AI Agents

CS 346 Application Development

https://student.cs.uwaterloo.ca/~cs346

Contents

Introduction

Motivation

Imagine that you want your application to interact with an LLM to process some data e.g., to suggest grammatical changes in a text editor that you're building. How do you do this?

You need some essential abilities to interact with a model:

- You need to be able to connect to an existing model (local or online),
- You need to provide it a prompt, and return results. This likely needs to be asynchronous, since processing can take time.
- You need the ability to coordinate results, or provide information as the LLM is processing.

These are all things that an AI agent can do for you.

We talked about using Junie as a coding agent. If we want our application to use an LLM for different tasks, we likely need a better agent.

Koog

Introduction

> Koog is an open-source JetBrains framework for building AI agents designed specifically for the JVM ecosystem. It provides a first-class development experience for both Kotlin and Java developers, featuring an idiomatic, type-safe Kotlin DSL and fluent builder-style Java APIs.
>
> While Java developers can leverage the full power of Koog on the JVM using idiomatic APIs, Kotlin developers can also deploy agents across JS, WasmJS, Android, and iOS targets using Kotlin Multiplatform.
>
> — Koog Homepage [1]

Installation & Setup

To install Koog, add the dependency into your `module.yaml` file.

```
dependencies:
  - ai.koog:koog-agents:1.0.0
```

You will also need either a locally running model, or an API key for your choice of online model. Setup the API as recommended by that vendor. e.g.,

```
export OPENAI_API_KEY=your-api-key
```

> ⚠ **Warning**
>
> Avoid hardcoding API keys in the source code! Use environment variables to store your API keys.

What are agents?

AI agents are autonomous systems that can reason, make decisions, interact with the environment, and take actions to achieve your goal [1]

Koog agents are built around the following core concepts:

- A prompt executor manages and executes prompts, enabling the agent to interact with LLMs for reasoning and decision-making.
- A strategy defines the agent's workflow. It can be in the form of a directed graph, a function, or a planner.
- An agent can use tools to interact with external data sources and services.

Koog has a number of types of agents available:

- `Basic agents` for simple tasks that don't require any custom logic.
- `Graph-based agents` with customizable agent workflow, state management.
- `Functional agents` enable custom logic with access to the agent's context.
- `Planner agents` that can plan and execute multistep tasks through iterative cycles until they reach a desired final state.

Basic Agents

We can define an agent that will pass a prompt to our LLM

```
val key = System.getenv("OPENAI_API_KEY")
val agent = AIAgent(
    promptExecutor = simpleOpenAIExecutor(key),
    llmModel = OpenAIModels.Chat.GPT4o
)

fun main() = runBlocking {
    val result = agent.run("Hello! How can you help me?")
    println(result)
}
/* LLM output */
```

We could expand our prompt to get better results.

```
val agent = AIAgent(
    promptExecutor = simpleOpenAIExecutor(key),
    systemPrompt = "You are an expert in internet memes ...",
    llmModel = OpenAIModels.Chat.GPT4o
)
```

Tools

You can define external functionality and make it available to the agent. For example, you can give it the ability to ask the user a question, and consume the answer.

Create a tool by annotating a function with the `@Tool` annotation:

```
@Tool
@LLMDescription("Ask the user a question by sending it to stdout and
return the answer from stdin")
fun askUser(
    @LLMDescription("Question from the agent")
    question: String
): String {
    println(question)
    return readln()
}
```

Tools

Use the ToolRegistry to make this tool available to the agent.

```
val agent = AIAgent(
    promptExecutor = simpleOpenAIExecutor(key),
    systemPrompt = "You are an expert in internet memes. Be helpful,
friendly, and answer user questions concisely, showing your
knowledge of memes.",
    llmModel = OpenAIModels.Chat.GPT4o,
    temperature = 0.7,
    toolRegistry = ToolRegistry {
        tool(::askUser)
    }
)
```

Event Handler

```
val agent = AIAgent(
    promptExecutor = simpleOpenAIExecutor(key),
    systemPrompt = "You are an expert in internet memes. Be helpful,
friendly, and answer user questions concisely, showing your
knowledge of memes.",
    llmModel = OpenAIModels.Chat.GPT4o,
    temperature = 0.7,
    toolRegistry = ToolRegistry {
        tool(::askUser)
    },
    maxIterations = 10
){
    handleEvents {
        // Handle tool calls
        onToolCallStarting { eventContext →
            println("Called: ${eventContext.toolName} ...")
        }
    }
}
```

Event Handler

The agent will now output something similar to the following when it calls the `askUser` tool.

```
Called: askUser with args {"question":"Which meme would you like me to explain?"}
```

Bibliography

[1] JetBrains Inc., “Koog.” [Online]. Available: https://docs.koog.ai/