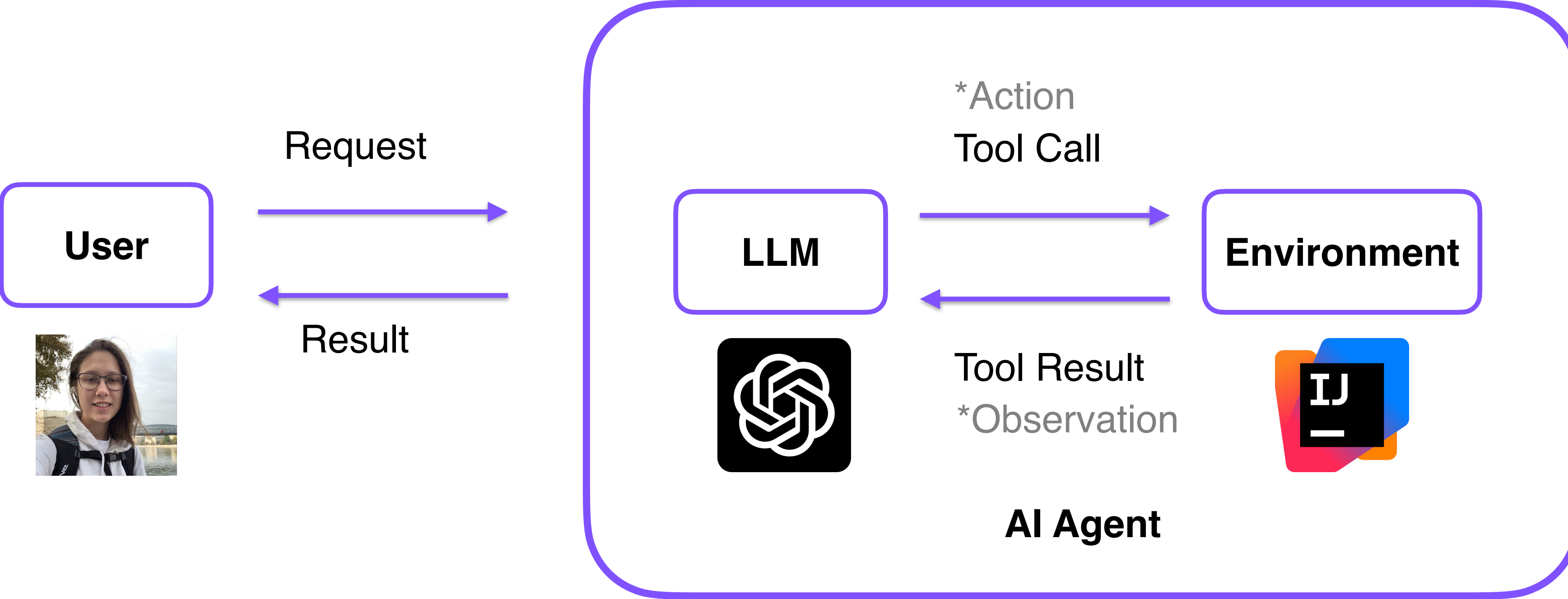


**JETBRAINS**

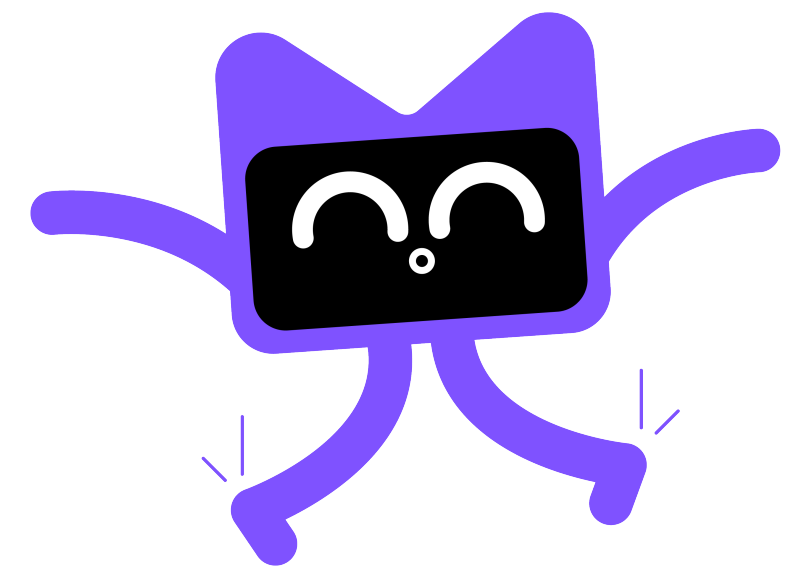
# AI Agents in the JVM World

Maria Tigina  
**Developer @ Koog, JetBrains**

# What is an AI Agent?



# History of AI Agents in JetBrains



# History of AI Agents

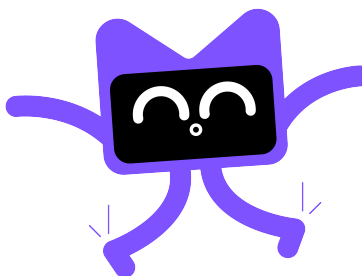
Dec 2022

- [ToolFormer](#) paper by Meta AI Research

Fine-tune GPT-J-6B to generate tool calls (qa, search, calculator, calendar, translation)

▸ Input: Out of 1400 participants, 400 passed the test. What is % of passed?

▸ Output: **[Calculator(400 / 1400)]%** passed the test.



# History of AI Agents

Dec 2022

- [ToolFormer](#) paper by Meta AI Research

Fine-tune GPT-J-6B to generate tool calls (qa, search, calculator, calendar, translation)



- [IdeFormer](#) paper by JetBrains Research

Use chat with LLM to trigger IDE actions (test/code running, inspections, renamings, refactorings, code search).

▸ Input: Where is the class ' AIAgent ' used?

▸ Output: **[Search(“AIAgent “)]**

# History of AI Agents

Dec 2022 | June 2023

- GPT-4 by OpenAI with function calling API

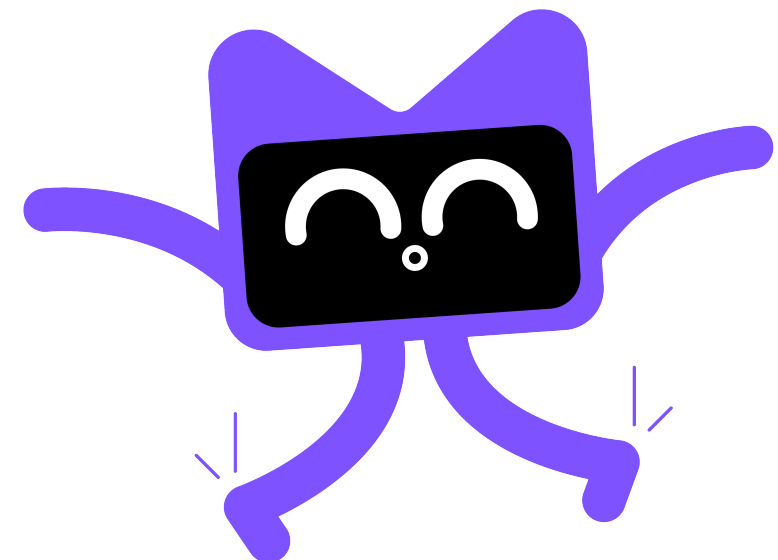
```
{
  "type": "function",
  "name": "get_weather",
  "description": "Retrieves current weather.",
  "parameters": {
    "type": "object",
    "properties": {
      "location": {
        "type": "string",
        "description": "City and country"
      }
    },
    "required": ["location"],
  }
}

{
  "id": "fc_12345xyz",
  "call_id": "call_12345xyz",
  "type": "function_call",
  "name": "get_weather",
  "arguments": "{\"location\": \"Paris, France\"}"
}
```

# History of AI Agents

Dec 2022 | June 2023 | **January 2024**

- LangChain releases [LangGraph](#), one of the first AI Agent frameworks
- Python first (TS/JS, LangChain4j later September 2025)
- Agent = state machine (functional strategy later)



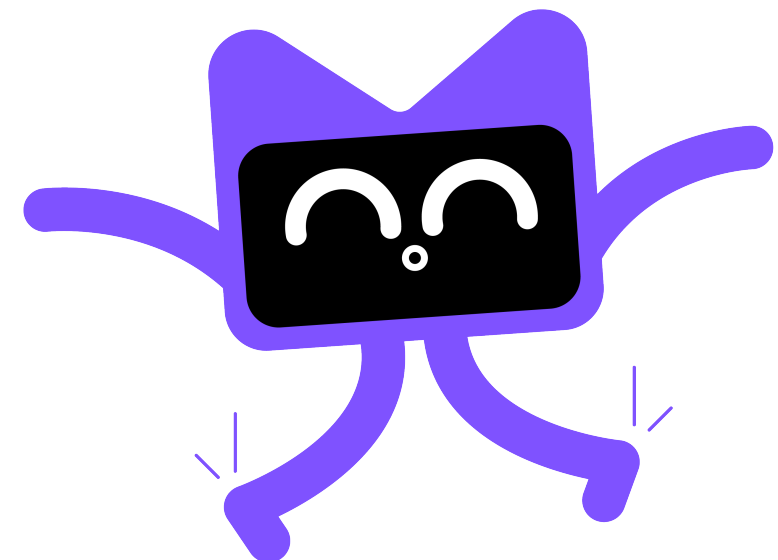
# History of AI Agents

Dec 2022 | June 2023 | **January 2024**

- LangChain releases [LangGraph](#), one of the first AI Agent frameworks
- Python first (TS/JS, LangChain4j later September 2025)
- Agent = state machine

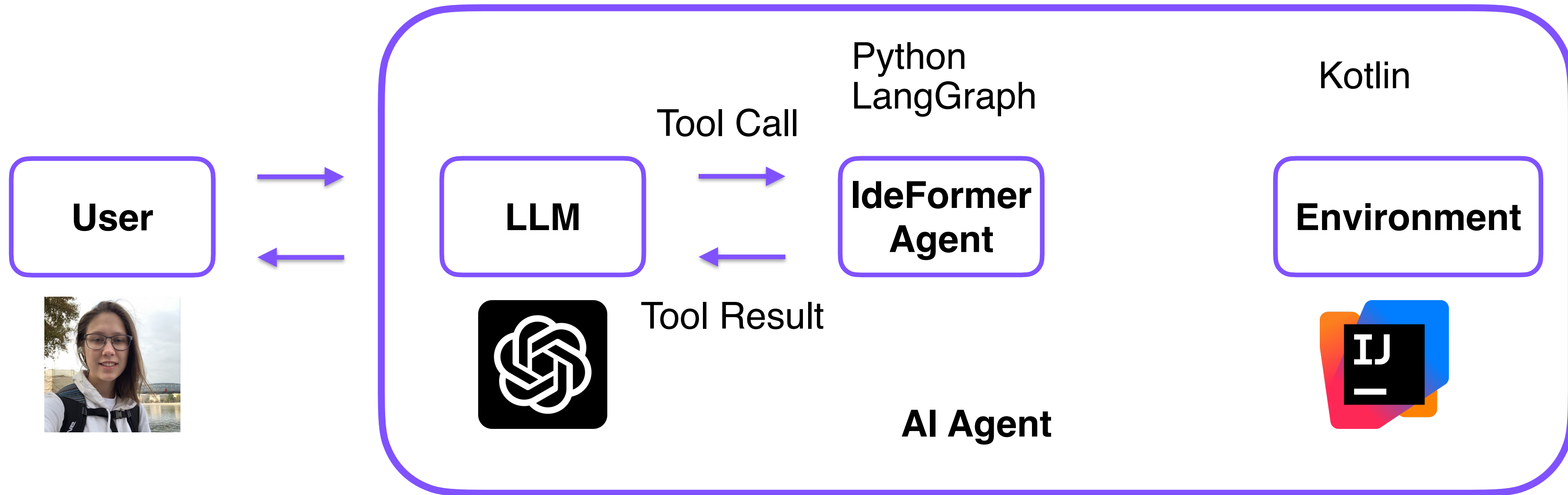


- IdeFormer as a platform for hosting LangGraph Agents



# History of AI Agents

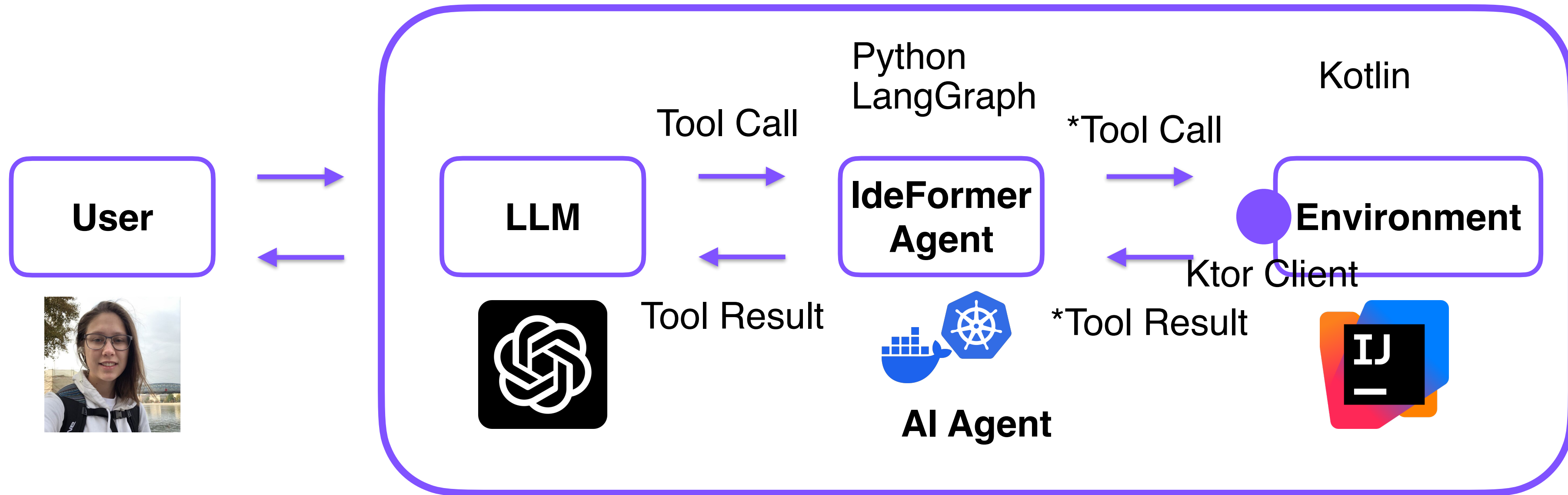
Dec 2022 | June 2023 | January 2024



- IDE is not a local tool or backend with an API which LangGraph working with

# History of AI Agents

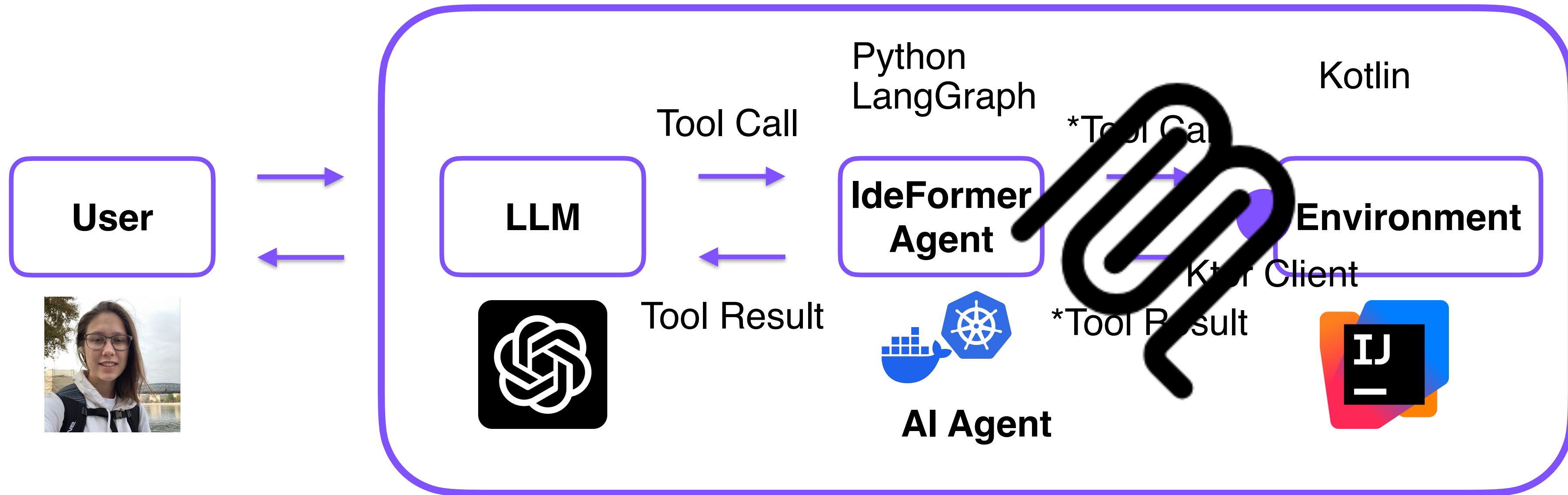
Dec 2022 | June 2023 | January 2024



- Kotlin IDE Client (triggers agent run in server, manages tool calls execution)
- Special protocol for IdeFormer and IDE Client communication

# History of AI Agents

Dec 2022 | June 2023 | January 2024



- Kotlin IDE Client (triggers agent run in server, manages tool calls execution)
- Special protocol for IdeFormer and IDE Client communication

# History of AI Agents

Dec 2022 | June 2023 | January 2024 | **2024-2025**

- Companies are actively building production-ready AI Agents
- Agentic protocols released (MCP, A2A)

# History of AI Agents

Dec 2022 | June 2023 | January 2024 | 2024 | **2025**

- Boom of coding agents (Codex, Claude Code, Cursor)



- Junie

# History of AI Agents

Dec 2022 | June 2023 | January 2024 | 2024 | **2025**

- Boom of coding agents (Codex, Claude Code, Cursor)



- Junie
- Koog, first JVM AI Agents framework

# History of AI Agents

Dec 2022 | June 2023 | January 2024 | 2024 | **2025**

- Boom of coding agents (Codex, Claude Code, Cursor)

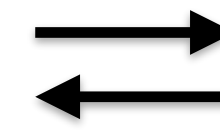


- Junie
- Koog, first JVM AI Agents framework

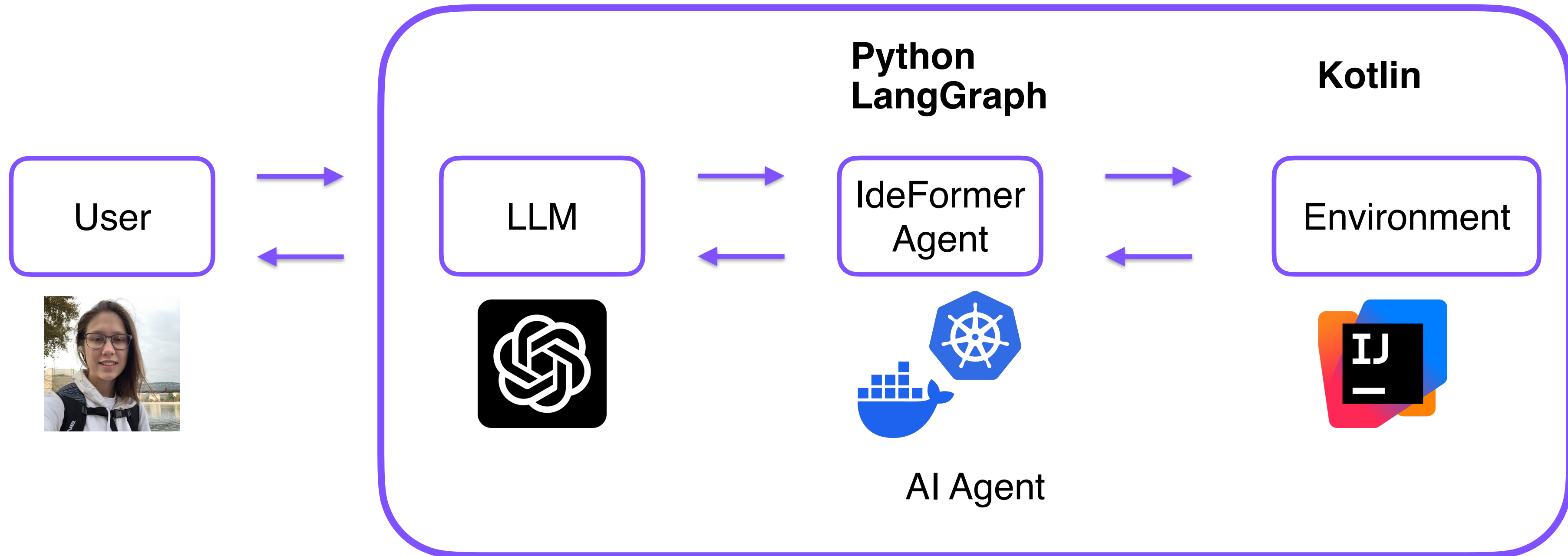


# History of AI Agents

5 ML Researchers



~100% Jetbarins  
developers



# History of AI Agents

Dec 2022 | June 2023 | January 2024 | 2024 | 2025 | **KotlinConf 2025**

- Koog was open-sourced
- 3.9k  on GitHub
- Several usages in production, even outside the JetBrains
- Coming to **KotlinConf 2026** with new updates

# Tools LangGraph

```
@tool
def search_database(query: str, limit: int = 10) → str:
    """Search the customer database for records matching the query.

    Args:
        query: Search terms to look for
        limit: Maximum number of results to return
    """
    return f"Found {limit} results for '{query}'"
```

# Tools Koog

```
@Tool("search_database")
@LLMDescription("Search the customer database for records matching the query.")
suspend fun searchDatabase(
    @LLMDescription("Search terms to look for.")
    query: String,
    @LLMDescription("Maximum number of results to return.")
    limit: Int = 10
): String = "Found ${limit} results of '${query}'."
```

# Graph strategy LangGraph

```
def call_llm(state: MessagesState):  
    return {"messages": [model_with_tools.invoke(state["messages"])]}  
  
def validate(state: MessagesState):  
    state["output"] = "0" if state["input"] == "" else state["input"]  
  
def should_continue(state: MessagesState) → Literal["execute_tools", "__end__"]:  
    last = state["messages"][-1]  
    return END if last.tool_calls else "validate"  
  
builder = StateGraph(MessagesState)  
builder.add_node("call_llm", call_llm)  
builder.add_node("validate", validate)  
  
builder.add_edge(START, "call_llm")  
builder.add_conditional_edges("call_llm", should_continue, ["validate", END])  
builder.add_edge("validate", END)  
  
graph = builder.compile()
```

# Graph strategy Koog

```
val strategy = strategy<Input, Output>("my-strategy") {  
    val callLLM by nodeLLMRequestMultiple()  
    val validate by node<Input, Output> {  
        input → if (input.isBlank()) "0" else input  
    }  
  
    edge(nodeStart forwardTo callLLM)  
    edge(callLLM forwardTo validate onAssistantMessage { true })  
    edge(callLLM forwardTo nodeFinish onToolCall { true } transformed { "0" })  
    edge(validate forwardTo nodeFinish)  
}
```

# Functional strategy LangGraph

```
@task
def call_llm(messages):
    return model_with_tools.invoke(messages)

@task
def call_tool(tool_call):
    return tools_by_name[tool_call["name"]].invoke(tool_call["args"])

@entrypoint()
def agent(messages):
    response = call_llm(messages).result()

    while response.tool_calls:
        tool_results = [call_tool(tc).result() for tc in response.tool_calls]
        messages = add_messages(messages, [response, *tool_results])
        response = call_llm(messages).result()

    return add_messages(messages, response)

result = agent.invoke([HumanMessage(content="What is 3 * 7 + 12?")])
```

# Functional strategy Koog

```
val strategy = functionalStrategy {  
    var responses = requestLLMMultiple(it)  
  
    while (responses.containsToolCalls()) {  
        val toolCalls = extractToolCalls(responses)  
  
        if (latestTokenUsage() > 10_000) {  
            compressHistory()  
        }  
  
        val results = executeMultipleTools(toolCalls)  
        responses = sendMultipleToolResults(results)  
    }  
  
    responses.single().asAssistantMessage().content  
}
```

# What to choose?

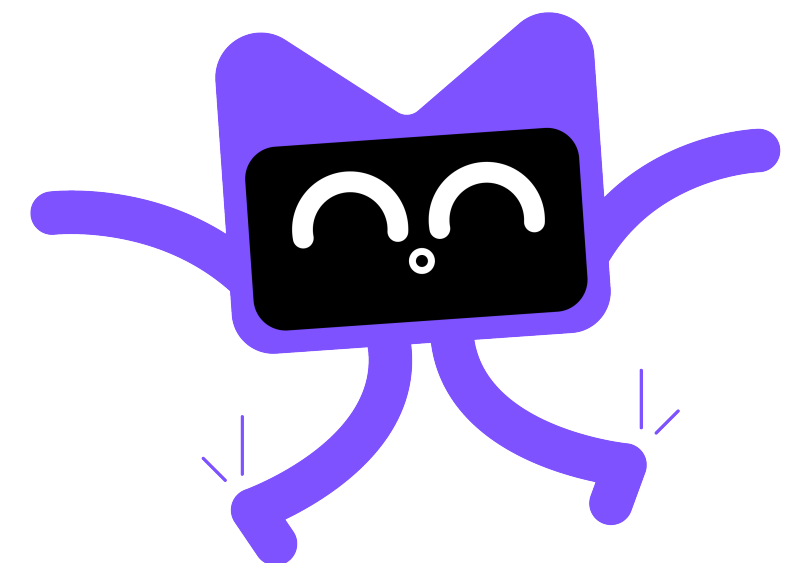
## LangGraph

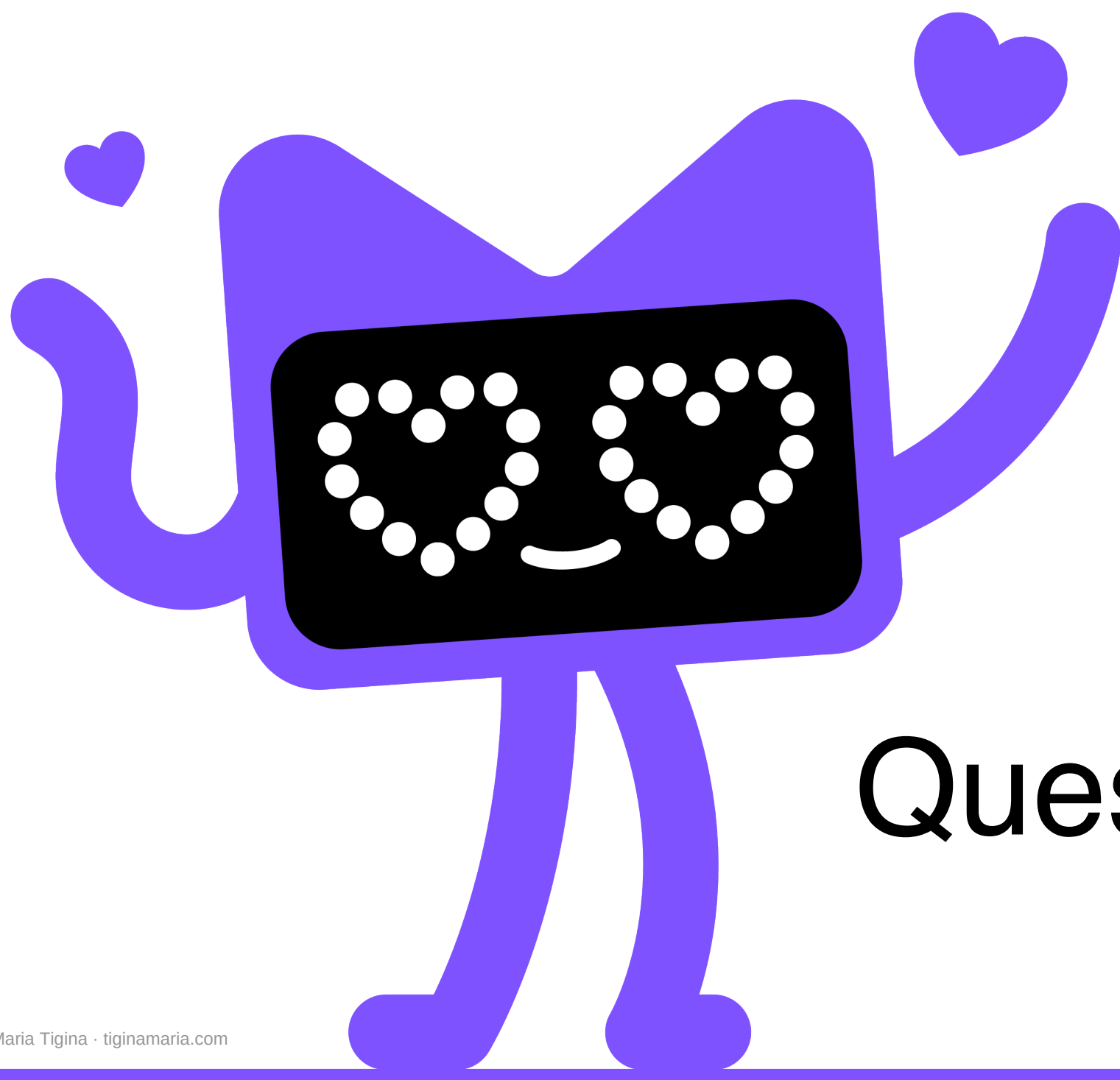
- Python/TS/JS
- Stable API
- Better LLM generation (Python & API)

## Koog

- Kotlin and Java (Multiplatform: JVM, JS, WASM)
- Safe (compiles = working, no shared mutable state)
- More ready-made components

- Common checkpoints approach
- Same tracing compatibilities (OTLP, Weave)





Questions?

