# Hands-on GenAI Development Bootcamp Day 1

John Davies

29th September 2025


Welcome
Willkommen

# What we're going to cover today

## The morning

- How do agents work?
- Agent patterns and scenarios
- Overview of open source agent frameworks
- Open source tool-calling models (Qwen3, Mistral, Gemma3, Granite4 and others)
- Local deployment with Ollama, LM Studio, MLX, vLLM and cloud alternatives
- Function calling with open source models (with OpenAI comparisons)
- File and document processing with local models

## The afternoon

- Data integration with Agentic RAG using open embeddings
- Model Context Protocol (MCP) - the standardisation layer
- Building custom MCP servers and using community tools
- Multi-agent teams with heterogeneous model selection
- Introduction to Embabel framework for JVM-based agents
- Q&A and discussion throughout the day

# Why Agents Matter Now

**LLMs answer – agents act**

- So many business flows require data, while technical information is usually stored in a database, up-to-date information is often only found on the internet – LLMs help with this task
- A lot of information is stored in human-readable documents – again LLM's help here

**LLMs basically extend what we used to be able to do, the rest however, the flows, is largely just code**

**Before you might have had to code large switch or if/else constructs – today and LLM can "decide" which tool to call based on the information you provided it**

**If information is required to complete a task, it can often be obtained by a tool call (the web, read a doc etc.)**

**Agents provide goal-oriented reasoning loops, LLMs are simply prompt-response system**

# Today’s (rough) agenda

09:00 – 09:15 | Welcome & Overview (15 min)
09:15 – 09:50 | Environment Setup & Verification (35 min)
**09:50 – 10:30 | How Agents Work (40 min)**
10:30 – 11:00 | Morning Break (30 min)

**11:00 – 11:40 | Tool-Calling in Practice (40 min)**
**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**
12:30 – 13:30 | Lunch Break (60 min)

13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)
**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**
**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**
15:00 – 15:30 | Afternoon Break (30 min)

**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

# Frontier vs. Open Source

**Why do we care whether we use ChatGPT or Claude for our work vs. an open source model running locally or in our private cloud?**

**Pro Frontier models (ChatGPT, Claude, Grok etc.)…**

- I think it's fair to assume, for now, that the very top 2% of requirements probably need these models
- The other 98% of tasks can be handled by open source models that are no more than 6 months behind the absolute best frontier models

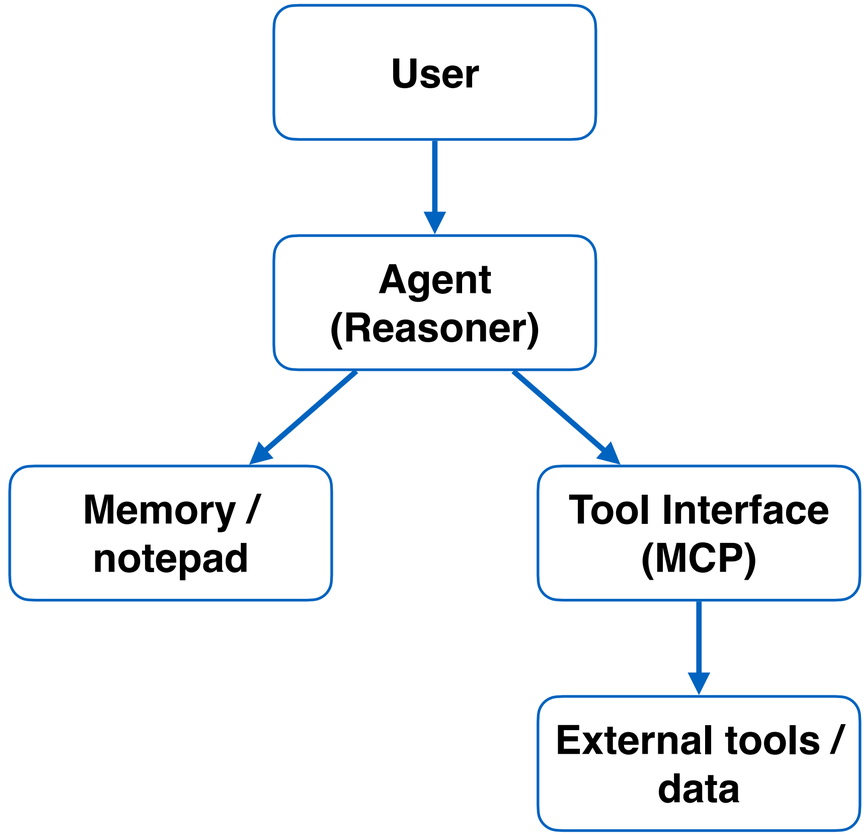
| Aspect | Open Source | Closed source (Proprietary) |
|---|---|---|
| **Privacy** | Full control | Data leaves org. |
| **Latency** | Low / on-device | Remote call plus API |
| **Cost** | Fixed hardware | Pay-per-token |
| **Control** | Full (quantisation / model version etc.) | Limited |
| **Maintenance** | You manage | Vendor manages (remember recent AWS outage) |
| **Consistency / SLA** | Full control | Vendor changes at will |

# Architecture at a Glance

**Agents mediate between users, models, and tools**

- Bear in mind that not everything has to be an LLM
- If you can write something in code then do so

**MCP defines how context and capabilities are exchanged**




**Embabel implements this architecture for the JVM ecosystem.**

# Workshop Methodology

Once we've gotten everyone, or at least most of you, set up we're going alternate between teaching (slides) and building/writing code

I strongly advocate pair programming, humans, AI assistants or both

If you have tools or tips then please share them, we should all be learning from each other here

# Today's Agenda

~~09:00 – 09:15 | Welcome & Overview (15 min)~~
09:15 – 09:50 | Environment Setup & Verification (35 min)
**09:50 – 10:30 | How Agents Work (40 min)**
10:30 – 11:00 | Morning Break (30 min)

**11:00 – 11:40 | Tool-Calling in Practice (40 min)**
**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**
12:30 – 13:30 | Lunch Break (60 min)

13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)
**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**
**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**
15:00 – 15:30 | Afternoon Break (30 min)

**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

# Prerequisites (Voraussetzungen) - part 1

**First we need to make sure everyone has a common platform that we can all work on**

- This is often problematic so we aim to get most people to the same base configuration. We try to provide other options for those who can't

**We are going to work in Java but some of the useful tools need Python too**

- You can use pretty much any programming language, however, we can't provide the same level of support for you
- Please make sure you're comfortable with your IDE and ready to roll… (VC Code, IntelliJ IDEA, PyCharm etc.)

**You will need Java 21 or 25, most things should work with older versions but we won't have time to debug if they don't**

- Make sure you have Maven to match (3.9.11)

**For Python I strongly recommend using MiniConda to manage your Python (it's free and works on pretty much everything)**

- Generally we'll be using Python 3.13.5 but anything from about 3.10 usually works

# Prerequisites (Voraussetzungen) - part 2

**You are going to ideally need some local LLMs (not critical but very useful)**

- To run these I recommend you install Ollama and LM Studio (both free and run on all operating systems)
- If you are using Windows/Linux then I also recommend vLLM and for Mac users install mlx_lm and mlx_vlm (through Python or Github)

**Please download some LLMs. Which ones depends on your machine and how much VRAM (GPU RAM) you have**

- If you're on an old machine with little or no GPU then stick to models under 4GB in size
- If you have more VRAM then download models up to about 2GB less than your VRAM size (for M-series Mac users, your max VRAM is 75% of your total RAM)

**Lastly, if you have or can get a OpenAI, Mistral, Anthropic (Claude) API key then please do, they are usually only a $20 or less**

- We can't share these during the session because they have rate limits. You will need your own keys, especially if you don't have a local model

# Prerequisites (Voraussetzungen) - Parts 1 & 2

**First we need to make sure everyone has a common platform that we can all work on**

- This is often problematic so we aim to get most people to the same base configuration. We try to provide other options for those who can't

**We are going to work in Java but some of the useful tools need Python too**

- You can use pretty much any programming language, however, we can't provide the same level of support for you
- Please make sure you're comfortable with your IDE and ready to roll… (VC Code, IntelliJ IDEA, PyCharm etc.)

**You will need Java 21 or 25, most things should work with older versions but we won't have time to debug if they don't**

- Make sure you have Maven to match (3.9.11)

**For Python I strongly recommend using MiniConda to manage your Python (it's free and works on pretty much everything)**

- Generally we'll be using Python 3.13.5 but anything from about 3.10 usually works

**You are going to ideally need some local LLMs (not critical but very useful)**

- To run these I recommend you install Ollama and LM Studio (both free and run on all operating systems)
- If you are using Windows/Linux then I also recommend vLLM and for Mac users install mlx_lm and mlx_vlm (through Python or Github)

**Please download some LLMs. Which ones depends on your machine and how much VRAM (GPU RAM) you have**

- If you're on an old machine with little or no GPU then stick to models under 4GB in size
- If you have more VRAM then download models up to about 2GB less than your VRAM size (for M-series Mac users, your max VRAM is 75% of your total RAM)

**Lastly, if you have or can get a OpenAI, Mistral, Anthropic (Claude) API key then please do, they are usually only a $20 or less**

- We can't share these during the session because they have rate limits. You will need your own keys, especially if you don't have a local model

# Some LLMs (models) you're going to need

**Remember you don't have to have these but it is the future. If you can't run these locally then we will have other solutions (sharing machines) but please try to bring your own OpenAI and Anthropic keys**

**For everyone… (just type "ollama pull modelname" to install)**

- LLMs
  - **granite4 (from IBM)**
  - **gemma3 (from Google)**
  - **mistral (the EU model from Mistral)**
  - **qwen3-vl:4b (A vision model from Alibaba Qwen)**
  - **qwen3:4b-instruct-2507-q8_0 (A instruct model from Alibaba Qwen)**
  - **qwen3:4b-thinking-2507-q8_0 (A thinking model from Alibaba Qwen)**
- Embedding models
  - **embeddinggemma**
  - **all-minilm**
  - **nomic-embed-text**
  - **qwen3-embedding:4b**

**For those with more VRAM (or Apple Silicon Macs) - All of the above first**

- Then fp16 versions of all the above (it will be good to keep both versions to compare)
- gpt-oss (13GB), qwen3:8b (4b & fp16 - 5 & 16GB), qwen3:30b-a3b (19GB), qwen3:32b (20GB), deepseek-r1:8b-0528-qwen3-fp16. Feel free to download others in Q4/4bit and fp16 options

# Environment Verification

https://mini.story-turtle.ts.net/w-jax/ollama

~/.ollama/models
tar -xf model

## ollama list

```
NAME                                ID              SIZE      MODIFIED
llama3.1:8b-instruct-fp16           4aacac419454    16 GB     2 weeks ago
incept5/Jan-v1-2509:fp16            3c58700d6a54    8.1 GB    3 weeks ago
Jan-v1-2509-gguf:Q8_0_32k           fed6c5779db0    4.3 GB    4 weeks ago
hf.co/janhq/Jan-v1-2509-gguf:Q8_0   1112f7e7730f    4.3 GB    4 weeks ago
```

## ollama run gemma3 --verbose

```
>>> Munich is the best city in Germany, discuss!  Reply in German.

Okay, du hast recht, München ist eine unglaublich beliebte und für viele auch die beste Stadt in Deutschland!
Lass uns darüber diskutieren. Ich verstehe deinen Standpunkt vollkommen. Hier ist meine Sichtweise, und dann
gerne deine Gedanken dazu:

München hat einfach so viel zu bieten, was es zu einem so attraktiven Ort macht. Der erste, und vielleicht
wichtigste Punkt, ist die **Lebensqualität**. München wird regelmäßig zu den Städten mit der höchsten
Lebensqualität weltweit gekürt, und das aus gutem Grund.
```

```
total duration:       5.202473625s
load duration:        120.80975ms
prompt eval count:    25 token(s)
prompt eval duration: 84.6675ms
prompt eval rate:     295.27 tokens/s
eval count:           581 token(s)
eval duration:        4.860904919s
eval rate:            119.53 tokens/s
```

## java -version

```
openjdk version "25" 2025-09-16
OpenJDK Runtime Environment (build 25+36-3489)
OpenJDK 64-Bit Server VM (build 25+36-3489, mixed mode, sharing)
```

## mvn -v

```
Maven home: /opt/homebrew/Cellar/maven/3.9.11/libexec
Java version: 25, vendor: Oracle Corporation, runtime: /Users/jdavies/.sdkman/candidates/java/25.ea.36-open
Default locale: en_GB, platform encoding: UTF-8
OS name: "mac os x", version: "26.0.1", arch: "aarch64", family: “mac"
```

## python --version

```
Python 3.13.5
```

# Repository Setup

## Clone: git clone `https://github.com/Incept5/w-jax-munich-2025-workshop`

### Quick Start

#### 1. Clone and Build

```
# Clone the repository
git clone <repository-url>
cd w-jax-munich-2025-workshop

# Build all modules
mvn clean package
```

#### 2. Verify Setup (Stage 0)

```
cd stage-0-demo
./run.sh "Hello from W-JAX Munich!"
```

Expected output: Response from your local LLM with timing information.

#### 3. Run Your First Agent (Stage 1)

```
cd stage-1-simple-agent
./run.sh "What's the weather in Munich?"
```

# Today's Agenda

~~09:00 – 09:15 | Welcome & Overview (15 min)~~
~~09:15 – 09:50 | Environment Setup & Verification (35 min)~~
**09:50 – 10:30 | How Agents Work (40 min)**
10:30 – 11:00 | Morning Break (30 min)

**11:00 – 11:40 | Tool-Calling in Practice (40 min)**
**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**
12:30 – 13:30 | Lunch Break (60 min)

13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)
**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**
**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**
15:00 – 15:30 | Afternoon Break (30 min)

**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

# From LLMs to Agents

**LLMs predict text (or images/speech etc.), they do this probabilistically**

- LLMs do not think or reason, they just appear to, you could argue they do it better many people

**LLMs have no state or memory, every time you run it, it has no information about the previous time**

**An agent is external to the LLM, it can record/“remember” the previous output and provide that the next time as a “memory”**

**An agent can offer the LLM a choice of tools and the LLM can suggest one (or more), the agent then runs this tool**

- Remember that the LLM has no memory, not only do you need to provide the output of the tool to the LLM but also tell it that it suggested the tool in the first place

**It is this second call to the LLM (with the question and answers from the first call and response from the tool) that provides real value**

- Two calls, first is input and tool choice, second is everything prior and the response from the tool call

# The Agent Loop

**The agent loop is Observe → Think → Act → Reflect**

**Observe = collect input or environment change**

- This is typically the first call but could be just data we've read programatically

**Think = internal reasoning (prompt expansion or plan)**

- This is typically the second call but could be the first with programatically read data from above

**Act = tool execution**

- Usually but not always a tool, it could be a simple response/decision from the LLM

**Reflect = evaluate result / update memory**

- Effectively the output – or input into the next loop

# Core Agent Components

**Reasoner (LLM brain) - generates thoughts/actions**

- You need a good reasoning models, that doesn't mean one with encyclopaedic knowledge
- You need a good tool-calling model, the more tools you have the better the model

**Toolset – JSON-declared functions exposed to model**

- The tools themselves need to be distinct, clear and well defined (like real code!)

**Memory – context window + persistent vector store**

- This itself could be a tool but equally a core part of your system (e.g. a blackboard)

**Environment – state manager and I/O interface**

- Again this is code just like in the good-old days

# Debugging

**The flows are, by definition, non-deterministic so when something doesn't work the way you expected it to work, you need to have a good audit-trail or debug**

**I tend to log models, input (system and user), parameters (e.g. temperature) and output for every call - Yes, it's a LOT of logging but it makes for excellent post-execution analysis**

**Your prompts will never be the right first time, you will need to change them to better "manipulate" the output, having logs makes it easier to see and compare the results**

- You can also isolate some specific agents for better optimisation

**I recently (vibe) coded a debug analyser, it allowed me to optimise the models I was using for better performance**

- I could see the context, sophistication of the prompt and output latency

# Architectures

**I don’t follow any of these specifically, they all make sense, in reading them I see things I’ve used**

**ReAct = Reason + Act (uses chain-of-thought and tool invocations)**

- Chain of thought can be split off into different agents

**Reflexion = agent reviews previous failures to improve**

- This is almost like self-learning, we modify the prompt to provide better results based on the past

**Planner - Executor = two agents (split planning and execution)**

- This is almost de facto today, I would strongly recommend splitting ALL of your Gen AI
- I’d go further, Plan - Execute - Reflect

**Supervisor–Worker = hierarchical variant used in Embabel**

- Again a very common pattern, I like to think of this as the “office model”, split the tasks as you would have done in a real (human) system

# Memory & Context

**Context is expensive (memory and time), double the context and you quadruple the memory**

**Always use enough but never too much**

**If you want memory then summarise it, perhaps store the original off-line, add to that and then summarise everything each time you update it**

- Repeated summarisation can result in errors (AI dementia)

**You can use a vector database (pgVector, Chroma etc.) but don't over-engineer**

**I prefer a raw "memory" that I can summarise and perform RAG on**

# When to Use Agents

**Personally I've been using agents since day one of open source LLMs**

- It's just another component we have in our programmer's arsenal
- Once you've learnt how to use them you can just add them where needed

**Obvious places…**

- Multi-step decisions and dynamic tool selection
- Semi-autonomous pipelines (data enrichment, ops bots)

**Avoid agents for:**

- Simple transformations or deterministic tasks – WRITE CODE!

**I'm a strong advocate of KISS, never pre-optimise, don't over engineer**

# Let’s get some models working and test them out

We’re going to get into the code of this later but to make sure everything’s working let’s get the code running

This is a function-calling-demo, it basically runs through every model you have in Ollama (with filters) and will run a few function-calling test to test the speed and ability to make the call

On my machine (with well over 100 models) I ran the following

```
mvn clean compile
./run-tests.sh jan 50
```

- This runs all models with “jan” in the name and under 50GB in size

The output (after a few minutes)…

```
================================================================================
FUNCTION SUPPORT TEST RESULTS SUMMARY
================================================================================
Model                                Size (GB)    Time (ms)    σ Time       Success Rate
--------------------------------------------------------------------------------
incept5/Jan-v1-2509:fp16             8.05         2653         ±711         30/30
Jan-v1-2509-gguf:Q8_0_32k            4.28         1697         ±370         30/30
hf.co/janhq/Jan-v1-2509-gguf:Q8_0    4.28         1596         ±481         30/30
hf.co/janhq/Jan-v1-2509-gguf:latest  2.50         1304         ±410         30/30
mannix/jan-nano:ud_q8_k_xl_32k       5.06         2128         ±786         30/30
mannix/jan-nano:ud_q8_k_xl           5.06         2109         ±740         30/30
mannix/jan-nano:latest               2.27         1345         ±528         30/30
```

# Today's Agenda

~~09:00 – 09:15 | Welcome & Overview (15 min)~~
~~09:15 – 09:50 | Environment Setup & Verification (35 min)~~
~~**09:50 – 10:30 | How Agents Work (40 min)**~~
10:30 – 11:00 | Morning Break (30 min)

**11:00 – 11:40 | Tool-Calling in Practice (40 min)**
**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**
12:30 – 13:30 | Lunch Break (60 min)

13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)
**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**
**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**
15:00 – 15:30 | Afternoon Break (30 min)

**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

# Today's Agenda

~~09:00 – 09:15 | Welcome & Overview (15 min)~~
~~09:15 – 09:50 | Environment Setup & Verification (35 min)~~
~~**09:50 – 10:30 | How Agents Work (40 min)**~~
~~10:30 – 11:00 | Morning Break (30 min)~~

**11:00 – 11:40 | Tool-Calling in Practice (40 min)**
**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**
12:30 – 13:30 | Lunch Break (60 min)

13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)
**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**
**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**
15:00 – 15:30 | Afternoon Break (30 min)

**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

# Purpose of Tool-Calling

**Remember when Chat GPT first came out?**

- No internet, no web, no up-to-date information

**Then they added “The Web”**

**The web is a tool, ask for something up-to-date or in the news and Chat GPT (or Claude etc.) will call the internet and search**

**Tool-calling is the ability, as the name suggests, to call tool (a.k.a. functions)**

**These can be the web, a database, the weather, the current date/time, your recent chat history**

**However, over the last few months this has gone crazy and almost anything is now a tool-call away with MCP**

# Tool Schema Structure

Typically a tool call is described in JSON however some models prefer or also handle XML

Either way, the structure is almost the same, you provide the model with the name of the tool (function name), the description, parameters and return type

```
{
   "name": "get_weather",
   "description": "Retrieve weather for a city",
   "parameters": {
      "type": "object",
      "properties": { "city": {"type":"string"} },
      "required": ["city"]
    }
}
```

This tells the LLM that is someone wants the weather for a city it needs to reply (to you) with the name of the function “get_weather” and the name of the city

- It does this by matching the request with the function name and description so the better the description and closer the name to the function the mode likely the call is to work

# How Models Handle Schemas

**Not all models handle JSON well, they are all getting better as it's becoming critical but you need to…**

- Test the models (see code recently used)

**Sometimes the full model (fp16) works fine but the quantised version is not so good**

- Test the models

**Sometimes it works fine for one, two of three tools and then falls down as you hit a dozen**

- Test the models

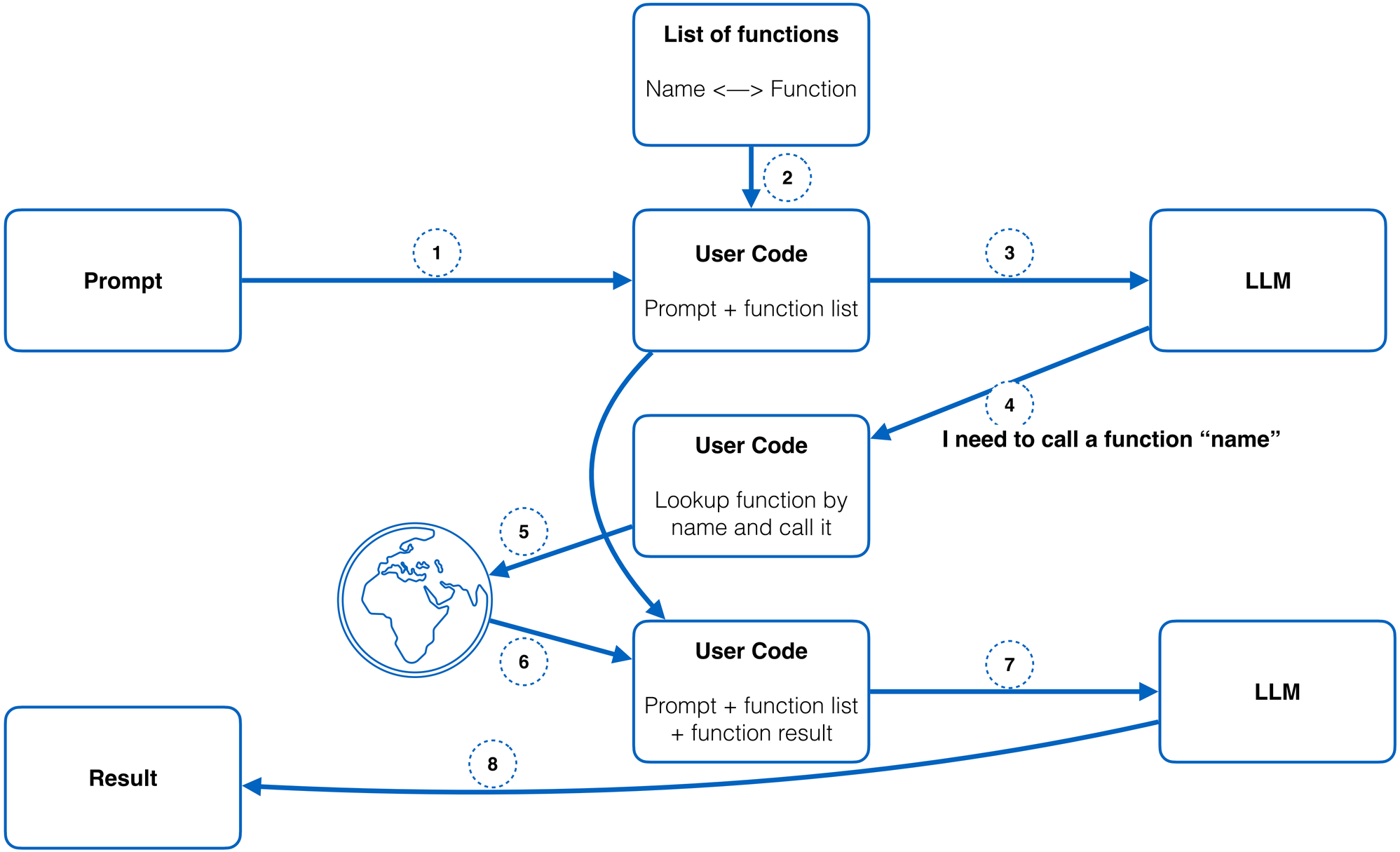
**You can use examples in your prompt**

- This works extremely well, think of it like teaching other people (or kids), teach by example

**Don't make your complex, remember —> KISS**

- You're better to take a little longer and do it twice than fail because you made it too complex

# Tool-calling flow



# Handling Errors & Retries

**There are so many sources of problems**

- The LLM could get stuck repeating crap, return an error or just something unintelligible
- The function might not return, of may return something you weren't expecting
- The JSON can get corrupted, perhaps embedded brackets or similar
- The LLM might just "decide" not to make the call you were expecting
- The resource might be rate-limited
- The LLM might be rate-limited (a good reason to use a local one)

**So, plenty of logging and keep an eye on it**

- You can even make the logging part of your agentic flow

**Use timeouts and catch exceptions**

# Latency & Reliability

**Every LLM call adds latency, CPU/GPU resource and complexity**

- If you can do it in code then use code - keep your job safe!

**Whether you're cloud-centric or local, the latter (local models) can significantly reduce latency for very little (GPU) cost**

**Tune both quantisation and context size, test, test, test!**

- And test again!

**Parallel execution can sometimes make a significant difference**

- However, it can equally add significant complexity

# Exercise: Tool Design Challenge

**Take the function-calling code and add a new function to convert currency**

**You don't need to actually call something on the internet but you could return a dummy rate and display the currencies**

- For example "What is EUR 50 in USD?" —> results in call to fx() with params…
  - **amount = 50.00**
  - **from = EUR**
  - **to = USD**
- If you check the parameters and return them in the form "50.00-EUR-USD" you can see if it worked

# Code Example: Simple Tool Interface

## The actual function...

```
private static String convertCurrency(double amount, String from, String to) {
    // Simple exchange rates (hardcoded for demo)
    double rate = switch(from + "-" + to) {
        case "USD-GBP" -> 0.79;
        case "USD-EUR" -> 0.91;
        case "EUR-USD" -> 1.10;
        case "GBP-USD" -> 1.27;
        default -> 1.0;
    };

    double result = amount * rate;
    return String.format("%.2f %s = %.2f %s (rate: %.2f)", amount, from, result, to, rate);
}
```

# Defining the tool(s) for the LLM

```
{
  "model": "%s",
  "messages": [
    {"role": "system", "content": "%s"},
    {"role": "user", "content": "%s"}
  ],
  "tools": [
    {
      "type": "function",
      "function": {
        "name": "convert_currency",
        "description": "Convert an amount from one currency to another",
        "parameters": {
          "type": "object",
          "properties": {
            "amount": {"type": "number", "description": "Amount to convert"},
            "from_currency": {"type": "string", "description": "Source currency code"},
            "to_currency": {"type": "string", "description": "Target currency code"}
          },
          "required": ["amount", "from_currency", "to_currency"]
        }
      }
    }
  ],
  "stream": false
}
```

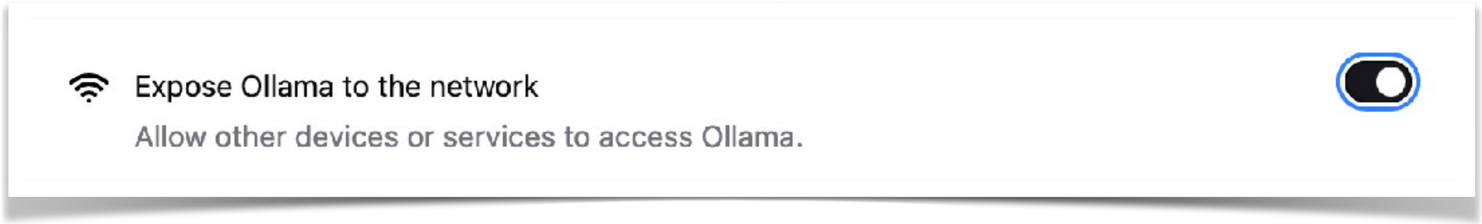
# Detecting the tool-call

## Unwrapping the result to call the tool

```
if (response1.contains("\"tool_calls\"")) {
    System.out.println("Model wants to call a function!");

    // Check which function is being called
    String functionName = extractString(response1, "name");
    System.out.println("   Function: " + functionName);

    String functionResult;
    if ("convert_currency".equals(functionName)) {
        // Parse function call parameters
        double amount = extractNumber(response1, "amount");
        String from = extractString(response1, "from_currency");
        String to = extractString(response1, "to_currency");

        System.out.println("   Arguments: amount=" + amount + ", from=" + from + ", to=" + to);

        // Execute the function
        functionResult = convertCurrency(amount, from, to);
        System.out.println("   Result: " + functionResult);
```

# Explore the code…

## Explore SimpleFunctionCalling.java




## Somethings to try…

- Different models
- If you have LM Studio, try that too
- Try to access someone else's Ollama (with permission) - There is an option in Ollama to expose the service to the network (in settings)

## We won't spend long here, it's just so that you understand what's happening at the lowest level

# Today's Agenda

~~09:00 – 09:15 | Welcome & Overview (15 min)~~
~~09:15 – 09:50 | Environment Setup & Verification (35 min)~~
~~**09:50 – 10:30 | How Agents Work (40 min)**~~
~~10:30 – 11:00 | Morning Break (30 min)~~

~~**11:00 – 11:40 | Tool-Calling in Practice (40 min)**~~
**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**
12:30 – 13:30 | Lunch Break (60 min)

13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)
**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**
**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**
15:00 – 15:30 | Afternoon Break (30 min)

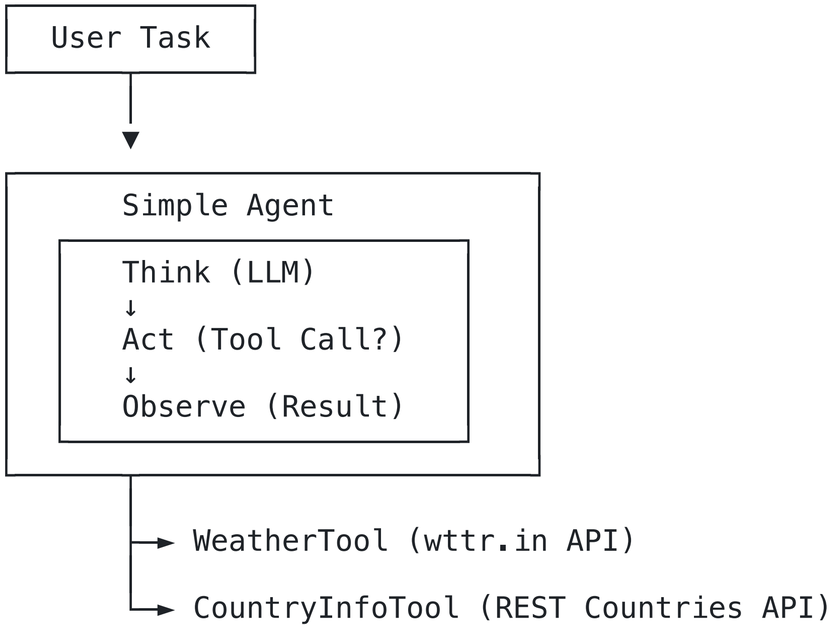
**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

# Exercise - Simple Agent with Tool Calling

**A simple but powerful AI agent that can reason about tasks and use real-world API tools to accomplish them. This stage introduces the fundamental agent architecture: Think → Act → Observe**

## Overview

- This agent demonstrates:
  - **Agent Loop: The classic think-act-observe pattern**
  - **Tool Abstraction: Clean interface for external capabilities**
  - **Real API Integration: Weather (wttr.in) and country info (REST Countries)**
  - **Multi-step Reasoning: Composing tool calls to solve complex queries**
  - **Simple Protocol: XML-like format for tool calling**




**stage-1-simple-agent**

# Today's Agenda

~~09:00 – 09:15 | Welcome & Overview (15 min)~~
~~09:15 – 09:50 | Environment Setup & Verification (35 min)~~
~~**09:50 – 10:30 | How Agents Work (40 min)**~~
~~10:30 – 11:00 | Morning Break (30 min)~~

~~**11:00 – 11:40 | Tool-Calling in Practice (40 min)**~~
~~**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**~~
12:30 – 13:30 | Lunch Break (60 min)

13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)
**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**
**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**
15:00 – 15:30 | Afternoon Break (30 min)

**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

# Today's Agenda

~~09:00 – 09:15 | Welcome & Overview (15 min)~~
~~09:15 – 09:50 | Environment Setup & Verification (35 min)~~
~~**09:50 – 10:30 | How Agents Work (40 min)**~~
~~10:30 – 11:00 | Morning Break (30 min)~~

~~**11:00 – 11:40 | Tool-Calling in Practice (40 min)**~~
~~**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**~~
12:30 – 13:30 | Lunch Break (60 min)

13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)
**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**
**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**
15:00 – 15:30 | Afternoon Break (30 min)

**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

# Morning Recap

**So, we've installed a lot of tools, code and hopefully LLMs**

- Hopefully you have had time to get comfortable with them

**We've explored tool calling from the very basics to the lowest level up to the agentic level**

**I hope you've manage to run all of this locally but as I write (at 2am before this morning), I'm sure some of you will have had issues with machines, lack of memory, slow CPUs, locked-down machines etc.**

- By now you should know that we have other machines you can access for the libraries and of course you can use the frontier models too

**The main issue with starting day one with a frontier models is that everything usually works because they have $100bn of hardware and that helps - A LOT**

**Anyway, now we're going to explore connection to through a standard protocol**

# Today's Agenda

~~09:00 – 09:15 | Welcome & Overview (15 min)~~
~~09:15 – 09:50 | Environment Setup & Verification (35 min)~~
~~**09:50 – 10:30 | How Agents Work (40 min)**~~
~~10:30 – 11:00 | Morning Break (30 min)~~

~~**11:00 – 11:40 | Tool-Calling in Practice (40 min)**~~
~~**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**~~
12:30 – 13:30 | Lunch Break (60 min)

~~13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)~~
**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**
**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**
15:00 – 15:30 | Afternoon Break (30 min)

**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

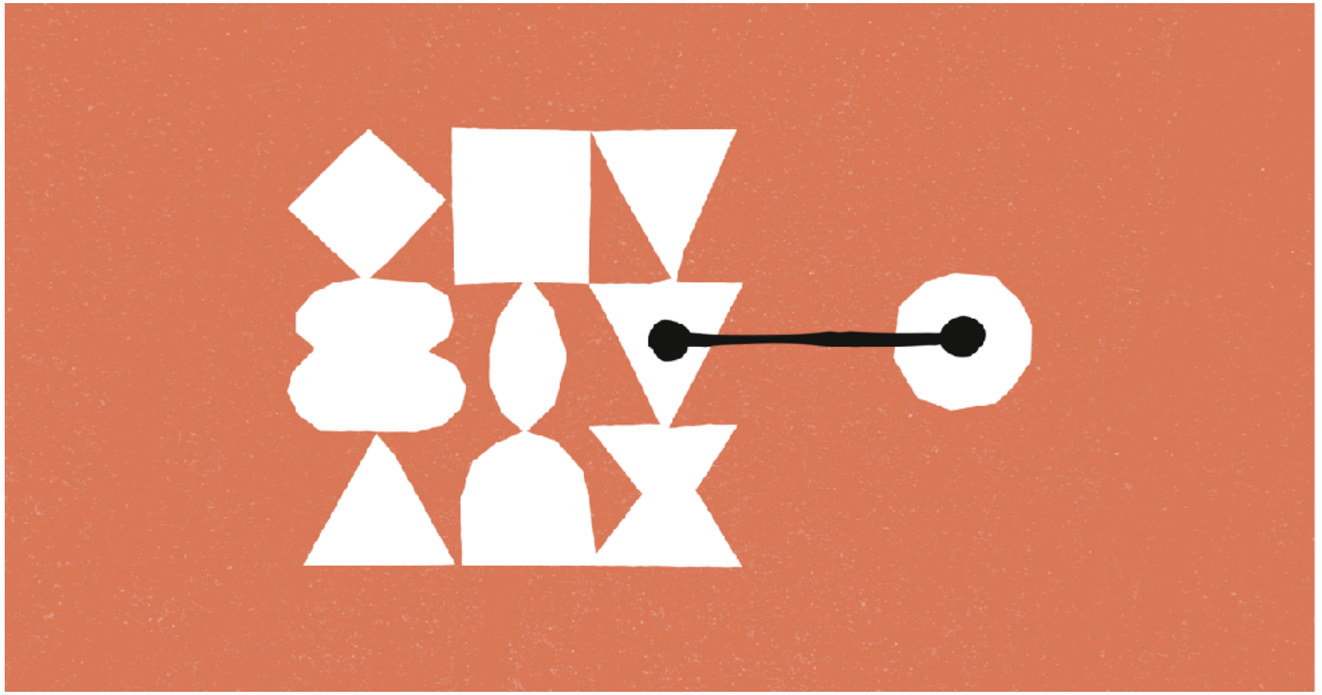
# From Local to Interoperable

At the level you've been working (the lowest), you're not really going to see much difference because you've already experienced the technical level

However, with more and more LLMs, both open source and frontier there was a need for a common way to connect services (tools)

In November last year Anthropic introduced the Model Context Protocol, better known as MCP

It defines a very basic standard for exposing tools on a server, exploring that server and then connection to the tools via a client



# Anthropic’s Model Context Protocol (MCP)

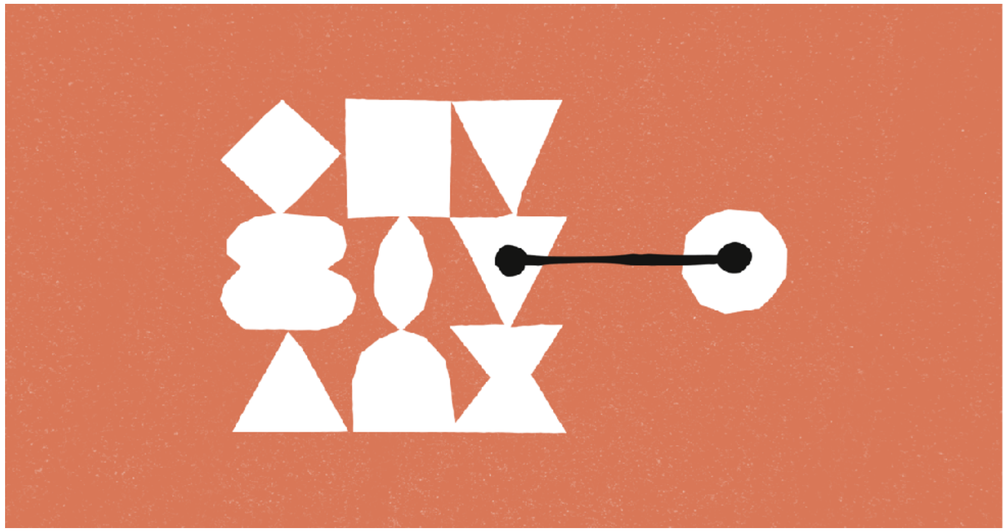
MCP is an approach to standardise tool calling

Tools are written generally in Node or Python and publish on a web site so that others can use them

It doesn’t only work in Claude but can be made (easily) to work in Ollama, OpenAI etc., it’s just a matter of making the calls

**https://modelcontextprotocol.io/quickstart/user**

MCP is an excellent way to build and publish tools for general use



```
{
  "mcpServers": {
    "filesystem": {
      "command": "npx",
      "args": [
        "-y",
        "@modelcontextprotocol/server-
        "/Users/username/Desktop",
        "/Users/username/Downloads"
      ]
    }
  }
}
```

# MCP galore!

https://github.com/modelcontextprotocol/servers

This reminds me of the early internet, before there were search engines, similar to the early days of Java, there used to be a web site in the 1995 or 1996 where you registered your java code to show others

My point is that there are too many and no quality of service or security

By all means browse but be very careful what you use

# MCP Purpose & Scope

**Goal: Define a universal interface between LLM agents and external tools or data sources**

- Enables consistent discovery, invocation, and description of tool capabilities

**Context Exchange: Specifies how session metadata, conversation state, and user context travel with each request**

- Prevents “stateless” tool calls - agents can reason over prior context

**Interoperability: Common contract across vendors, languages, and frameworks (OpenAI, Anthropic, Embabel, LangChain)**

- Promotes plug-and-play tool ecosystems

**Developer Benefit: Reduces vendor lock-in-build once, run anywhere that speaks MCP**

**Enterprise Impact: Unifies compliance, observability, and auditability across toolchains**

**Future Trend: Evolving into a Model-to-Model protocol, allowing chained reasoning across heterogeneous agents**

# Core Objects

## All are serialisable to JSON, allowing transparent transport via HTTP/WebSocket/gRPC

## MCP enforces schema validation for deterministic behaviour across languages

| Object | Description | Java Mapping |
|---|---|---|
| **Tool** | Describes a callable capability (name, parameters, schema). | Interface or POJO annotated with @MCPTool. |
| **Context** | Holds runtime data (session, auth, environment). | MCPContext object—injects user/session metadata. |
| **Session** | Logical channel between client ↔ server. | MCPSession handles lifecycle, reconnect, timeout. |
| **Message** | JSON-RPC 2.0 payload (request/response). | MCPMessage with id, method, params, result. |
| **Error** | Structured exception data. | MCPError extending RuntimeException. |
| **ToolRegistry** | Manages discovery and registration of tools. | Embabel ToolRegistry or custom implementation. |

# Debugging & Monitoring

**Verbose Logging:**
- Enable via `mcp.logging.level=DEBUG or SLF4J`
- Print raw JSON requests/responses for inspection

**Schema Validation:**
- Use jsonschema or Java validators (Everit, Jackson)
- Detect mismatched field names or nullability errors

**Diagnostics Tools:**
- Postman, Wireshark, or wscat for real-time traffic sniffing

**Latency Metrics:**
- Track RTT per call, 95th percentile response time

**Error Codes:**
- -32601 = method not found, -32602 = invalid params, -32000 = server error

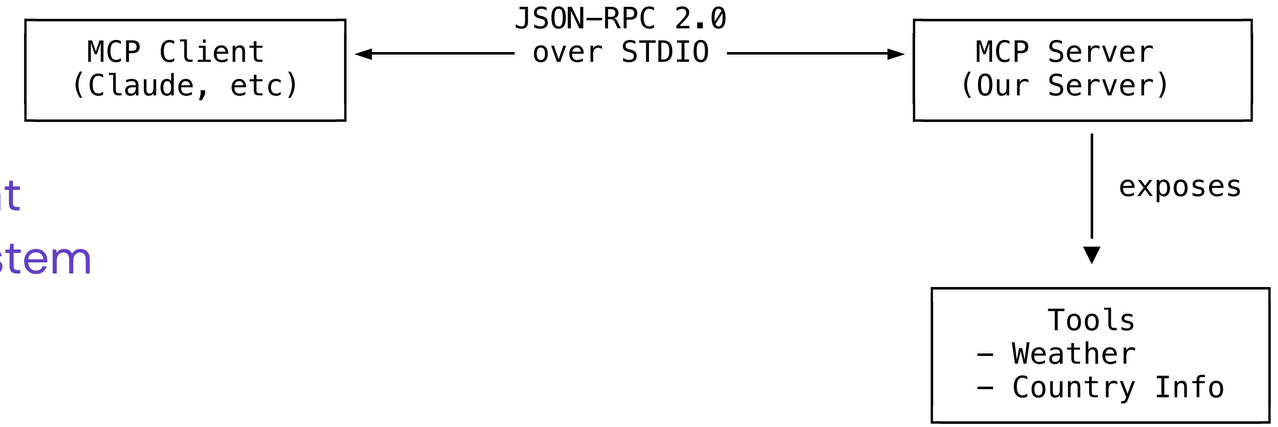
**Observability Stack:**
- Integrate with Prometheus, Grafana dashboards for agent health

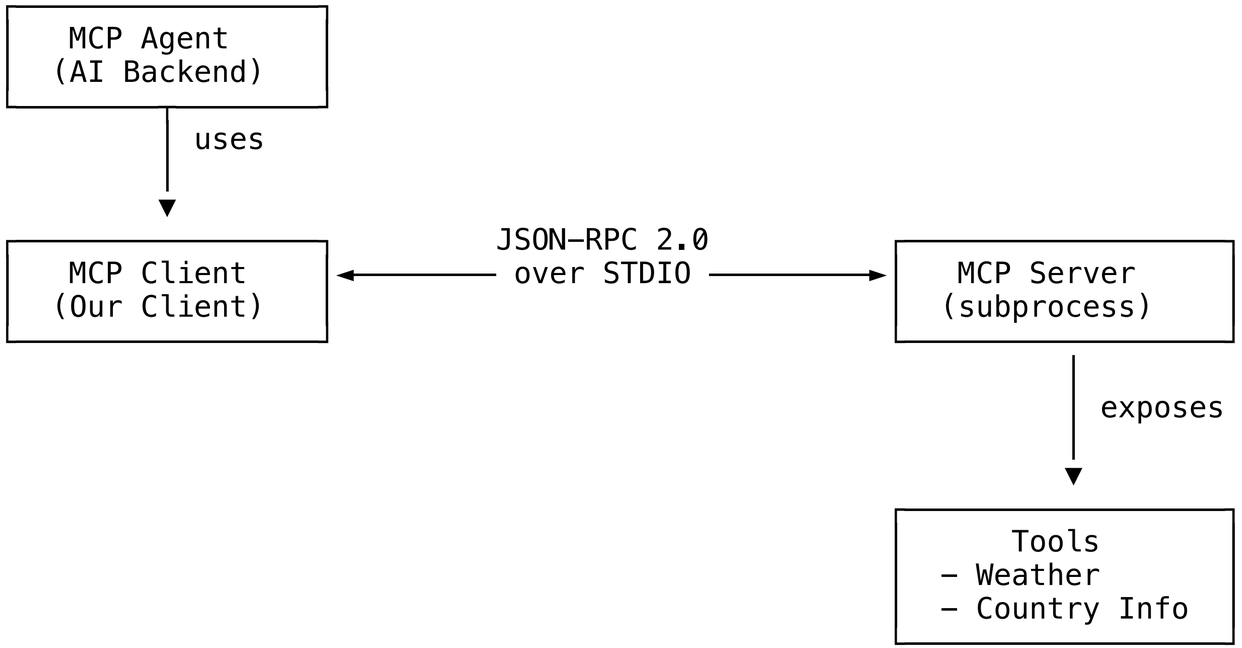
# Exercise - MCP Server

**This stage introduces the Model Context Protocol (MCP), a standardised protocol for connecting AI applications with external tools and data sources**

**You'll learn how to:**

- Expose tools through MCP as a server
- Build an agent that uses MCP tools as a client
- Connect the two to create a complete AI system



# Today's Agenda

~~09:00 – 09:15 | Welcome & Overview (15 min)~~
~~09:15 – 09:50 | Environment Setup & Verification (35 min)~~
~~**09:50 – 10:30 | How Agents Work (40 min)**~~
~~10:30 – 11:00 | Morning Break (30 min)~~

~~**11:00 – 11:40 | Tool-Calling in Practice (40 min)**~~
~~**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**~~
12:30 – 13:30 | Lunch Break (60 min)

~~13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)~~
~~**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**~~
**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**
15:00 – 15:30 | Afternoon Break (30 min)

**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

# Why RAG Matters for Agents

**Definition**

- Retrieval-Augmented Generation (RAG) grounds LLM reasoning in verified external data, extending the model's factual reach

**Purpose in Agents**

- Reduces hallucination by constraining responses to retrieved evidence
- Bridges static model training with live, enterprise knowledge bases

**Operational Flow**

- Agents trigger retrieval before reasoning - think "fetch → think → act"

**Enterprise Benefits**

- Supports auditability (traceable document sources)
- Meets compliance needs (data residency, reproducibility)

**Agent Behaviour**

- Enables context-aware decision-making where actions depend on factual grounding

**Typical Domains**

- Policy analysis, financial summaries, product documentation, or risk evaluation (as used in the workshop exercise)

# RAG Pipeline Overview

**1. Query: user or agent forms a natural-language question**

**2. Embedding: text → numerical vector using an embedding model**

**3. Search: nearest-neighbour match in vector index (FAISS/Chroma)**

**4. Retrieve: fetch top-N snippets with metadata and source**

**5. Synthesis: LLM combines retrieved context + prompt → final answer**

## Design Considerations

- Tune top_k and similarity threshold for precision/recall balance
- Use context window trimming for long snippets (avoid token overflow)

## Java Implementation

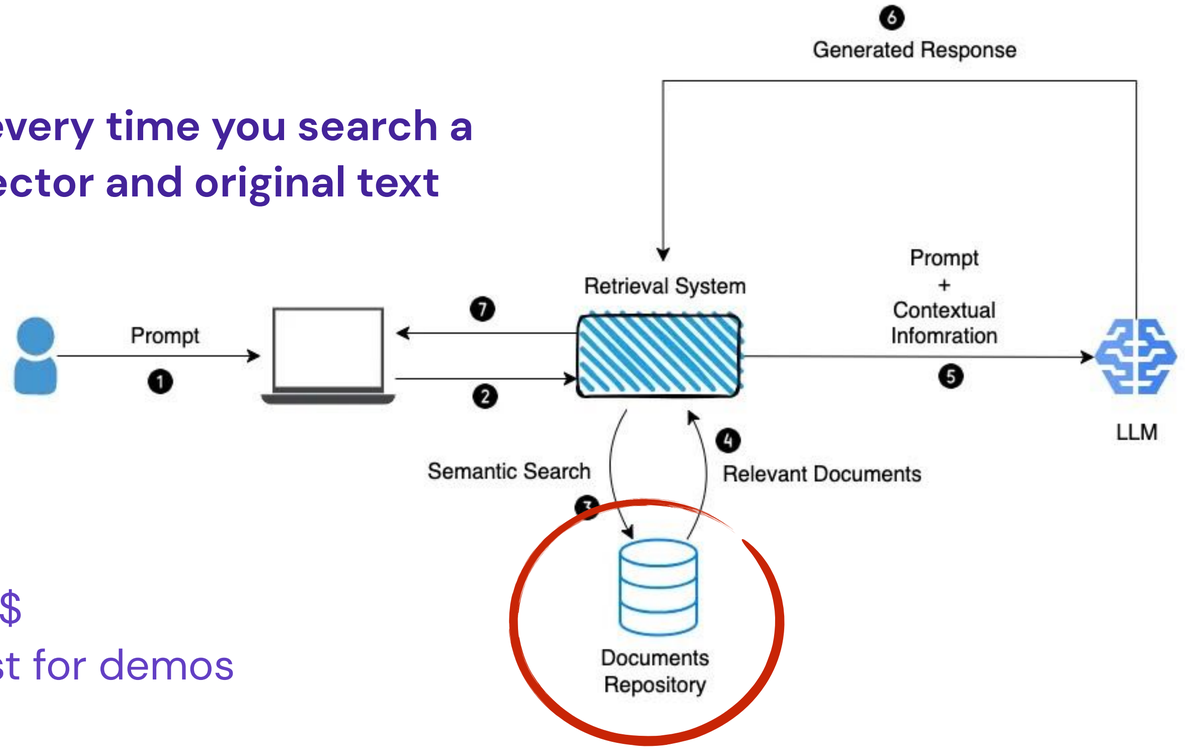
- Embabel RAG module encapsulates this flow via KnowledgeConnector and MCP calls (search_documents, get_snippets)

# Vector databases - storing embeddings

**Using advanced chunking and embedding techniquest could mean that your embedding can take several tens of second**

**You obviously don't want to do chunking and embedding every time you search a document so we can use a vector database to store the vector and original text**

**The database will often take your search vector and brute-force search the data using cosine-similarity returning the best matches**




**Notable Vector databases…**

- Pinecone - Closed, SaaS, easy to get started with then $$$
- Chroma - MIT, relatively easy to use and one of the easiest for demos
- Qdrant - Apache 2, good performance
- pgVector - PostgreSQL, basically and extension on Postgres

**Storing numbers and text in a database is not rocket science, you can easily knock up a table in SQLite but Chroma and pgVector are easy to use**

# Open Embeddings

| Model | Dim | Strength | License | Notes |
|---|---|---|---|---|
| **bge-small-en** | 384 | Fast, multilingual | MIT | Great default for on-device. |
| **nomic-embed-text** | 768 | High semantic accuracy | Apache-2 | Good for policy & long text. |
| **gte-base** | 768 | Code + prose support | MIT | Compact for JVM deployments. |

## Also look at embeddinggemma and Qwen3-embedding

- These are both more modern and multi-lingual

## Local Hosting

- All can run via Ollama or Transformers4J; data never leaves the environment

## Vector Size Trade-off

- Smaller dimensions → faster lookup
- larger → higher precision (usually multi-lingual)

## Batching

- Most embedding APIs support batching for performance
- Compute embeddings asynchronously

## Don’t just use one embedding

- It’s not unusual to use more than one embedding when search results are more critical

# MCP RAG Service

**Architecture**

- Acts as an MCP-compliant retriever server exposing standard tools:
  - **`search_documents(query)` → returns document IDs + scores**
  - **`get_snippets(doc_ids)` → returns text segments**

**Agent Workflow:**

- Agent calls retriever via MCP connector → gets relevant snippets → injects into reasoning

**Benefits:**

- No bespoke HTTP clients; all discovery and schema exchange handled by MCP
- Unified logging, authentication, and observability through MCP layer

**Scalability**

- Supports multiple retrievers (e.g. “policies”, “transactions”)

**Embabel Hook**

- KnowledgeConnector translates MCP responses → ContextMemory entries

**Workshop Alignment**

- Identical to the service in the Stage-3 repo under /mcp-rag-server

# Integration with Embabel

**Connector Role**

- KnowledgeConnector links agent reasoning loop ↔ RAG service

**Execution Path**

- 1. Agent formulates a sub-goal needing external data
- 2. Issues MCP call search_documents
- 3. Retrieves snippets via get_snippets
- 4. Injects them into agent.memory.context
- 5. LLM synthesises grounded output

**Self-Containment: runs locally; no OpenAI/Anthropic APIs required**

**Extensible: easily swap vector stores or embedding back-ends (SQLite → Chroma)**

**Thread-Safety: Embabel handles concurrent tool calls through async Java Futures**

**Workshop Use: the exercise agent (RagAgent.java) demonstrates exactly this flow**

# Debugging RAG

**Validation Metrics:**
- Check embedding coverage = % of queries returning ≥ 1 result
- Use cosine similarity ≥ 0.75 as baseline precision

**Performance Tuning:**
- Profile recall vs latency; tweak vector dim + top_k
- Employ batch retrieval to reduce I/O

**Logging:**
- Log retrieved text + source file path for audit (e.g. policy.pdf → page 12)

**Visual Inspection:**
- Print top-scoring snippets to console for manual check

**Failure Modes:**
- “Empty retrieval” → wrong embedding model or low similarity
- “Irrelevant context” → domain drift → re-train embeddings

**Workshop Tip: enable --verbose in RagAgent to view the full retrieval + reasoning chain**

# Local Vector Stores

### Lightweight Setup
- SQLite + FAISS → simple, fast, minimal dependencies

### Intermediate Options
- ChromaDB – Python/HTTP API, good for demos
- Milvus / Weaviate – scalable, production-grade

### Storage Model
- Each vector row = embedding + metadata (title, timestamp, source)
- Indexed by ID for O(1) snippet lookup

### Persistence
- Full control → define retention, expiry, or encryption at rest

### Java Integration
- Use org.sqlite or JDBC FAISS wrapper for local retrieval

### Workshop Context
- Stage-3 repo seeds a local SQLite + FAISS index for policy.pdf

# Exercise: “Embabel + RAG Connector”

**Objective: extend your agent to query the provided policy.pdf**

**Setup Steps**

- 1. Start the mcp-rag-server from the workshop repo
- 2. Run RagAgent.java in your IDE or CLI
- 3. Configure connector URL (ws://localhost:3000)

**Task**

- prompt → “Summarise key compliance risks.”

**Expected Flow**

- agent.ask() → embedding lookup → retrieved snippets → grounded answer

**Observation Points**

- Inspect logs for MCP requests/responses
- Confirm citations or snippet IDs in output

**Stretch Goal**

- Replace policy.pdf with your own corpus; rebuild index using Embabel CLI tool

# Today

~~09:00 – 09:15 | Welcome & Overview (15 m~~
~~09:15 – 09:50 | Environment Setup & Veri~~
~~**09:50 – 10:30 | How Agents Work (40 min)~~**
~~10:30 – 11:00 | Morning Break (30 min)~~

~~**11:00 – 11:40 | Tool-Calling in Practice~~**
~~**11:40 – 12:30 | Exercise – Your First Wo~~**
12:30 – 13:30 | Lunch Break (60 min)

~~13:30 – 13:40 | Recap & Transition to MC~~
~~**13:40 – 14:20 | Model Context Protocol (~~**
~~**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)~~**
15:00 – 15:30 | Afternoon Break (30 min)

**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

# Today's Agenda

~~09:00 – 09:15 | Welcome & Overview (15 min)~~
~~09:15 – 09:50 | Environment Setup & Verification (35 min)~~
~~**09:50 – 10:30 | How Agents Work (40 min)**~~
~~10:30 – 11:00 | Morning Break (30 min)~~

~~**11:00 – 11:40 | Tool-Calling in Practice (40 min)**~~
~~**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**~~
12:30 – 13:30 | Lunch Break (60 min)

~~13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)~~
~~**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**~~
~~**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**~~
~~15:00 – 15:30 | Afternoon Break (30 min)~~

**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

# Message Flow

**Discovery phase**

- `client → request_tool_list`
- `server → [tool_metadata…] (JSON schema + signature)`

**Invocation phase**

- `client → execute_tool(id,args)`
- `server → response(result) or error(code,msg)`

**Bidirectional parity: both tools and agents can act as MCP servers (exposing capabilities) or clients (invoking others)**

**Parallels RPC: semantics identical to JSON-RPC 2.0—method name, parameters, result**

**Streaming support: incremental partial_result messages for long-running tasks**

**Enterprise practice: include correlation IDs and tracing headers (OpenTelemetry, Zipkin) for audit trails**

# Transport Options

**HTTP (S):**

- Simplest setup; one request = one tool invocation
- Easy integration with RESTful back-ends
- Drawback: lacks streaming and push events

**WebSocket:**

- Persistent bidirectional channel
- Ideal for interactive agents or tool streaming (e.g. token updates)
- Embabel uses this by default for local dev

**gRPC:**

- Binary protocol with protobuf schemas
- Efficient for high-frequency calls; strong typing

**UNIX Domain Sockets / Named Pipes:**

- For on-device LLMs (Ollama, LM Studio) with minimal latency

**Message Brokers (MQ, Kafka):**

- Experimental—adds queuing and reliability
- All transports serialise the same MCP message schema, so implementations remain interchangeable

# Heterogeneous Models

## Goal

- Balance cost, latency, and capability by using the right model per role

## Example Configuration (examples)

- Qwen3:30b-a3b → deep reasoning & planning
- Gemma3 → fluent summarisation and language output
- Granite4 → lightweight retrieval / fast chat

## Embabel YAML

- Specify model per agent under model: key

## Local Deployment

- All can run via Ollama, ensuring private inference

## Mixed Precision

- CPU for lighter models, GPU for heavy planning loops

## Workshop Relevance

- Mirrors Tripper's configuration where distinct agents use specialised models for itinerary generation, cost optimisation, and itinerary narration

# Embabel Team Configuration

```
agents:
  - name: researcher
    model: qwen3
  - name: summariser
    model: gemma2
links:
  - from: researcher
    to: summariser
    channel: MCP
```

**Defines logical topology**

- Who talks to whom and how

**Channels**

- wMCP (default), shared memory, or local queue

**Automatic Discovery**

- wEmbabel resolves agent endpoints and initialises connectors

**Thread Model**

- Each agent runs asynchronously in a separate thread pool

**Observability Hooks**

- Logs messages and tool invocations with correlation IDs

**Workshop Tie-In**

- Replicate this to link your RAG retriever agent with a planner and summariser, forming a micro-team

# Scaling & Performance

### Concurrency
- Use Java CompletableFuture or parallel streams for sub-tasks

### Thread Safety
- Isolate model sessions per agent to avoid token collisions

### Caching
- Memoise tool results (e.g., flight search) to reduce recomputation

### Resource Monitoring
- CPU/GPU load can spike under multi-agent orchestration
- Use Micrometer + Prometheus for live metrics

### Async Streaming
- Stream intermediate results via WebSocket to UI or logs

### Workshop Demo
- Tripper shows concurrent reasoning for multiple destinations — one thread per leg

# Enterprise Integration

**Framework Support**
- Deploy Embabel agents as Spring Boot or Quarkus microservices

**Networking**
- Secure communication via WebSocket (TLS) or HTTP + JWT

**Observability Stack**
- Logs → ELK or OpenSearch
- Metrics → Prometheus / Grafana dashboards

**Governance**
- Add policy layer to control tool use per agent role

**Fault Tolerance**
- Use Spring Retry / Circuit Breaker for tool timeouts

**Workshop Bridge**
- Same approach scales your RAG prototype to multi-tenant enterprise deployments

# Security & Auditability

### Full Trace Logging
- Every message and tool invocation persisted

### Cryptographic Integrity
- Sign requests (Ed25519 / HMAC-SHA256)

### Encryption
- Data at rest (AES-GCM) and in transit (TLS 1.3)

### Replay Logs
- Reproduce reasoning chains for regulatory review

### Access Control
- Define per-agent credentials; principle of least privilege

### Compliance Readiness
- Aligns with ISO 27001 / GDPR data handling for AI systems.

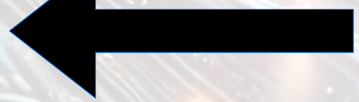
# Today’s Agenda

~~09:00 – 09:15 | Welcome & Overview (15 min)~~
~~09:15 – 09:50 | Environment Setup & Verification (35 min)~~
~~**09:50 – 10:30 | How Agents Work (40 min)**~~
~~10:30 – 11:00 | Morning Break (30 min)~~

~~**11:00 – 11:40 | Tool-Calling in Practice (40 min)**~~
~~**11:40 – 12:30 | Exercise – Your First Working Agent (50 min)**~~
12:30 – 13:30 | Lunch Break (60 min)

~~13:30 – 13:40 | Recap & Transition to MCP + Embabel (10 min)~~
~~**13:40 – 14:20 | Model Context Protocol (MCP) Deep Dive (40 min)**~~
~~**14:20 – 15:00 | Agentic RAG and Data Integration (40 min)**~~
~~15:00 – 15:30 | Afternoon Break (30 min)~~

~~**15:30 – 15:55 | Multi-Agent Teams & Heterogeneous Models (25 min)**~~
**15:55 – 16:20 | Enterprise Patterns & Deployment (25 min)**
16:20 – 16:30 | Wrap-Up & Discussion (10 min)

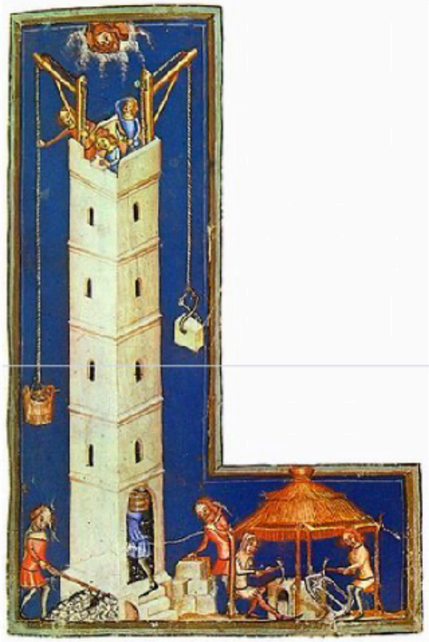


# Embabel – Clone the Github repo

**https://github.com/embabel**

## Embabel Agent Framework

KOTLIN JAVA SPRING APACHE TOMCAT APACHE MAVEN CHATGPT JINJA JSON
GITHUB ACTIONS SONARQUBE DOCKER INTELLIJIDEA



Framework for authoring agentic flows on the JVM that seamlessly mix LLM-prompted interactions with code and domain models. Supports intelligent path finding towards goals. Written in Kotlin but offers a natural usage model from Java. From the creator of Spring.

### Getting Started

There are two GitHub template repos you can use to create your own project:

- Java template - https://github.com/embabel/java-agent-template
- Kotlin template - https://github.com/embabel/kotlin-agent-template

Or you can use our project creator to create a custom project:

# Python vs Java – The Chasm

**Pro Python**

- Python is great for teaching - it's easy to read, compact (no excessive brackets or semicolons etc.)
- With Python I can read a file, process it and plot is in a graph in just a few line of code
- Python runs everywhere and easy to install - OK packages can be difficult sometimes
- Python libraries can be written in C/C++ for performance - Great for GPU access

**Against Python**

- It's SLOOOOW 🐢
- It can crash unexpectedly, and debugging stack traces isn't always helpful.
- Hard to manage in production - environments, dependencies, and scaling are tricky.

## THE CHASM

**Pro Java**

- The enterprise workhorse – it replaced mainframes for most mission-critical workloads.
- Virtually every bank and large enterprise runs Java on Linux.
- It's reliable, scalable, and fast – the foundation of production systems.

# Challenges with Gen AI in Enterprise

## Unavoidable Technical Challenges

- Non-determinism is now the norm, not the exception
- LLM hallucinations create reliability concerns
- Prompt engineering is alchemy, not true engineering
- Cost and environmental implications

## Organisational Challenges

- Top-down board mandates without developer buy-in
- Siloed AI teams disconnected from the business
- Greenfield fallacy: ignoring existing systems
- Cannot rollback mistakes like in development
  - **Example: Air Canada chatbot case**
  - **https://www.cbsnews.com/news/aircanada-chatbot-discount-customer/**

MoneyWatch

**Air Canada chatbot costs airline discount it wrongly offered customer**

By Megan Cerullo

February 19, 2024 / 1:05 PM EST / CBS News

# What is Embabel?

**JVM-based AI agent framework**

- Created by Rod Johnson (founder of Spring Framework) for building production-ready AI agents in Java and Kotlin

**Mixes LLM interactions with traditional code**

- Actions can invoke LLMs for structured output or execute pure code transformations through strongly-typed domain models

**Built on Spring AI ecosystem**

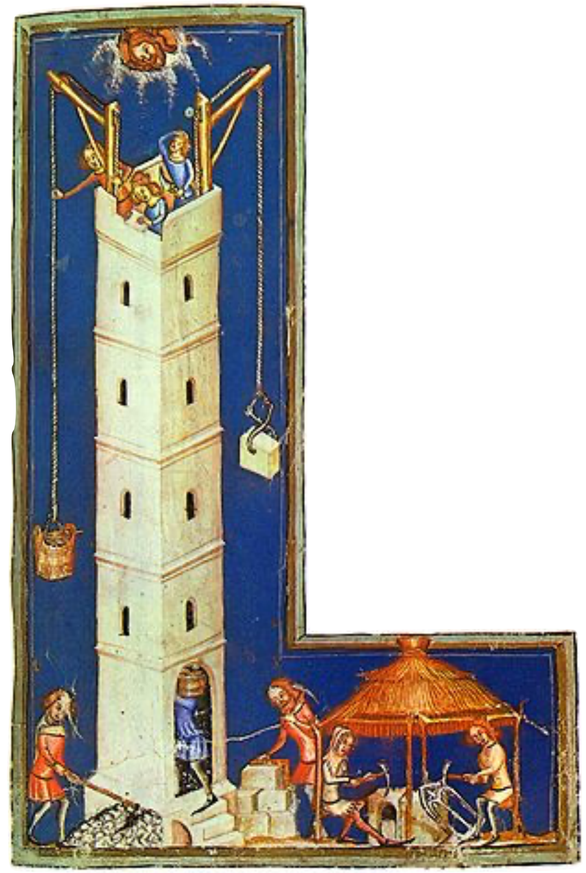
- Leverages Spring's enterprise capabilities while providing higher-level abstractions for complex agentic workflows

**Dynamic planning with continuous adaptation**

- Automatically replans after each action execution, forming an OODA loop (Observe-Orient-Decide-Act) that adapts to new information

**Enterprise-grade and production-ready**

- Fully unit testable like Spring beans, with comprehensive testing support and event-driven observability



# Embabel - key features

**Uses GOAP algorithm instead of LLM-based planning**

- Leverages Goal-Oriented Action Planning (from game AI) for deterministic, cost-optimised path finding rather than relying on LLMs for planning

**Novel path discovery**

- Can combine known actions in unanticipated ways not explicitly programmed, enabling emergent solutions to achieve goals

**Strong typing throughout**

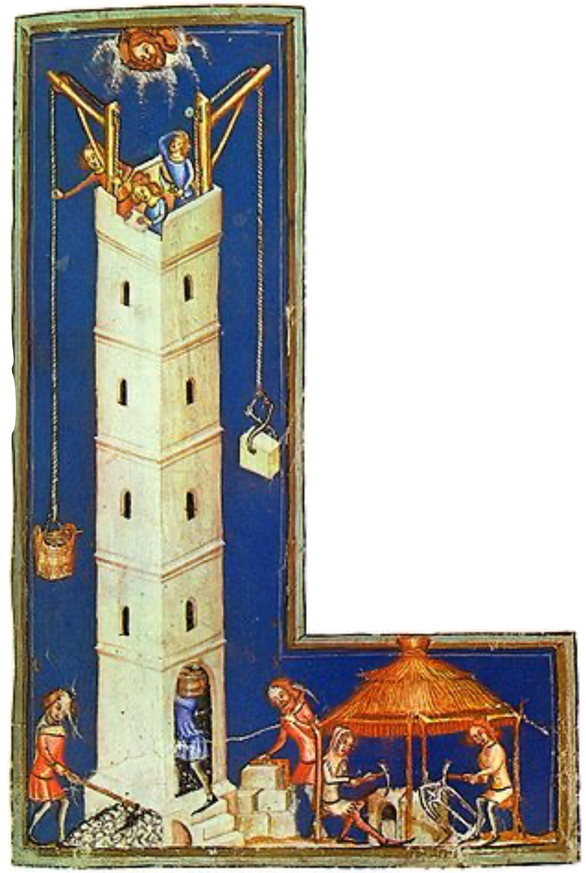
- All LLM interactions are strongly-typed with compile-time checking, refactoring support, and IDE assistance

**Reduced unnecessary LLM calls**

- Evaluates preconditions and effects to find optimal action sequences, improving efficiency and cost-effectiveness

**Extensible without modification**

- Can add new domain objects, actions, goals, and conditions without changing existing code, supporting parallelisation when appropriate



# Core Features of Embabel

## Built for Enterprise JVM

- Framework designed specifically for JVM-based systems
- Directly integrates with critical business technology
- Builds on Spring AI foundation
- Modern API with excellent tool support

## Server Architecture

- More than a framework: it's a server managing agents
- Knows about all deployed capabilities
  - **Extend via actions and goals**
- Supports long-running processes
- Built for testability and toolability

# Embabel - High-Level Architecture recap

**We'll cover these points very shortly…**

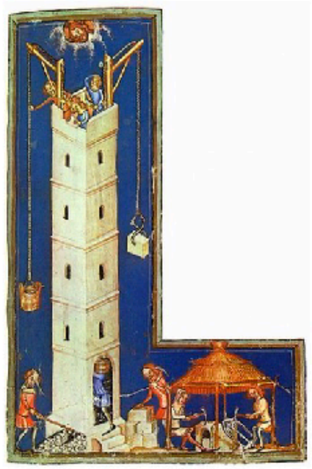
- Domain model layer
- Agent abstraction
- Actions
- Goals & Conditions
- Planner / Planning Engine - Goal Oriented Action Planning (GOAP) Layer
- Execution Modes & Platform
- Integration with Spring / Enterprise Java Stack
- LLM / Tool / External System Integration
- Testing, Observability & Safety
- Extensibility and Reuse

## The Embabel Travel Planner Agent: `https://github.com/embabel/tripper`

# Tripper: Embabel Travel Planner Agent

Build passing

KOTLIN | SPRING | APACHE TOMCAT | APACHE MAVEN | CHATGPT | JSON | HTMX | GITHUB ACTIONS | DOCKER | INTELLIJIDEA



**Tripper** is a travel planning agent that helps you create personalized travel itineraries, based on your preferences and interests. It uses web search, mapping and integrates with Airbnb. It demonstrates the power of the Embabel agent framework.

**Key Features:**

- 🤖 Demonstrates Embabel's core concepts of deterministic planning and centering agents around a domain model
- 🌍 Illustrates the use of multiple LLMs (Claude Sonnet, GPT-4.1-mini) in the same application
- 🗺️ Extensive use of MCP tools for mapping, image and web search, wikipedia and Airbnb integration
- 📱 Modern web interface with htmx
- 🐋 Docker containerization for MCP tools
- 🚀 CI/CD with GitHub Actions

# Wrap-Up

## Achievements

- Built local intelligent agent (Stage 1–3)
- Integrated RAG for factual grounding
- Added multi-agent collaboration
- Deployed under secure, auditable architecture

## Demonstration Recap

- You've gone into the depths of the LLM and how they handle tool calling
- You've seen how tool calling is partly standardised the MCP but presents its own problems
- Tripper shows full agent lifecycle from plan → data retrieval → summarisation → presentation

## Skill Outcome

- I very much hope you have a better understanding of tool-calling and agentic AI flows
- It's early days, there's a lot more to come
- You have touched on some of the very latest tech today
- With Java & Embabel you have built and run an end-to-end enterprise-ready agentic system

w-jax by entwickler.de

John Davies
Incept5

**Please remember to rate this talk!**
**Thank you**

https://x.com/jtdavies

https://www.linkedin.com/in/jdavies/