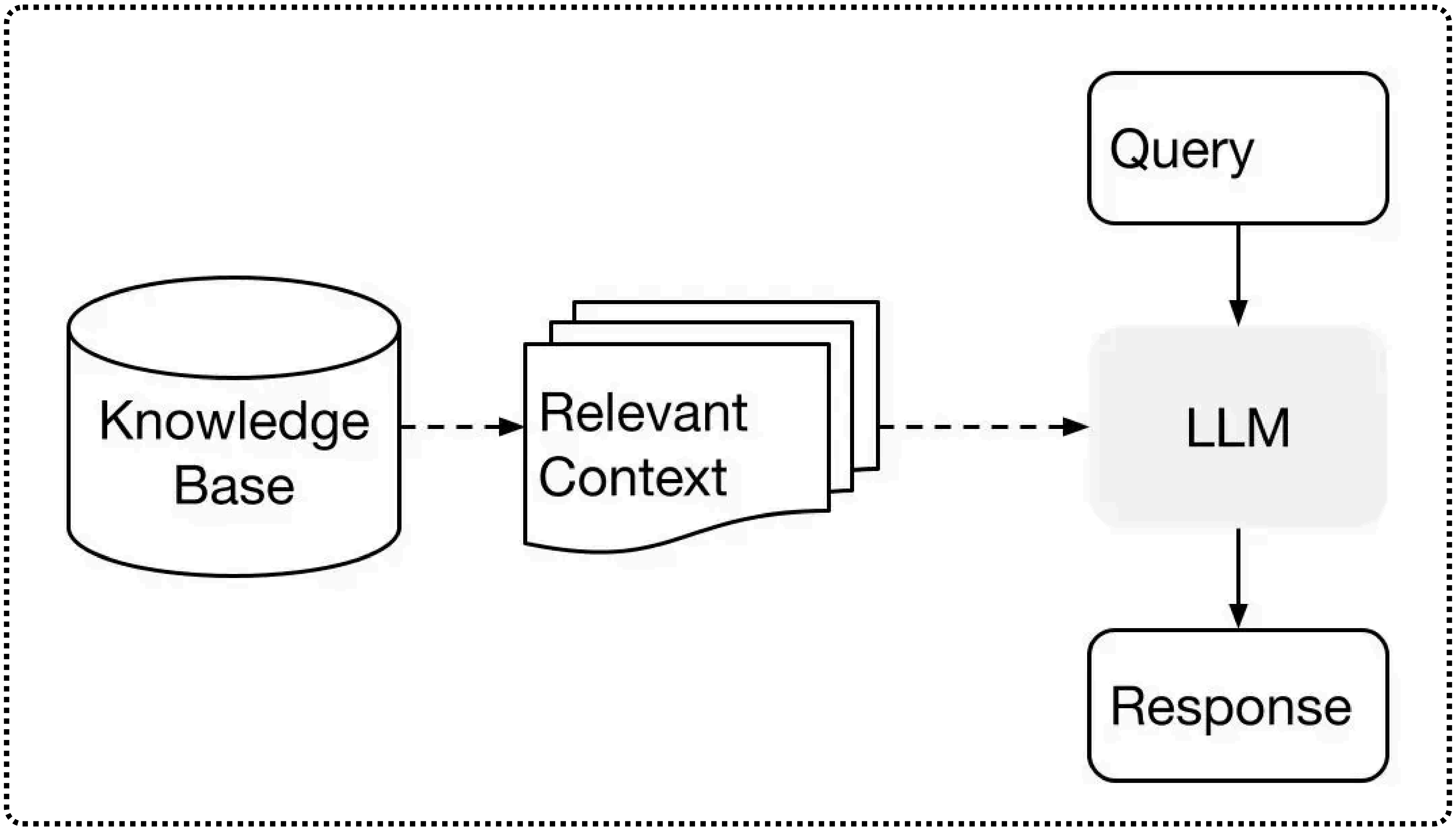
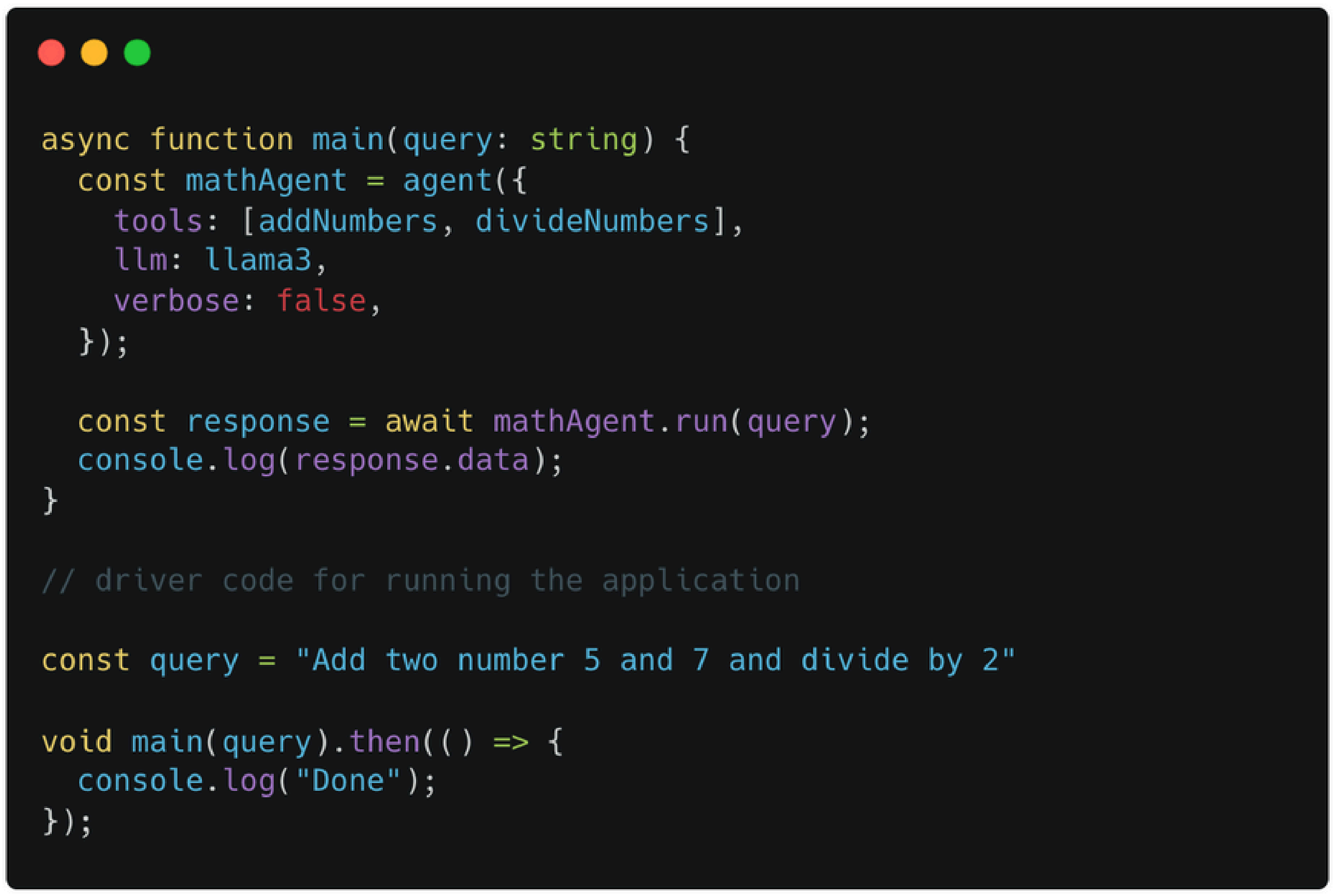
Guide to Agentic RAG Using LlamaIndex TypeScript





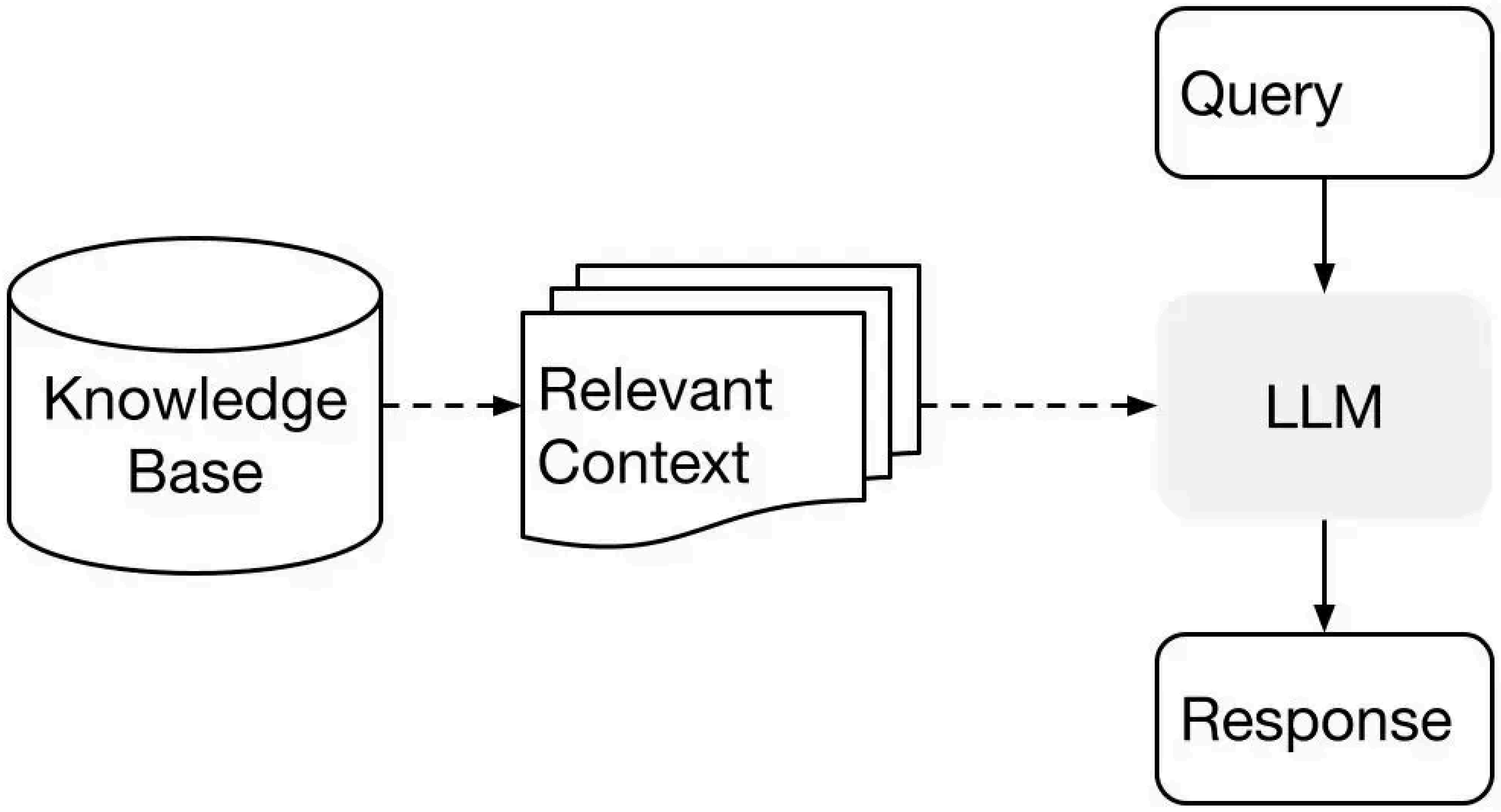
```
async function main(query: string) {
  const mathAgent = agent({
    tools: [addNumbers, divideNumbers],
    llm: llama3,
    verbose: false,
  });

  const response = await mathAgent.run(query);
  console.log(response.data);
}

// driver code for running the application

const query = "Add two number 5 and 7 and divide by 2"

void main(query).then(() => {
  console.log("Done");
});
```

We are offering a Free Course on RAG

What is Agentic RAG?

Before diving into implementation, let’s clarify what it means by Agentic RAG.

- RAG(Retrieval-Augmented Generation) is a technique that enhances language model outputs by first retrieving relevant information from a knowledge base, and then using that information to generate more accurate, factual responses.
- Agentic systems involve AI that can decide which actions to take based on user queries, effectively functioning as an intelligent assistant that chooses appropriate tools to fulfill requests.



Setting Development Environment

Install Node in Windows

To install Node into Windows follow these steps.

```
# Download and install fnm:
winget install Schniz.fnm

# Download and install Node.js:
fnm install 22

# Verify the Node.js version:
node -v # Should print "v22.14.0".

# Verify npm version:
npm -v # Should print "10.9.2".
```

A Simple Math Agent

Let’s create a simple math agent to understand the LlamaIndex TypeScript API.

Step 1: Set Up Work Environment

Create a new directory and navigate into it and Initialize a Node.js project and install dependencies.

```
$ md simple-agent
$ cd simple-agent
$ npm init
$ npm install llamaindex @llamaindex/ollama
```

We will create two tools for the math agent.

- An addition tool that adds two numbers
- A divide tool that divides numbers

Step 2: Import Required Modules

Add the following imports to your script:

```
import { agent, Settings, tool } from "llamaindex";
import { z } from "zod";
import { Ollama, OllamaEmbedding } from
"@llamaindex/ollama";
```

Step 3: Create an Ollama Model Instance

Instantiate the Llama model:

```
const llama3 = new Ollama({
  model: "llama3.2:1b",
});
```

Now using Settings you directly set the Ollama model for the system’s main model or use a different model directly on the agent.

```
Settings.llm = llama3;
```

Step 4: Create Tools for the Math Agent

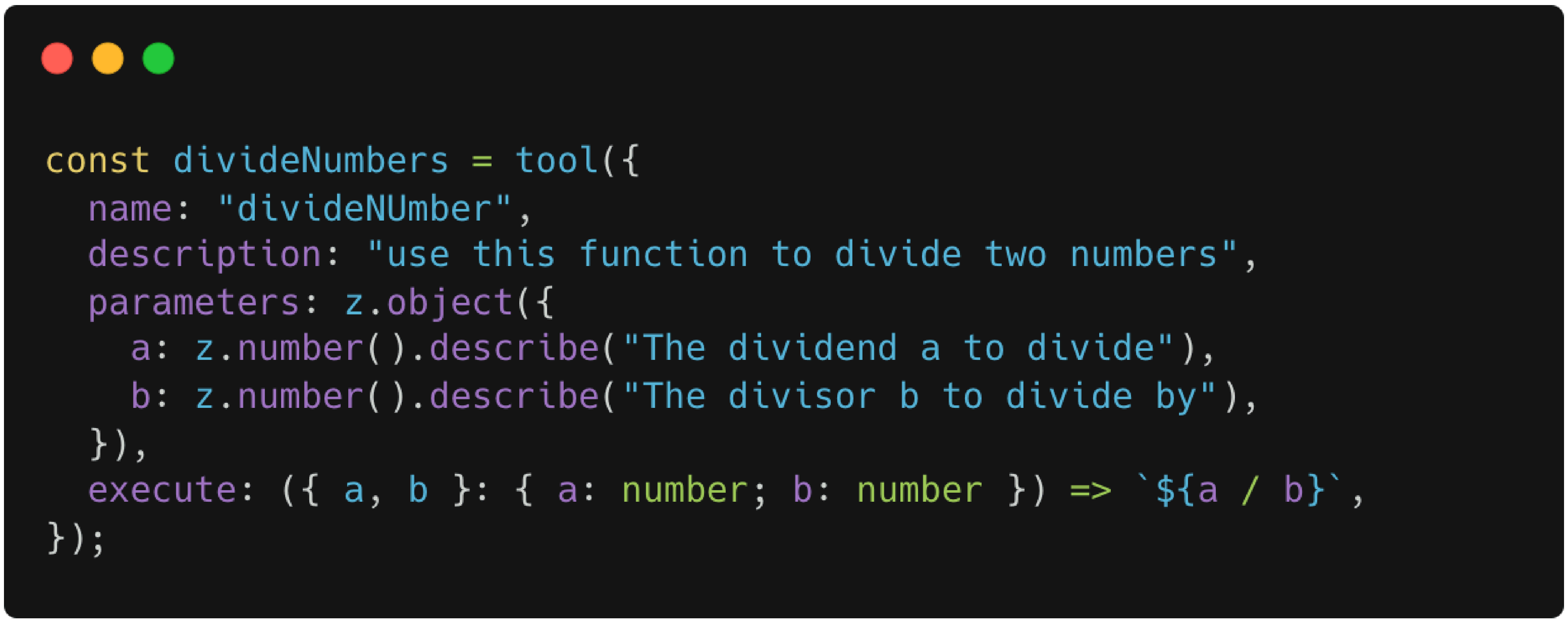
Add and divide tools

```
const addNumbers = tool({
  name: "SumNubers",
  description: "use this function to sun two numbers",
  parameters: z.object({
    a: z.number().describe("The first number"),
    b: z.number().describe("The second number"),
  }),
  execute: ({ a, b }: { a: number; b: number }) => `${a + b}`,
});
```

Here we will create a tool named addNumber using LlamaIndex tool API, The tool parameters object contains Four main parameters.

- name: The name of the tool
- description: The description of the tool that will be used by the LLM to understand the tool’s capability.
- parameter: The parameters of the tool, where I have used Zod libraries for data validation.
- execute: The function which will be executed by the tool.

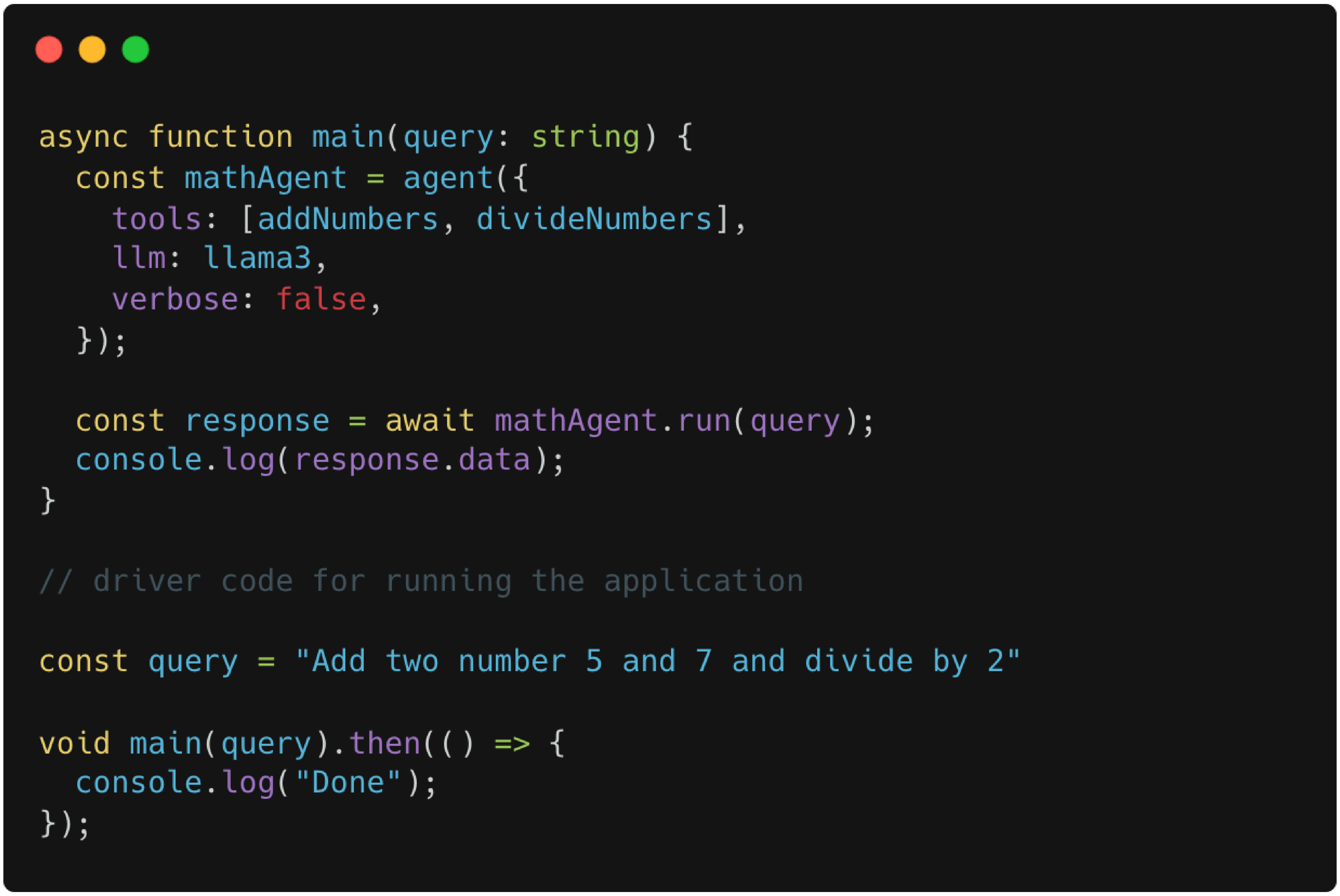
In the same way, we will create the divideNumber tool.



```
const divideNumbers = tool({
  name: "divideNUmber",
  description: "use this function to divide two numbers",
  parameters: z.object({
    a: z.number().describe("The dividend a to divide"),
    b: z.number().describe("The divisor b to divide by"),
  }),
  execute: ({ a, b }: { a: number; b: number }) => `${a / b}`,
});
```

Step 5: Create the Math Agent

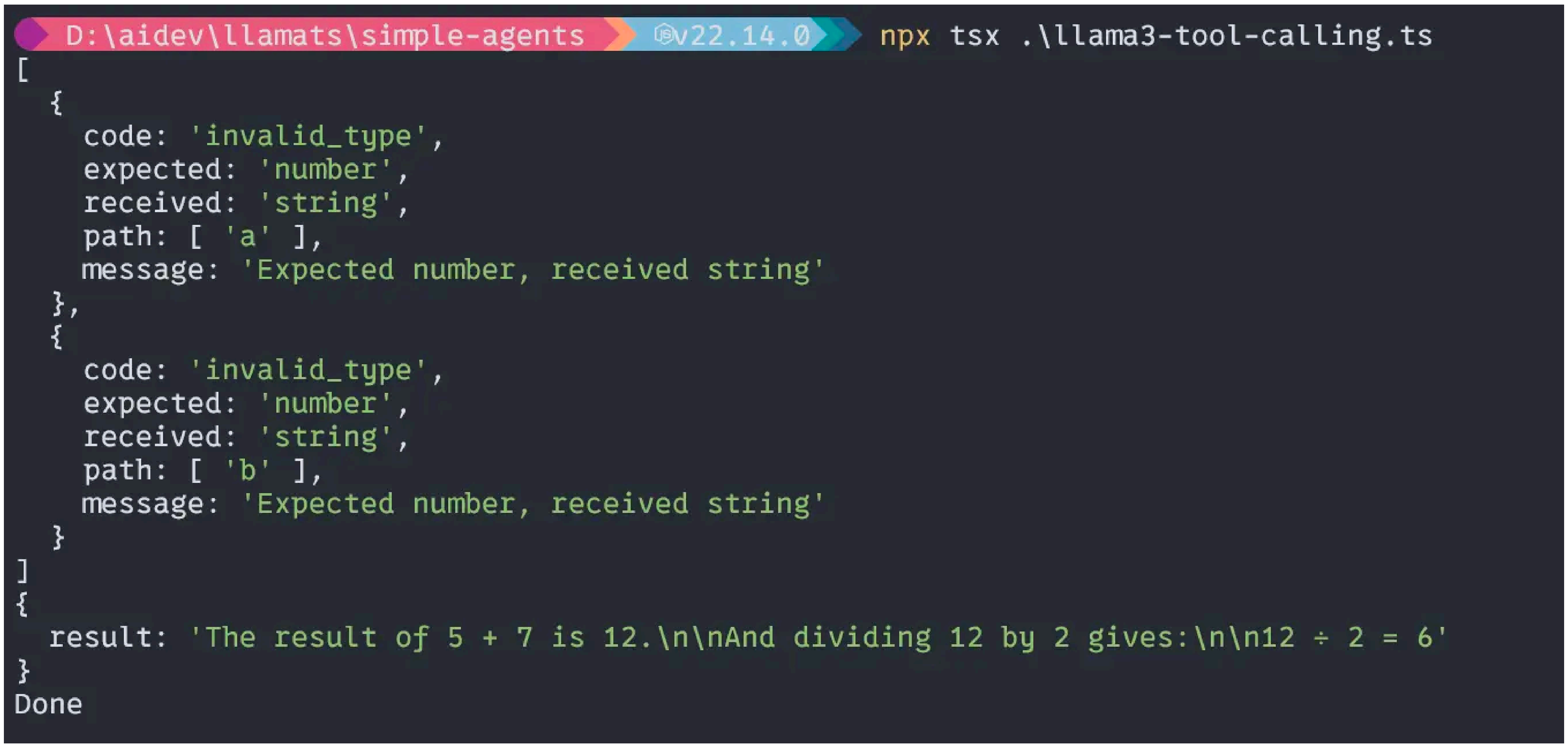
Now in the main function, we will create a math agent that will use the tools for calculation.



```
async function main(query: string) {
  const mathAgent = agent({
    tools: [addNumbers, divideNumbers],
    llm: llama3,
    verbose: false,
  });

  const response = await mathAgent.run(query);
  console.log(response.data);
}

// driver code for running the application

const query = "Add two number 5 and 7 and divide by 2"

void main(query).then(() => {
  console.log("Done");
});
```

If you set your LLM directly to through Setting then you don’t have to put the LLM parameters of the agent. If you want to use different models for different agents then you must put llm parameters explicitly.

After that response is the await function of the mathAgent which will run the query through the llm and return back the data.



```
D:\aidev\llamats\simple-agents  v22.14.0  npx tsx .\llama3-tool-calling.ts
[
  {
    code: 'invalid_type',
    expected: 'number',
    received: 'string',
    path: [ 'a' ],
    message: 'Expected number, received string'
  },
  {
    code: 'invalid_type',
    expected: 'number',
    received: 'string',
    path: [ 'b' ],
    message: 'Expected number, received string'
  }
]
{
  result: 'The result of 5 + 7 is 12.\n\nAnd dividing 12 by 2 gives:\n\n12 ÷ 2 = 6'
}
Done
```

For more information, kindly visit this article

Advanced Generative AI Guide LLMs RAG

Guide to Agentic RAG Using LlamaIndex TypeScript

Learn how to build an Agentic RAG Using LlamaIndex TypeScript with a step-by-step guide on setup, tool creation, and agent execution.

Avijit Biswas 04 Apr, 2025