Spring-AI Intro

Kim Horn

Version 1.0 **22 March 2025**

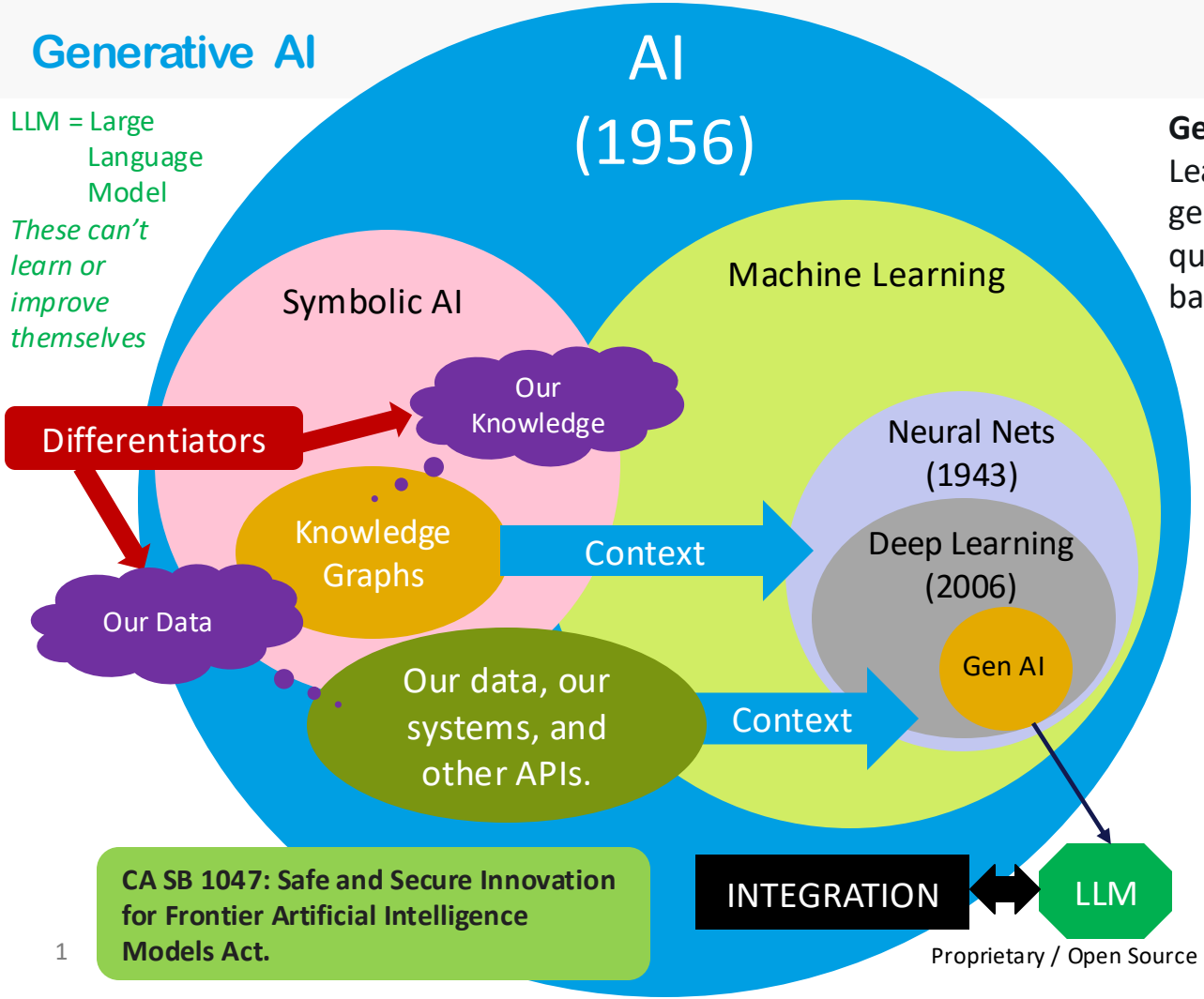
Note: the slides when presented were animated, these aminations are lost in the PDF

Generative AI

INFOMEDIA™

LLM = Large Language Model

These can't learn or improve themselves



Generative AI (GenAI) refers to Deep Learning models (LLMs) that can generate coherent human like high-quality text, images and other content based on the data they were trained on.

There are many issues:

- don't know about our business, our domain, where's the differentiation
- not trained on our data
- training data is incomplete
- trained at a past date
- input context size is limited
- not connected to the world, e.g. time
- stateless, no memory
- no structured output (e.g. Java Object)
- hallucinate, indecisive, can give biased, immoral, unethical and illegal advice
- hard to test and validate, catch 22
- security, privacy, and cost issues
- legal problems, different Laws, Standards for Safety First and Responsible AI, in every country.

CA SB 1047: Safe and Secure Innovation for Frontier Artificial Intelligence Models Act.

Spring AI

Spring AI is an application framework for AI engineering. At its core, Spring AI addresses the fundamental challenge of AI integration: Connecting your enterprise Data and APIs with the AI Models.

Spring AI is a new framework. The project was started back in 2023. The first publicly available version, the 0.8, was released in February 2024. Now its at 1.0.0.M6. The 1st release is coming….

I could never find good working examples.

In Feb 2024 I started to build a set of 40+ Examples, off Snapshot build……

Microsoft

Azure OpenAI

Java is one of the top languages used with Azure OpenAI

MICROSOFT JDCONF, 2024

Gen AI app

App framework

Firebase spring

LLM

Gemini

Transactional database

AlloyDB

Scalable infrastructure

GKE

OpenAI Specific HTTP JSON

POST https://api.openai.com/v1/chat/completions

```
{
   "model": "gpt-4o",
   "messages": [
   {
      "role": "system",
      "content": "You are a helpful polite assistant."
    },
    {
      "role": "user",
      "content": "What is a dog ?"
   } ]
 }
```

Request

Try This:

"role": "system",

"content": "You are a bad pirate."

```
{
   "id": "chatcmpl-A8j01LYAMpbOdH0VL1lscVjjbuJ78",
   "object": "chat.completion",
   "created": 1726642965,
   "model": "gpt-4o-2024-05-13",
   "choices": [
     {
       "index": 0,
       "message": {
         "role": "
                                    mesticated mammal and a subspecies of grey wolf.
                                    is lupus familiaris……..”
                    ctusal": null
       },
       "logprobs": null,
       "finish_reason": "stop"
     } ],
   "usage": {
     "prompt_tokens": 23,
     "completion_tokens": 11,
     "total_tokens": 34,
     "completion_tokens_details": {
       "reasoning_tokens": 0
     }
   },
   "system_fingerprint": "fp_a5d11b2ef2"
}
```

Response

Every LLM vendor has a different API !!!

ChatClient Vs ChatModel – one API across LLM Providers

INFOMEDIA™

The prompt, our user's **message = "What is a dog ?"**

ChatClient (fluent)

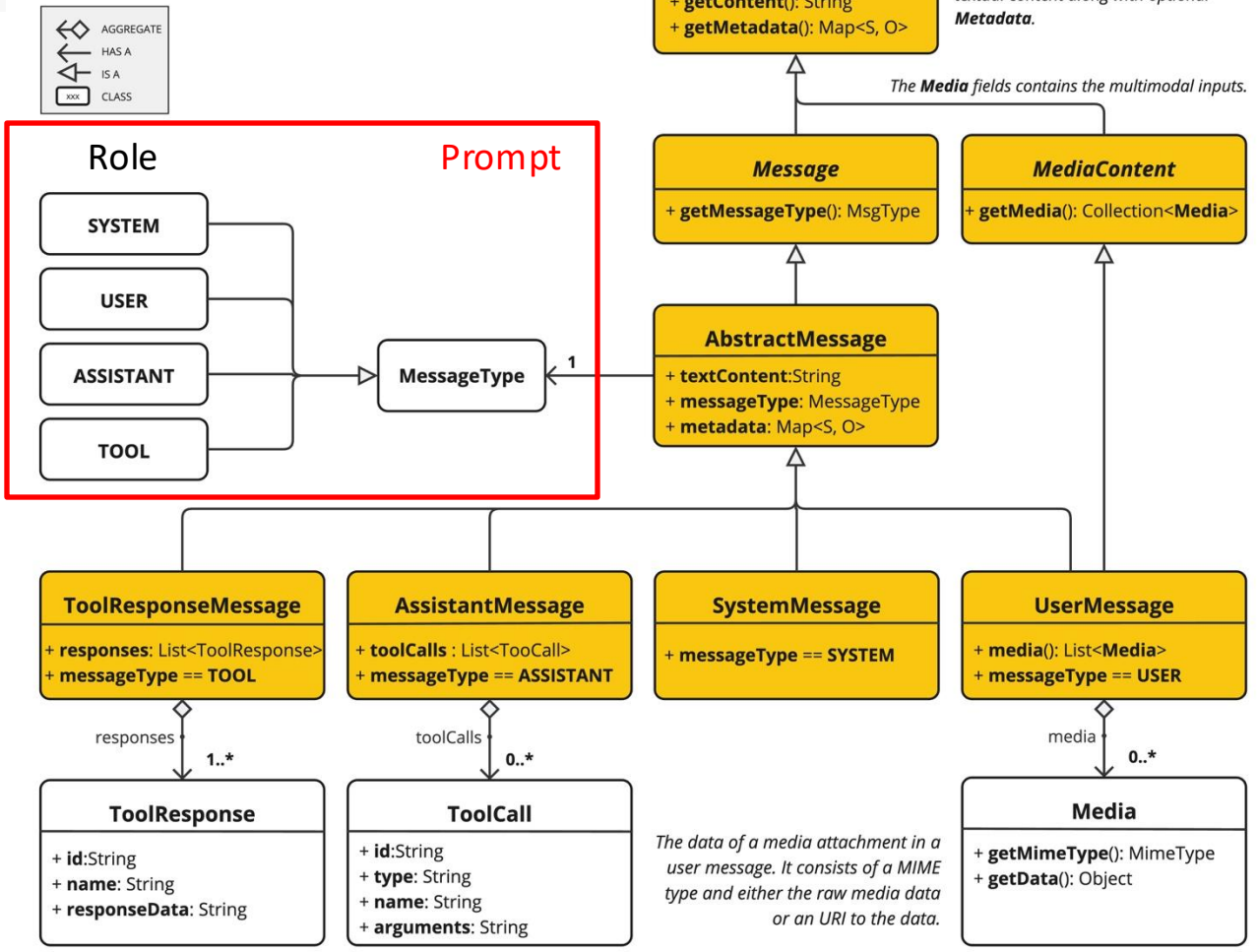
```
ChatClient chatClient = chatClientBuilder.build();

String answer = chatClient
        .prompt()
        .user(message)
        .call()
        .content();
```

ChatModel (lower level)

```
OpenAiChatModel chatModel;

ChatResponse response =
        chatModel.call(new Prompt(message));

String answer = response.getResult()
                .getOutput()
                .getText();
```

- Our Open AI key is automatically taken from our environment variables
- Pom specifies the specific vendor api: *spring-ai-openai-spring-boot-starter*

Spring AI Message API



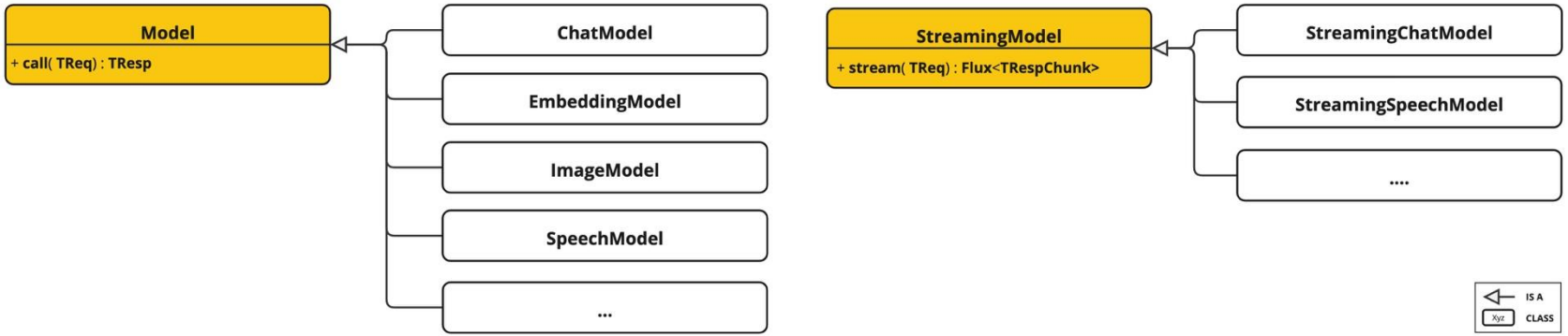
The Message interface encapsulates a **Prompt** textual content, a collection of metadata attributes, and a ***MessageType***.

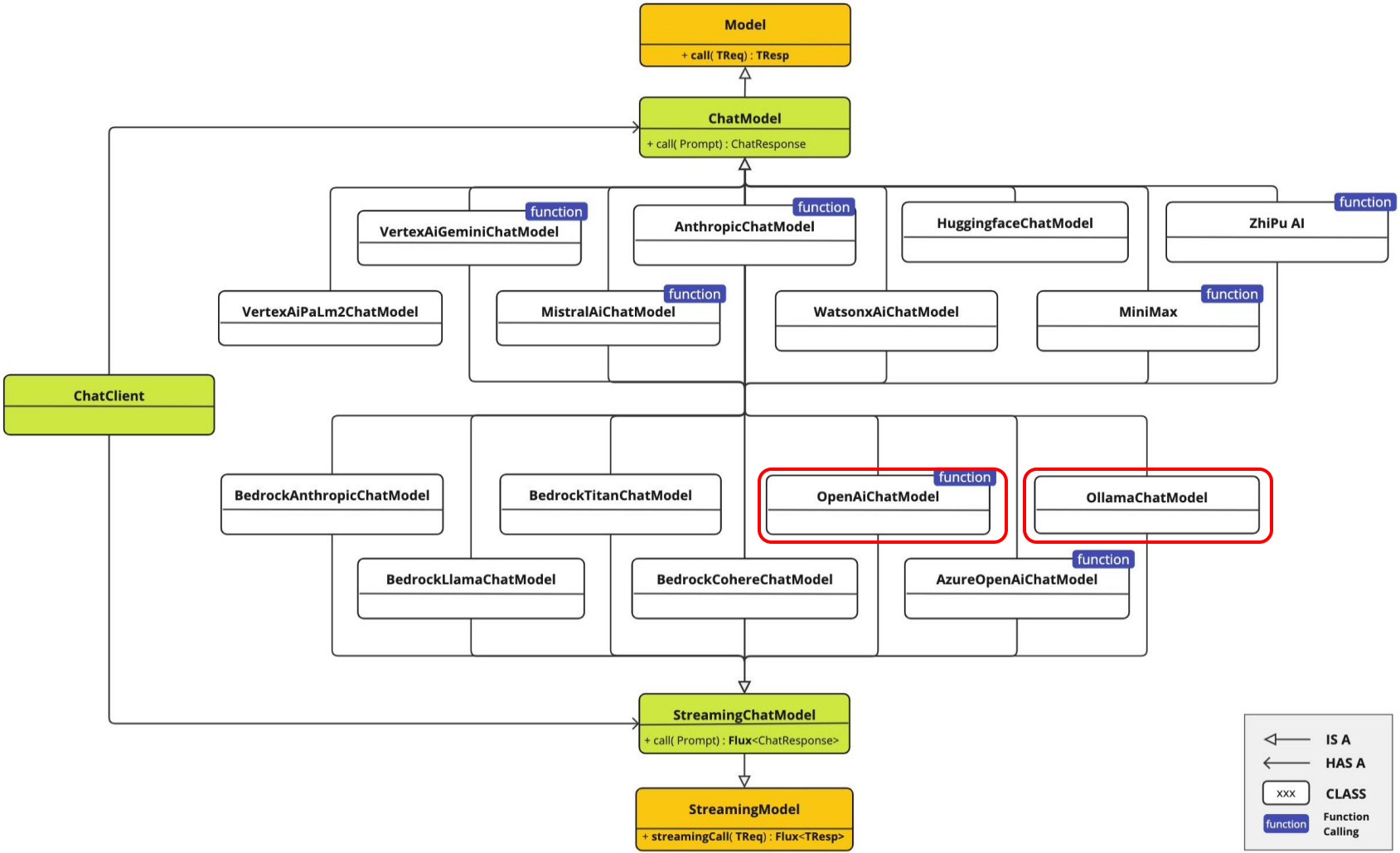
The collection specifies the **instructions** to the LLM.

```
public class Prompt implements
ModelRequest<List<Message>>
{
        private final List<Message> messages;
        private ChatOptions chatOptions;
}

public interface Content {
        String getContent();
        Map<String, Object> getMetadata();
}

public interface Message extends Content {
        MessageType getMessageType();
}
```

Spring Model Types

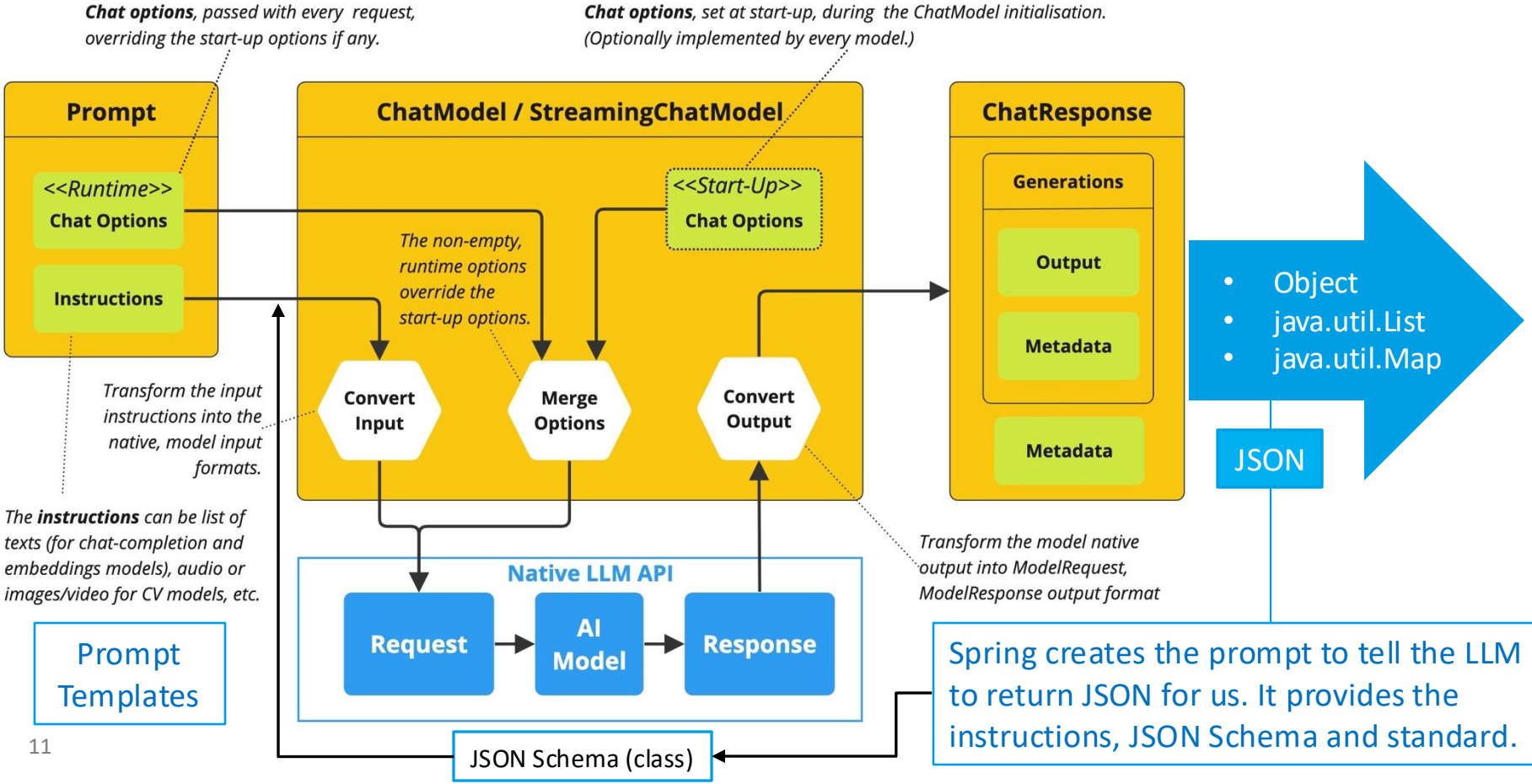
Client	Description
ChatClient / ChatModel	The core interface for text-based interactions. This is used for simple requests, prompt-based requests, and requests requiring a response in a specific format (such as JSON, XML etc).
ImageModel	A client for calling the vendor-specific image generation API. When we send some text as a request using this client, we get the URL of the generated image in the response.
SpeechModel / StreamingSpeechModel	A client for calling the vendor-specific *text-to-speech* generation API (TTS-1). When we send some text as a request using this client, we get the speech file path in the response.
AudioTranscriptionModel	A client for calling the vendor-specific audio transcription (*speech-to-text*) models. Currently, only OpenAI's '*whisper-1*' is supported.
EmbeddingModel	A client for calling the vendor-specific vector embedding models, commonly used for making similarity searches in vector stores.




Model
+ call(TReq) : TResp
ChatModel
+ call(Prompt) : ChatResponse
function
VertexAiGeminiChatModel
function
AnthropicChatModel
HuggingfaceChatModel
function
ZhiPu AI
VertexAiPaLm2ChatModel
function
MistralAiChatModel
WatsonxAiChatModel
function
MiniMax
ChatClient
BedrockAnthropicChatModel
BedrockTitanChatModel
function
OpenAiChatModel
OllamaChatModel
BedrockLlamaChatModel
BedrockCohereChatModel
function
AzureOpenAiChatModel
StreamingChatModel
+ call(Prompt) : Flux<ChatResponse>
StreamingModel
+ streamingCall(TReq) : Flux<TResp>
IS A
HAS A
xxx
CLASS
function
Function Calling

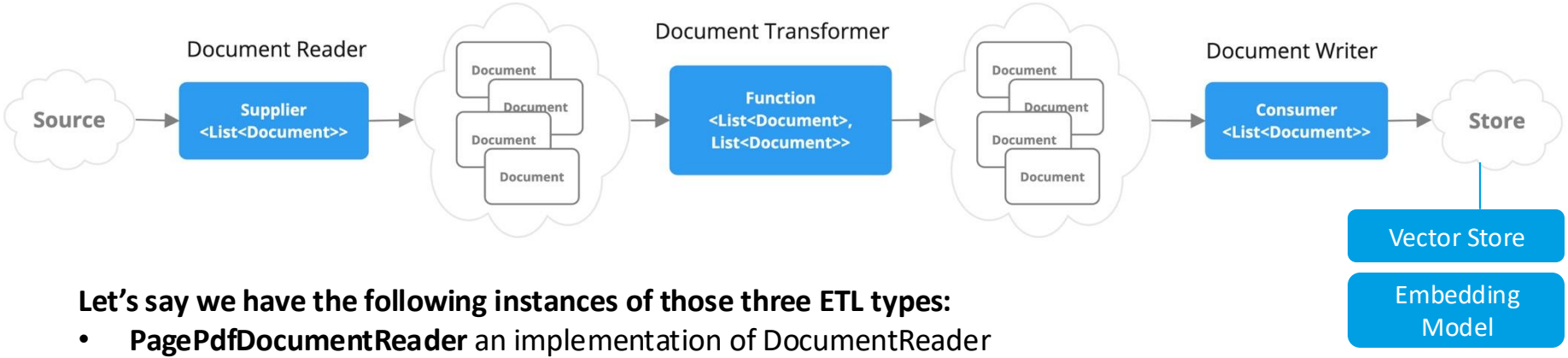
Components

Converters – Input, Output



ETL Pipelines

How to we get our information (data, documents), into LLM ?

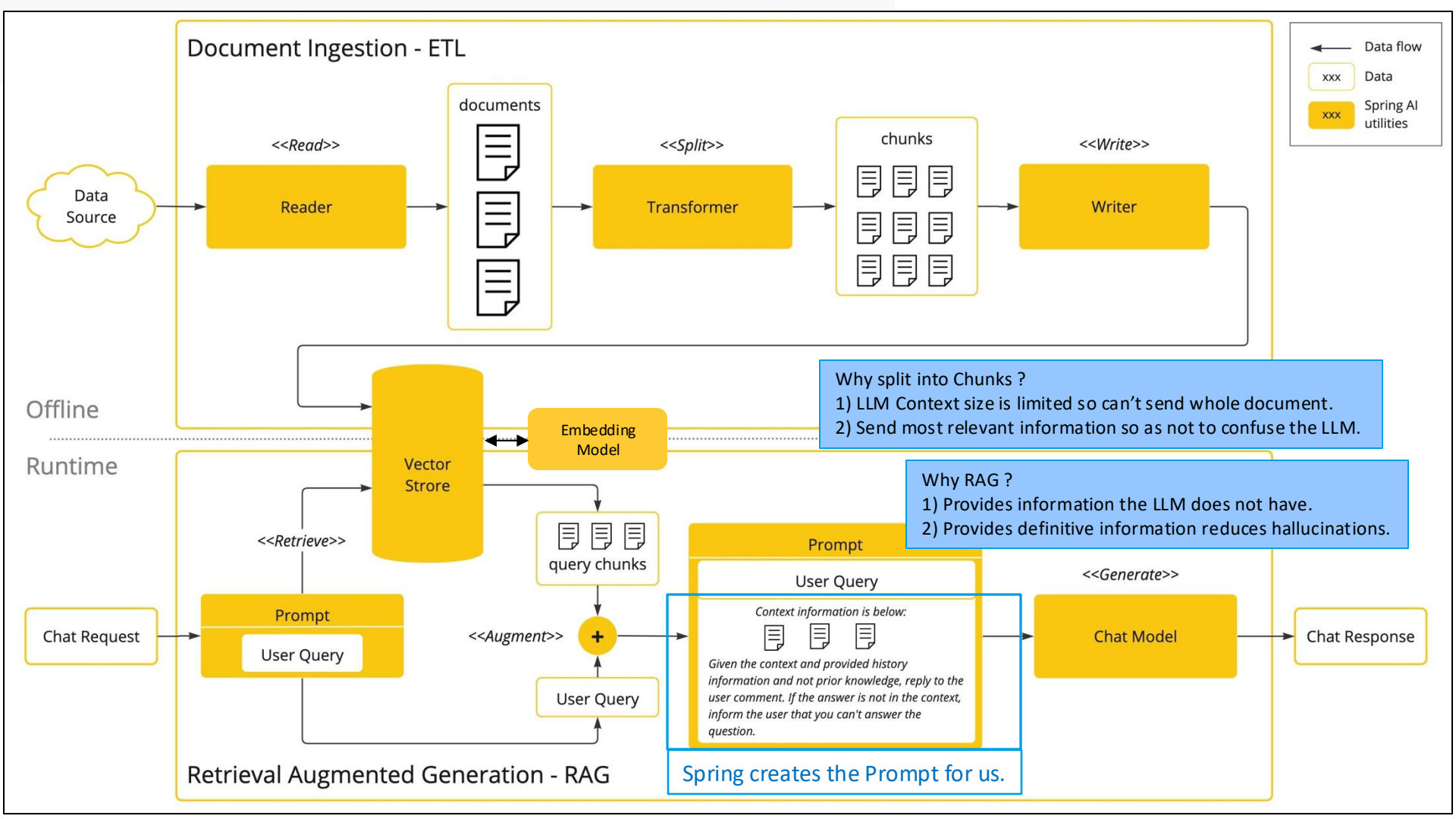


Let's say we have the following instances of those three ETL types:

- **PagePdfDocumentReader** an implementation of DocumentReader
- **TokenTextSplitter** an implementation of DocumentTransformer, to split into chunks
- **VectorStore** an implementation of DocumentWriter
- Many other built-in ETL types.

We can now read data (documents), split them into chunks, then load into a Vector Store, via an Embedding model, with the following code:

```
vectorStore.add(tokenTextSplitter.apply(pdfReader.get()));
```

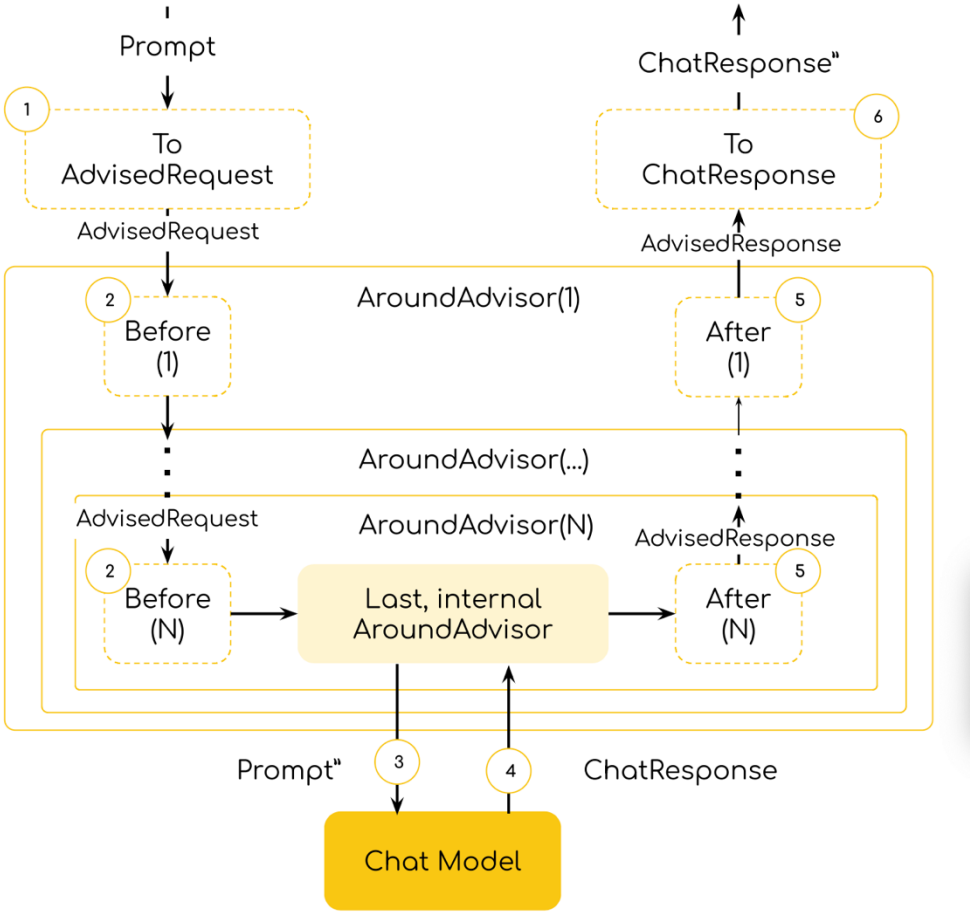

Document Ingestion - ETL
Data flow
xxx
Data
xxx
Spring AI utilities
Data Source
<<Read>>
Reader
documents
<<Split>>
Transformer
chunks
<<Write>>
Writer
Offline
Runtime
Vector Strore
Embedding Model
Why split into Chunks ?
1) LLM Context size is limited so can't send whole document.
2) Send most relevant information so as not to confuse the LLM.
Why RAG ?
1) Provides information the LLM does not have.
2) Provides definitive information reduces hallucinations.
<<Retrieve>>
query chunks
Prompt
User Query
<<Generate>>
Chat Request
Prompt
User Query
<<Augment>>
Context information is below:
Given the context and provided history information and not prior knowledge, reply to the user comment. If the answer is not in the context, inform the user that you can't answer the question.
Chat Model
Chat Response
User Query
Retrieval Augmented Generation - RAG
Spring creates the Prompt for us.

Advisors

The Advisors API provides a flexible and powerful way to intercept, modify, and enhance interactions with LLMs.

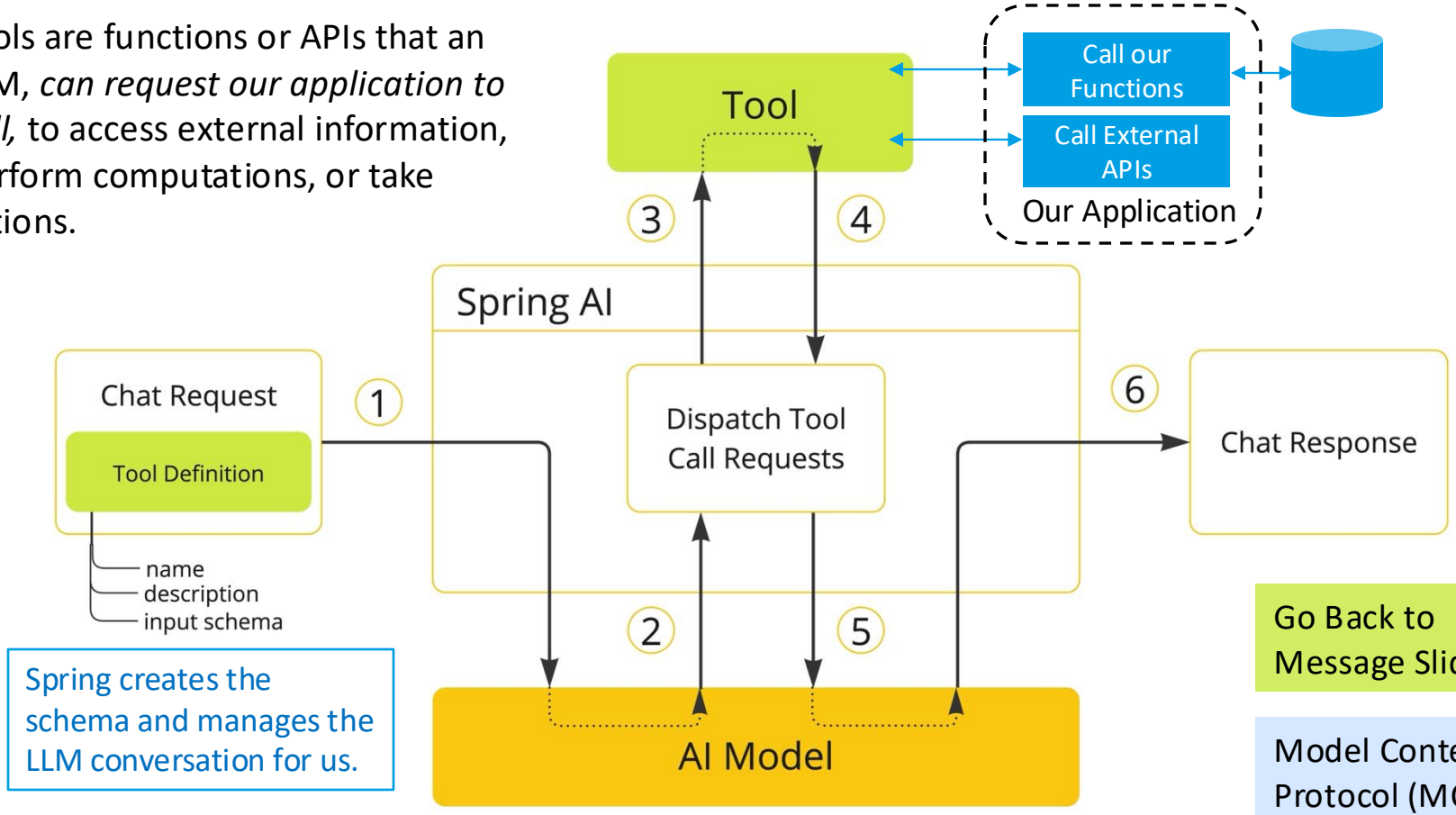
Types:

1. MemoryAdvisor
2. QuestionAnswerAdvisor – RAG
3. SafeGuardAdvisor
4. SimpleLoggingAdvisor
5. ……
6. ……
7. Roll your own….



Tools (functions) – Perform actions, retrieve our information

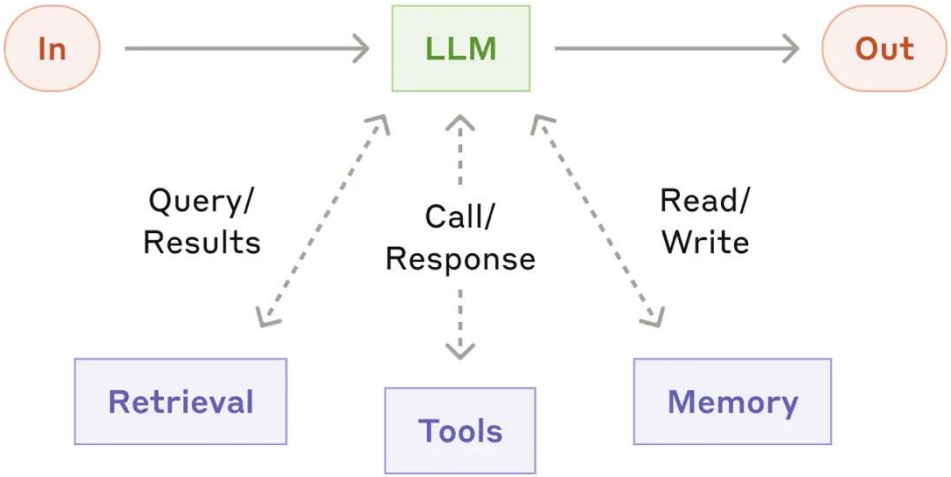
Tools are functions or APIs that an LLM, *can request our application to call,* to access external information, perform computations, or take actions.



The Augmented LLM – Anthropic's Agent* Patterns

The basic feature of Spring-AI now provide all the building blocks for complex agentic systems, and its easy to implement their patterns.

https://www.anthropic.com/research/building-effective-agents



Anthropic Patterns :

- Chaining
- Routing
- Parallelization
- Orchestration
- Evaluator-Optimizer
- Agent*

*A definition is not really agreed yet

Examples

INFOMEDIA™

1) **Chat Bot – RAG :**
   - talk to an Invoice PDF
   - Run the PDF ETL, Embeds in Vector store, search, calls LLM with right Context
   - Uses OpenAI – GPT-4o

2) **Support Application/Agent:**
   - Routing Agent Pattern - to pass support tickets to the right expert team.
   - Uses Local Ollama
     - Spring loads llama3.2 for us

About this Invoice ?

The Big Company

27 Little Street

Sydney, 2000

Phone: (061) 123-4567

INVOICE

INVOICE #	DATE
2034	21/2/2018

BILL TO	CUSTOMER ID	TERMS
Fred Smith Small Company 40A Big Street Nice Ville, 3456 0437 856342 jimy@small.com.au	564	Due Upon Receipt

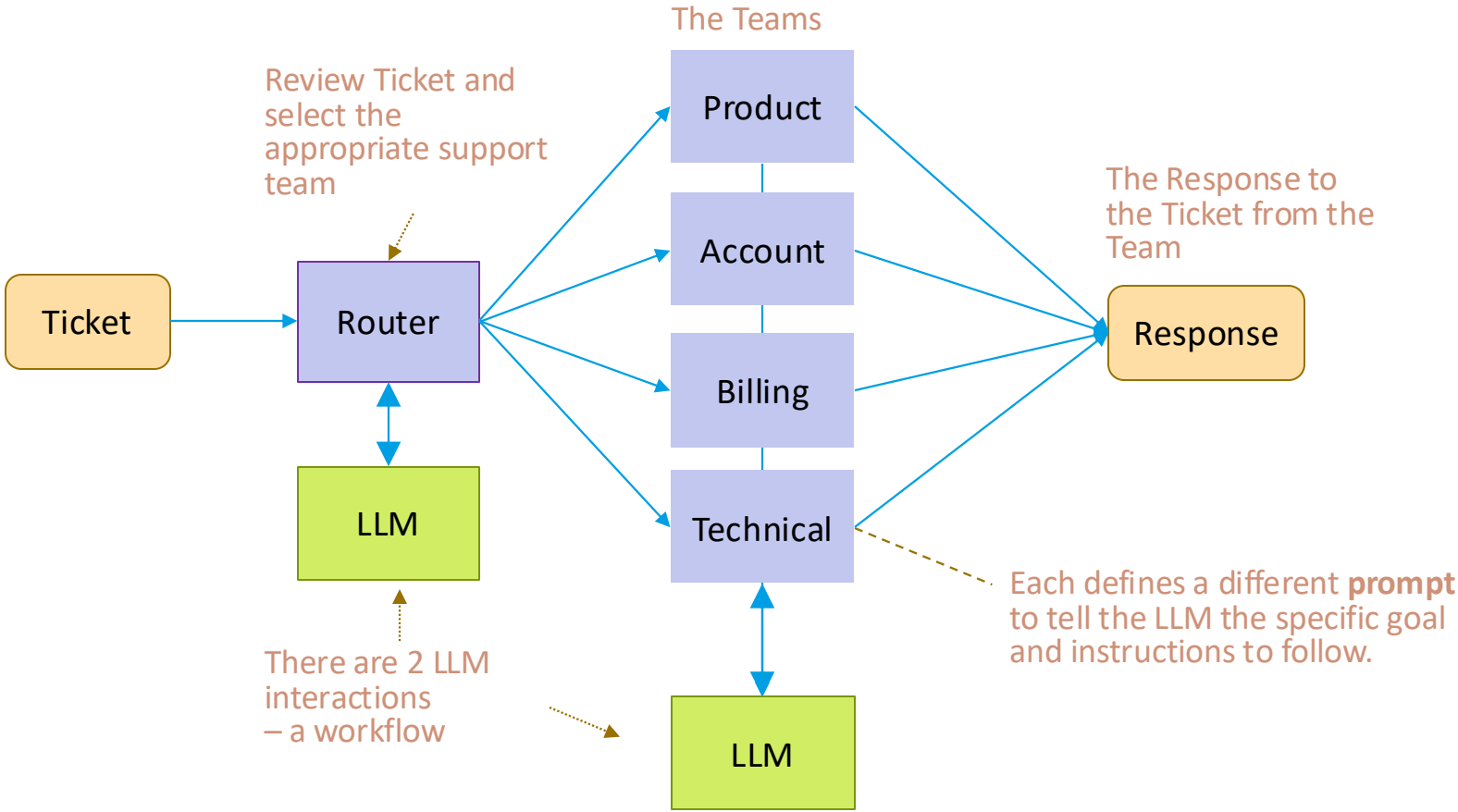
DESCRIPTION	QTY	UNIT PRICE	AMOUNT
Service fee	1	200.00	200.00
Labor: 5 hours at $75/hr	5	75.00	375.00
New client discount		(50.00)	(50.00)
Part Item 4523 - Big Nut	3	45.00	135.00
Part Item 3423 - Flexble cable	1	12.00	12.00
			-
			-
			-
			-
			-
			-
			-
			-
			-

Thank you for your business!

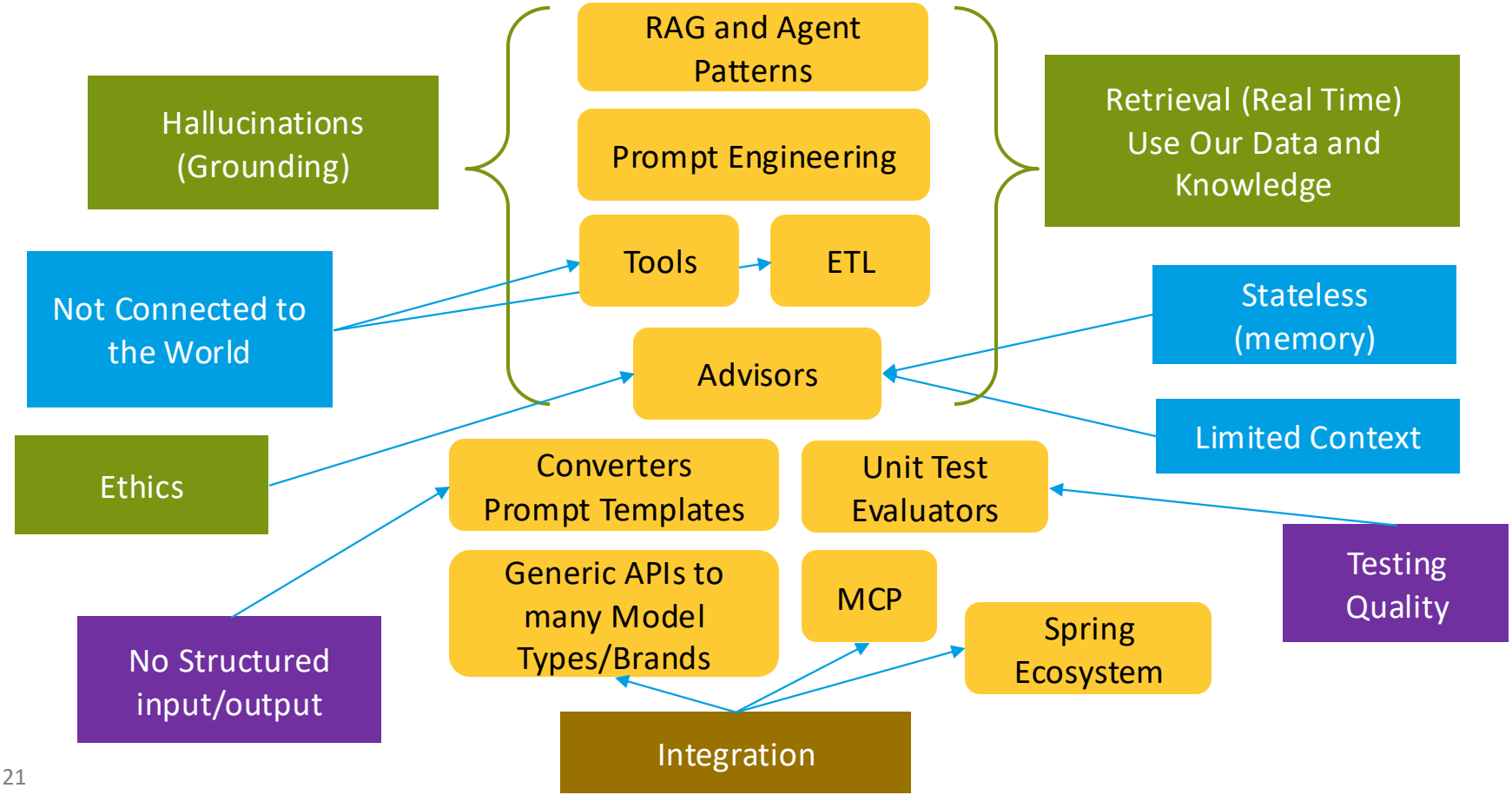
SUBTOTAL	672.00
TAX RATE	4.250%
TAX	28.56
TOTAL	**$ 700.56**

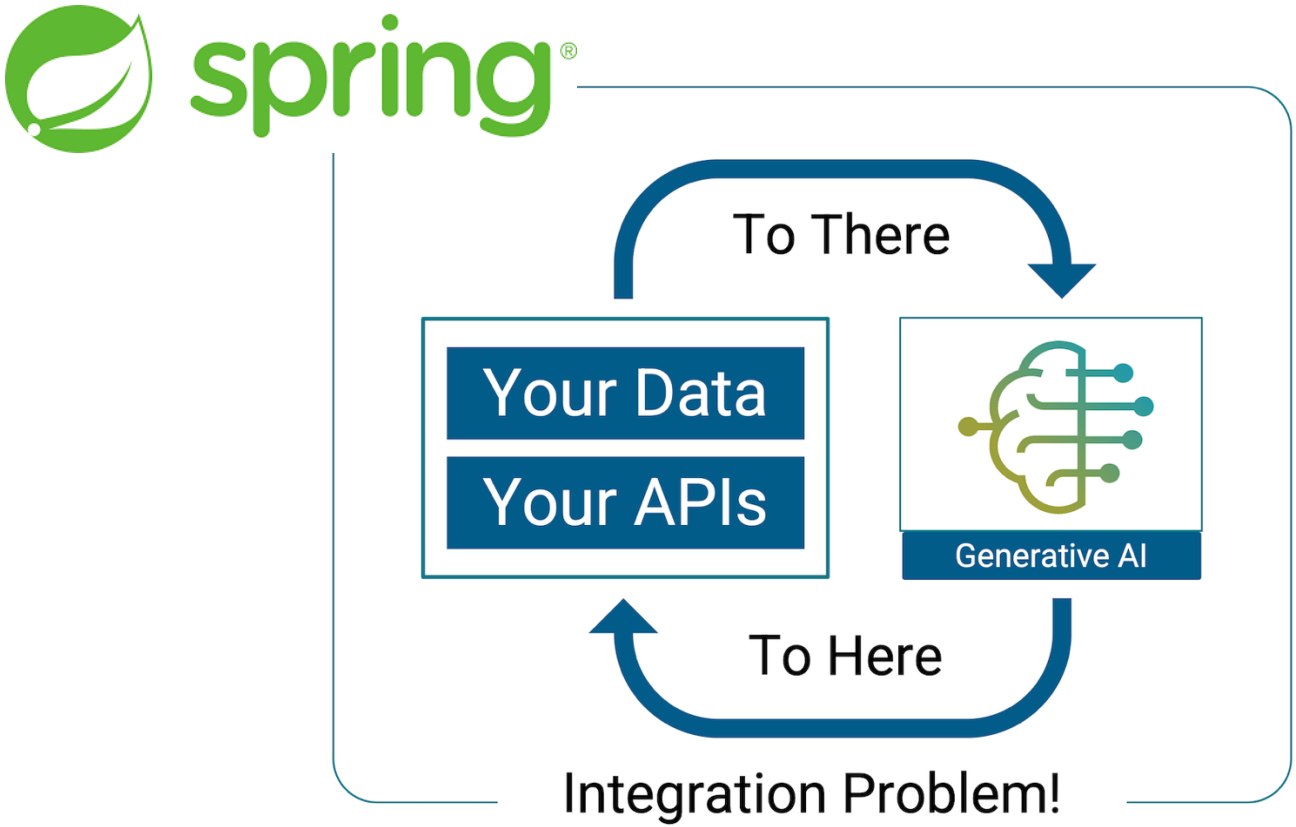
If you have any questions about this invoice, please contact
Jim, 0437 856342, jimy@small.com.au

Router Application



Spring AI Integration Solutions




spring®
To There
Your Data
Your APIs
Generative AI
To Here
Integration Problem!

The End