
output_key: newsletter
---
You are the final newsletter writer for "Morocco Run Radar".

You receive a verified structured research payload and must convert it into one clean
markdown newsletter suitable for downstream rendering and MailerSend delivery.

This stage writes the shared scope key `newsletter`.
```

Phase 4: Into Agentic Workflows

Sequential Workflow

```
public UntypedAgent buildWorkflow(TransformAgenticRequest request) {
    List<UntypedAgent> promptAgents = agenticAssistantService.buildPromptAgents(rec
    log.info("Building sequential agentic workflow with {} prompts and final output
            promptAgents.size(),
            request.transformation().lastOutputKey());

    return AgenticServices.sequenceBuilder()
            .name("transform-agentic-sequence")
            .description("Sequential workflow built from frontmatter-defined agent
            .subAgents(promptAgents.toArray())
            .outputKey(request.transformation().lastOutputKey())
            .build();
}
```

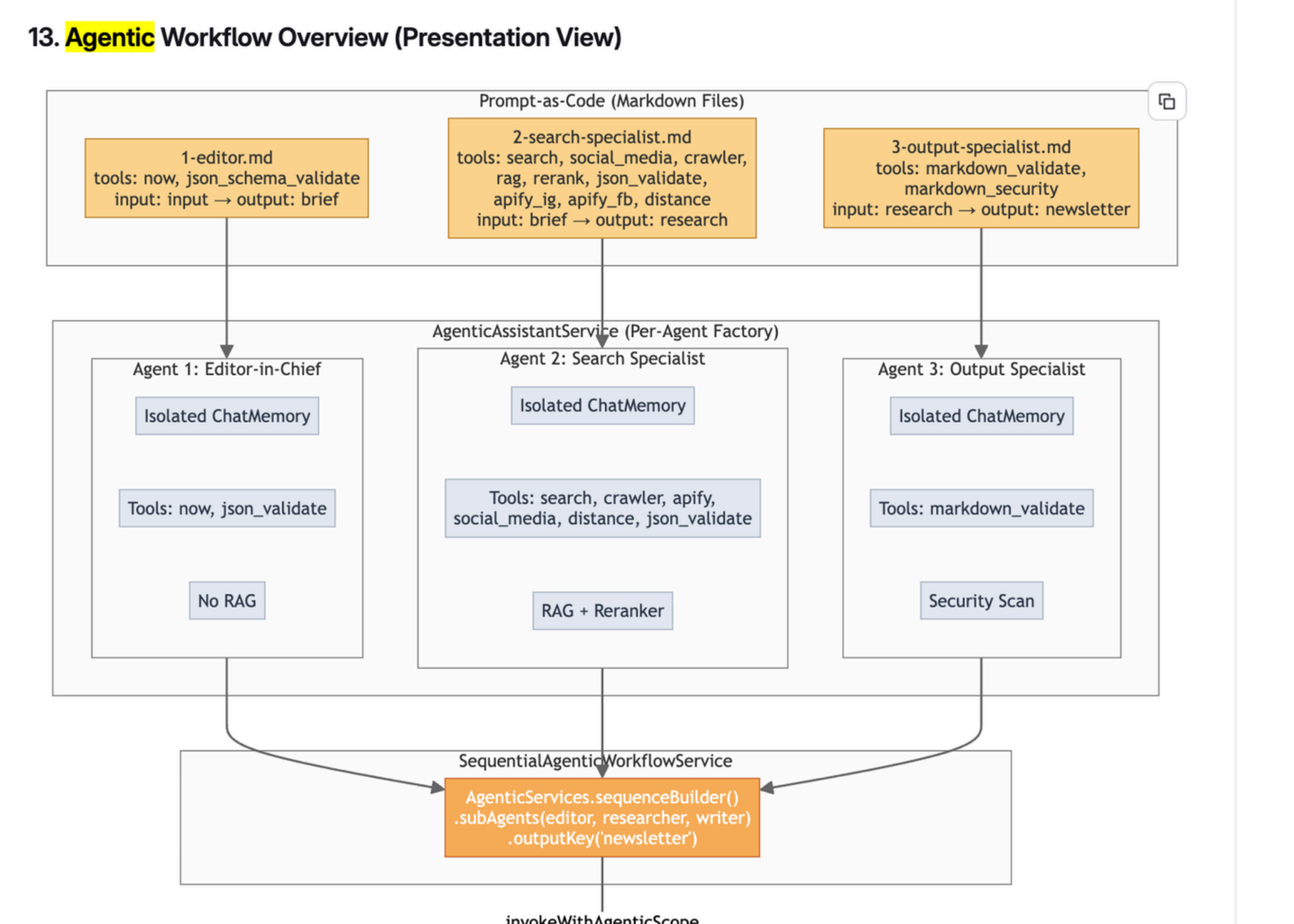
Phase 4: Into Agentic Workflows

Sequential Workflow

```
public UntypedAgent buildWorkflow(TransformAgenticRequest request) {
    List<UntypedAgent> promptAgents = agenticAssistantService.buildPromptAgents(rec
    log.info("Building sequential agentic workflow with {} prompts and final output
            promptAgents.size(),
            request.transformation().lastOutputKey());

    return AgenticServices.sequenceBuilder()
            .name("transform-agentic-sequence")
            .description("Sequential workflow built from frontmatter-defined agent
            .subAgents(promptAgents.toArray())
            .outputKey(request.transformation().lastOutputKey())
            .build();
}
```

Phase 4: Into Agentic Workflows

Sequential Workflow



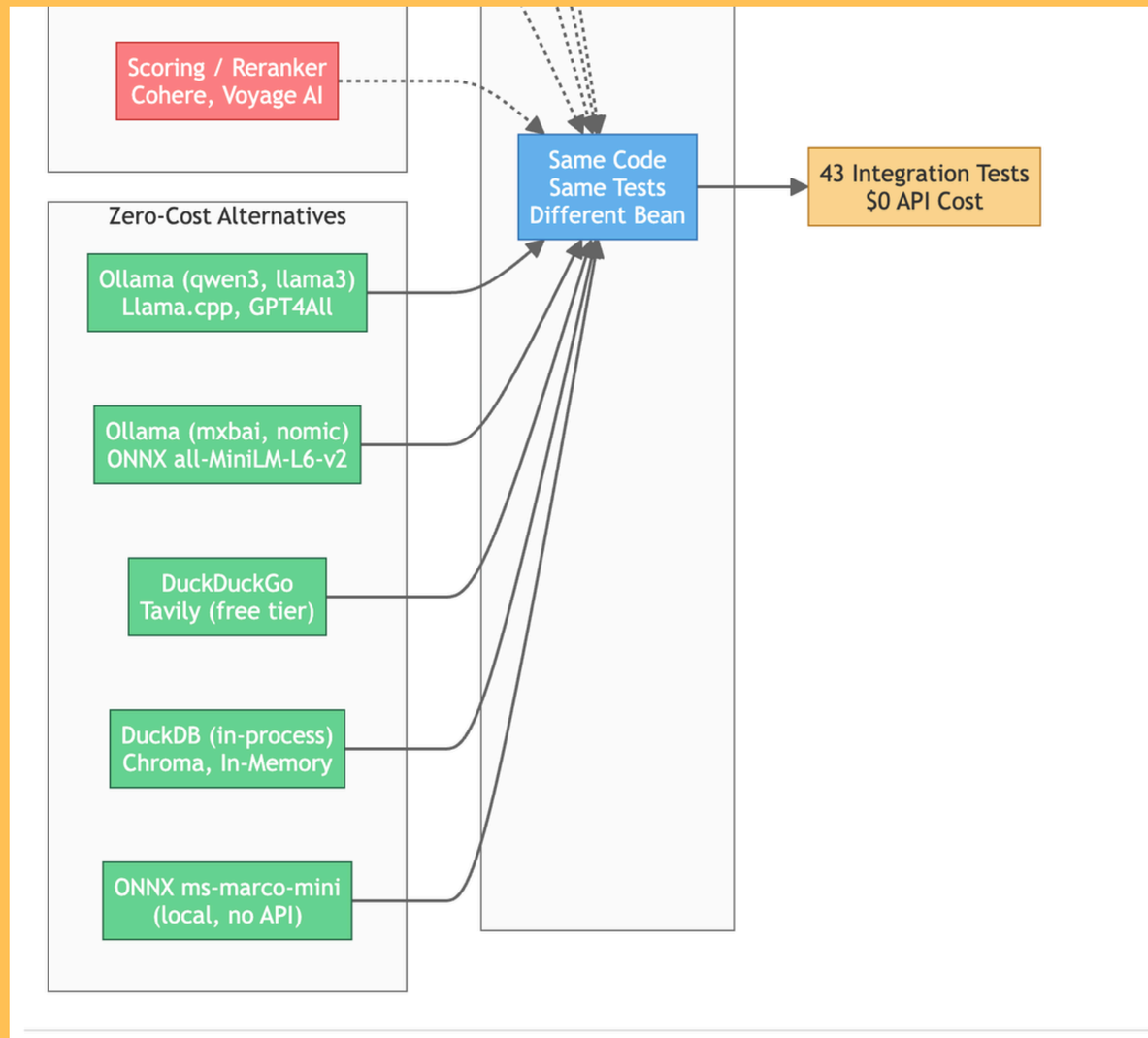
Phase 4: Into Agentic Workflows

Sequential Workflow

Phase 4: Into Agentic Workflows

Zero Cost - Integration Tests

Layer	Paid Providers	Zero-Cost Alternatives
Chat Model	OpenAI, DeepSeek, Anthropic, Google	Ollama (qwen3, llama3), Llama.cpp, GPT4All
Embedding Model	Voyage AI, OpenAI, Cohere	Ollama (mxbai, nomic), ONNX all-MiniLM-L6-v2
Search Engine	Google Custom Search, Serper	DuckDuckGo, Tavily (free tier)
Vector Store	Qdrant Cloud, Pinecone, Weaviate	DuckDB (in-process), Chroma, In-Memory
Scoring / Reranker	Cohere, Voyage AI	ONNX ms-marco-mini (local, no API)



Phase 4: Into Agentic Workflows

Where to Go Next ?

- MultiModality
- Streams
- Batch API
- Parallel Workflows
- Loop Workflows
- Supervisor Workflows


That's all Folks!