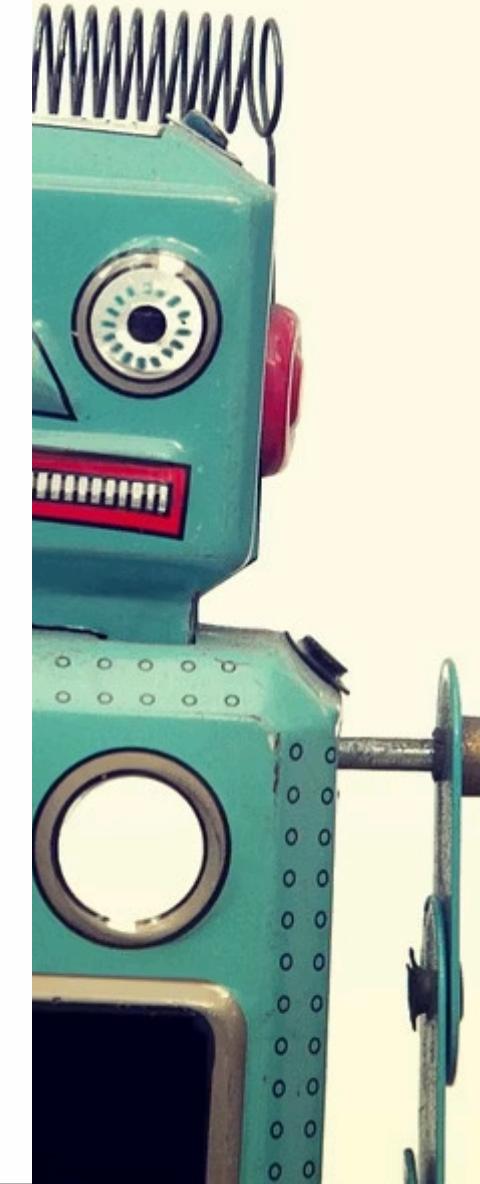


LLM Agents

Andrés Muñoz / Iván Ruiz



Contents

- Introduction
- Modular RAG
- Use of tools
- Collaboration
- IDEs for Agentic AI

Introduction

Evolution of Human-Computer Interaction

1 Command Line Interfaces

Basic text-based interaction, requiring specific technical knowledge.

2 Graphical User Interfaces

Improved accessibility with intuitive visual elements.

3 Touch Interfaces

Interaction through touch screens, revolutionising the mobile experience.

4 Virtual Assistants

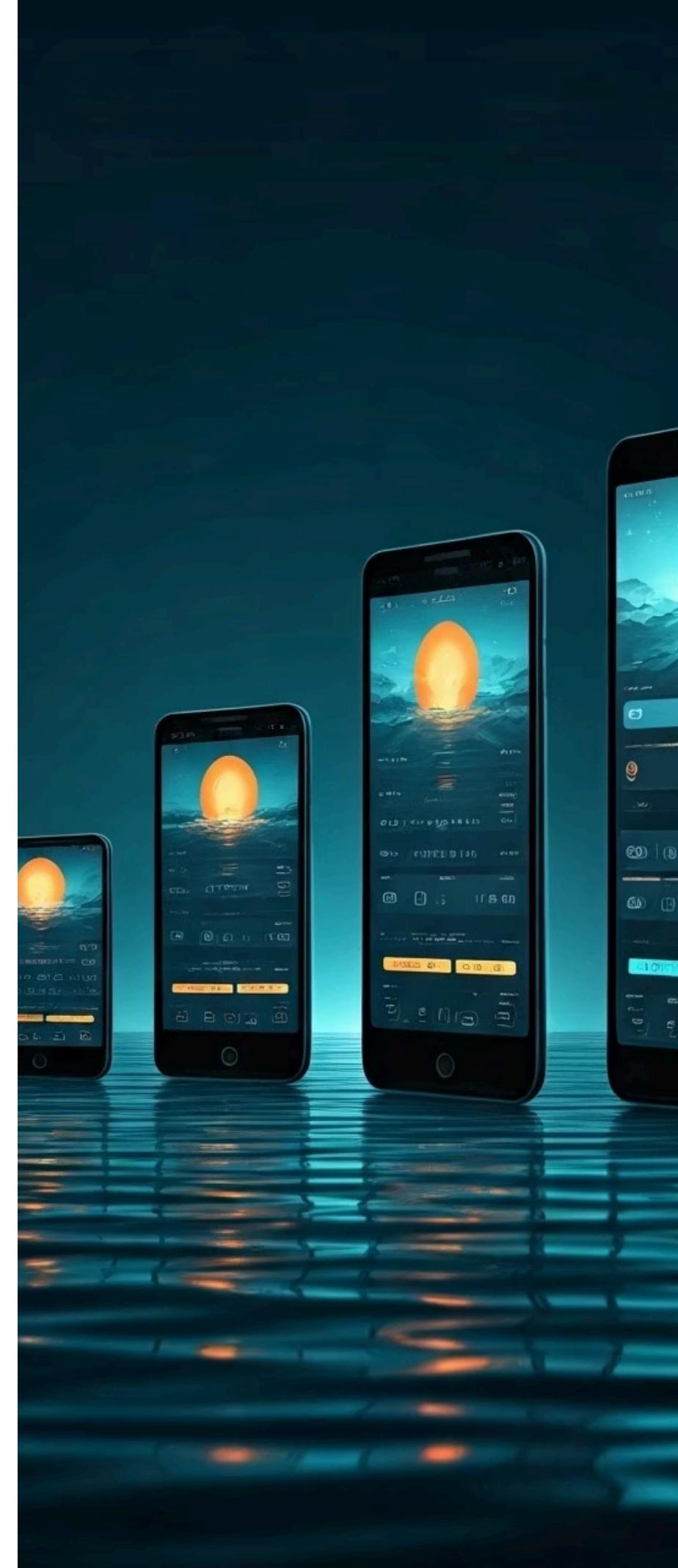
Introduction of voice interaction and natural language processing.

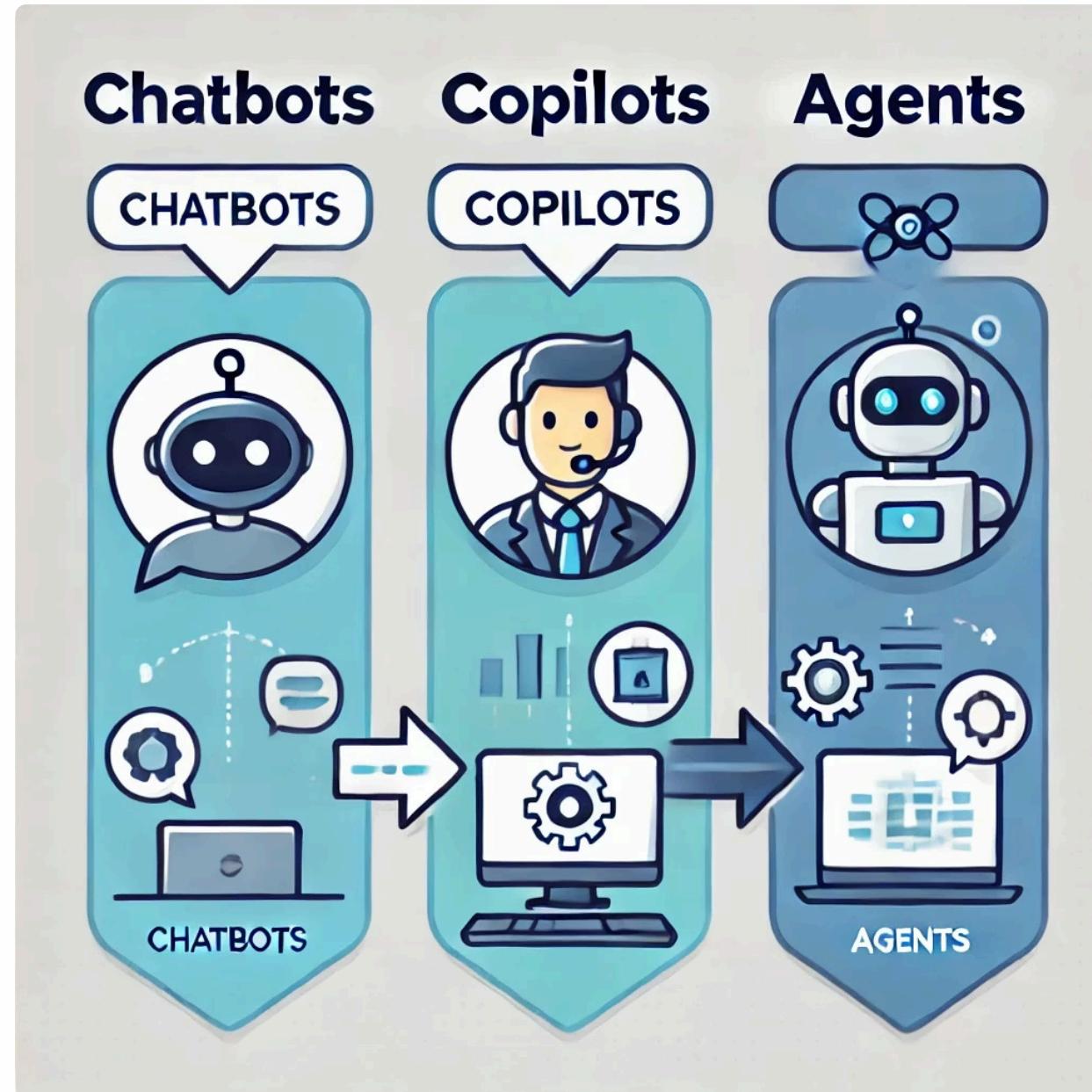
5 Augmented and Virtual Reality

Integration of virtual elements into the real world for immersive experiences.

6 AI-based Chatbots

Assistants, co-pilots and agents that enable natural conversation and execution of complex tasks.







Chatbots

Definition

Software designed to imitate human conversations, mainly through text. Its motivation is to assist the user with a specific task.

Pre-LLM Chatbots

They are based on predefined responses and have limited capacity to handle complex or ambiguous queries.

LLM-based Chatbots

With generative AI, chatbots can better understand user questions and provide more accurate and natural responses.

Co-pilots

Features

- Real-time collaboration with the user
- Personalised recommendations based on context
- Continuous evolution through machine learning
- Seamless integration with the tools used daily

Benefits

- Optimisation of time and resources
- Minimisation of mistakes and errors
- Accelerated and effective learning





Agents



Autonomy

Capable of making independent decisions based on objectives and context.



Multifunctionality

Integrate various tools and APIs to perform complex tasks.



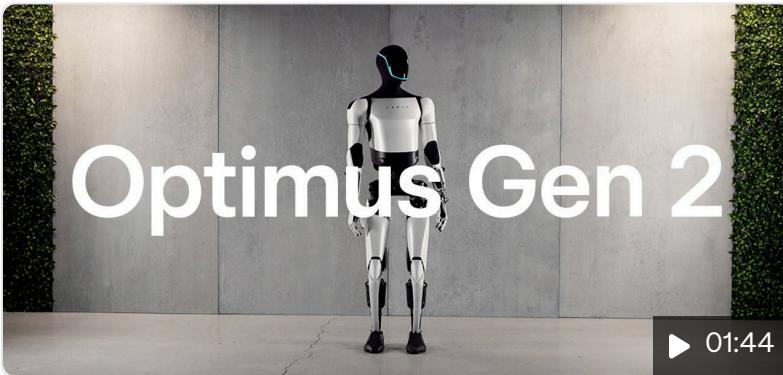
Collaboration

Interact with other agents and systems to solve problems in a coordinated manner.



Adaptability

Adjust to changes in the environment and improve over time.

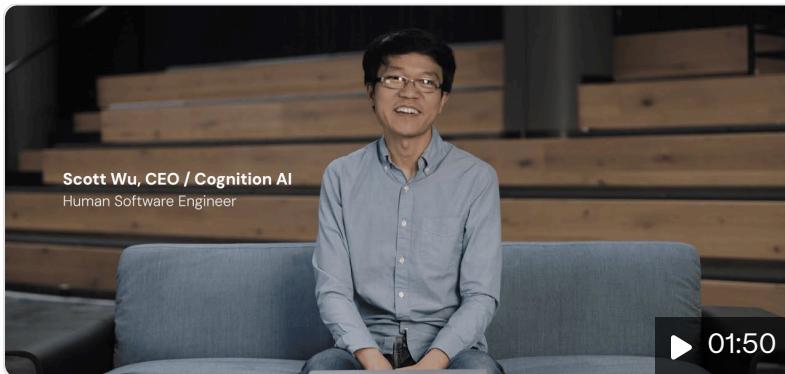


The thumbnail features a white humanoid robot standing in front of a concrete wall with green plants on either side. The text "Optimus Gen 2" is overlaid in large white letters. A play button icon and the duration "01:44" are at the bottom right.

 YouTube 

Optimus – Gen 2 | Tesla

New bot in town! Optimus Gen 2 features Tesla-designed actuators and sensors, faster and more capable hands, faster walking, lower total weight, articulated...

 YouTube

Introducing Devin, the first AI software engineer

Meet Devin, the world's first fully autonomous AI software engineer. Devin is a tireless, skilled teammate, equally ready to build alongside you or independently...

Characteristics of LLM Agents



Streaming Conversation

Fluid real-time responses, enhancing interaction.



Context Information

Access to internal and external data, via KAG, for more complete responses.



Contextual Memory

Remembers previous conversations for more accurate responses.



Tool Integration

Uses APIs and external services to expand its capabilities.



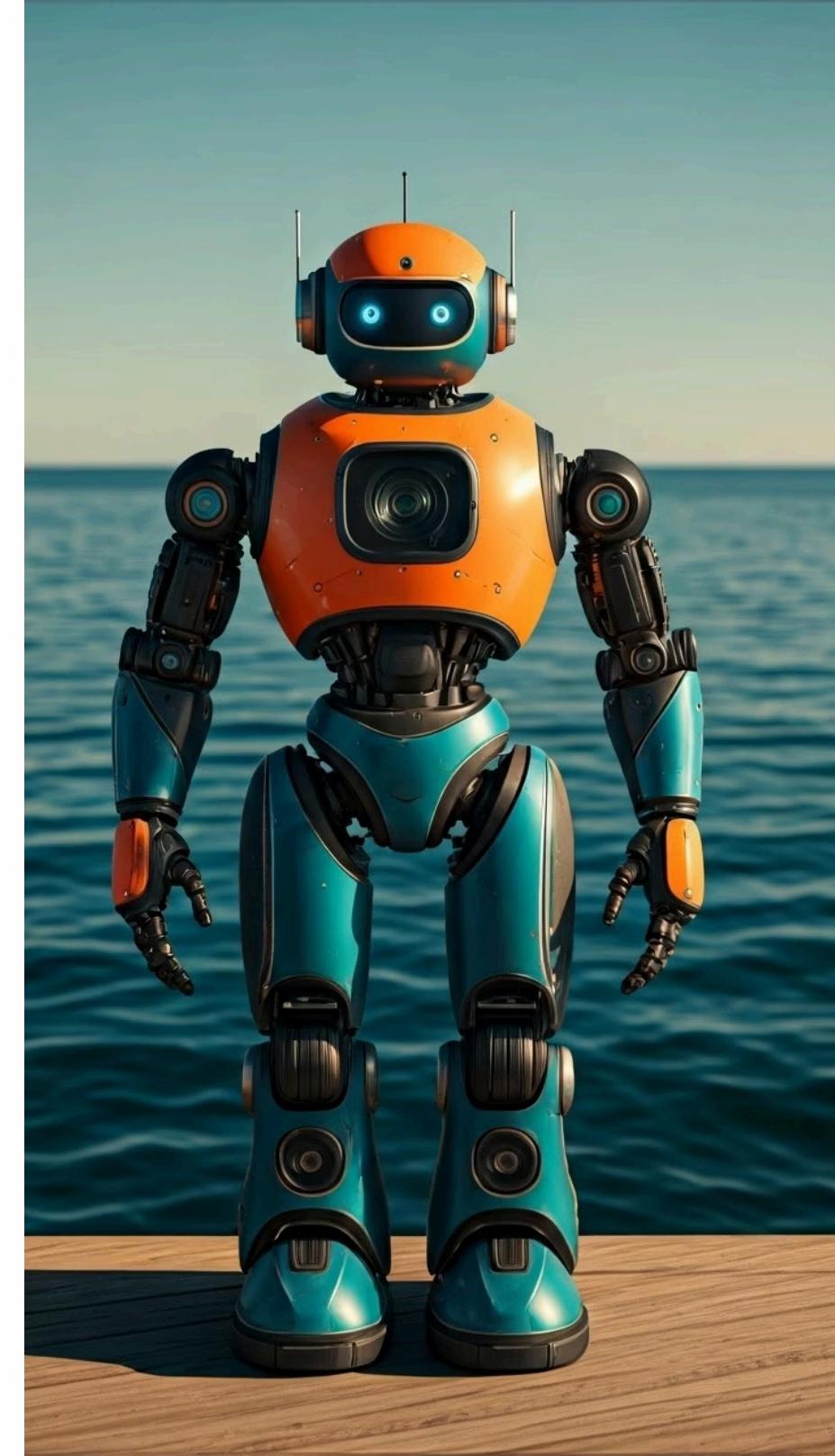
Collaboration between Agents

Teamwork with other agents to solve complex problems.



Adaptability

Adapts to changes and takes initiative.



The (human) process of writing a text



1 Plan an outline

Define the structure of the text to ensure coherence and clarity.

2 Research

Search for relevant information on the Internet to complement and enrich the content.

3 First draft

Write a first draft to organise the ideas and main content.

4

Review the draft

Identify weak arguments and remove superfluous information.

5

Rewrite

Rewrite the draft to correct the weaknesses identified.

6

Iterate

Repeat the previous steps to improve the quality of the text.



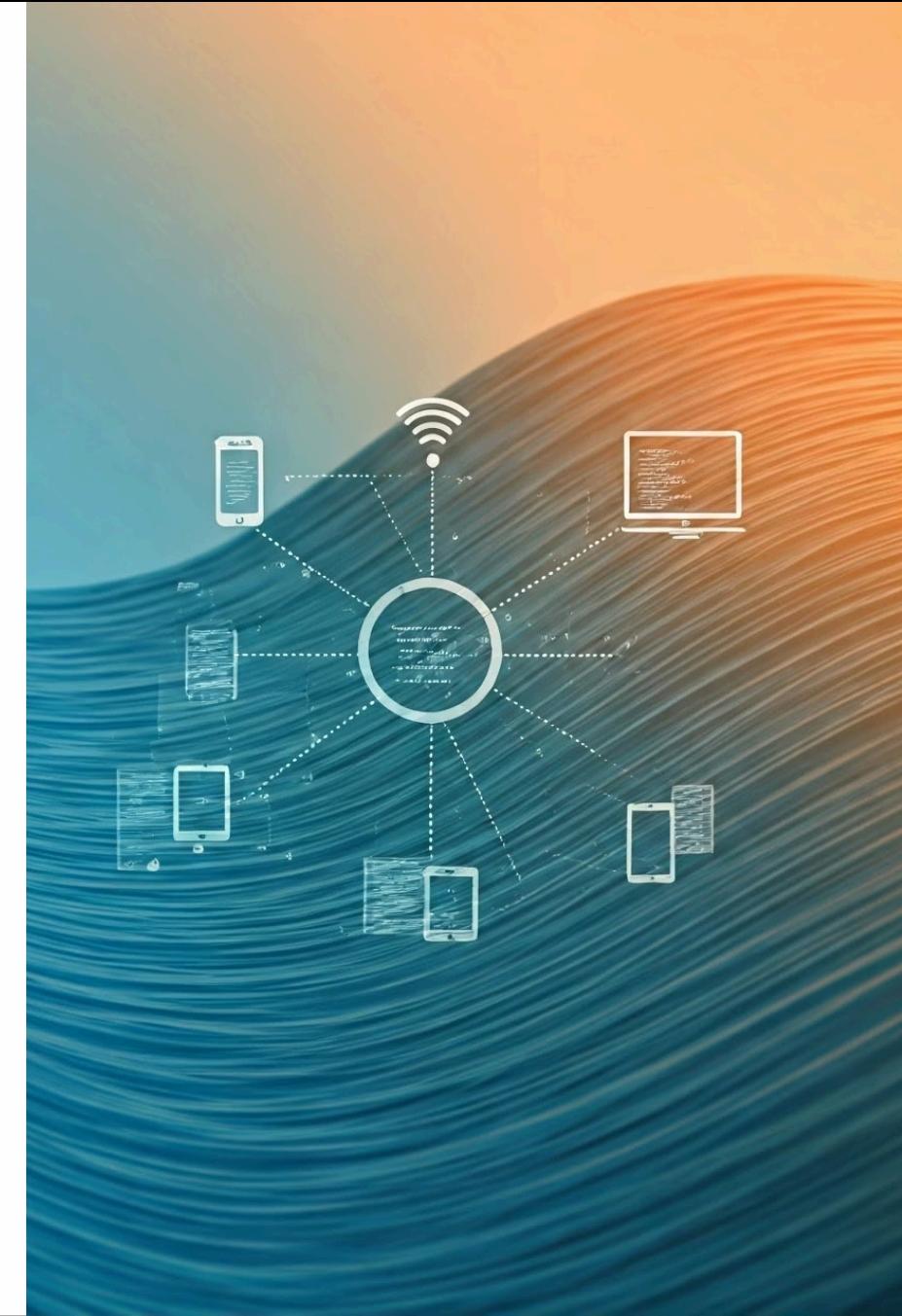
Use of Tools

LLM with Functions

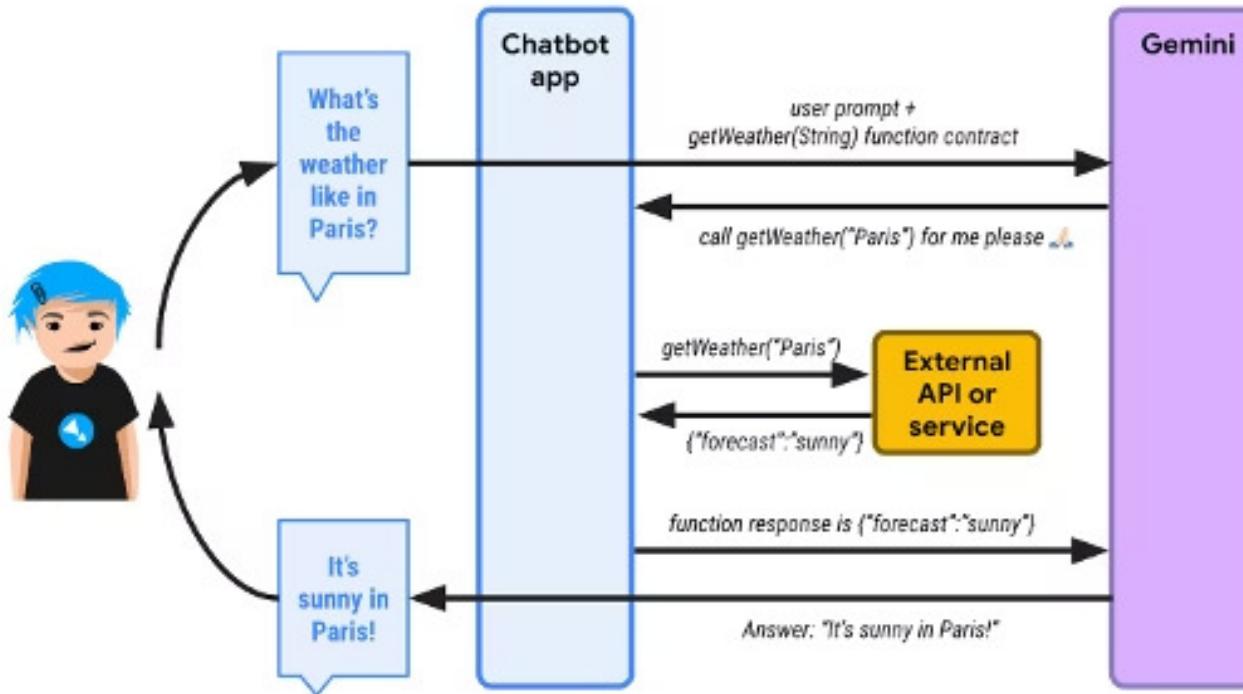
Large language models (LLMs) not only generate text, but can also execute actions, known as "plugins" or "tools".

However, not all large language models (LLMs) currently support this feature.

When invoking an LLM with these capabilities, the LLM does not directly call the tool, but instead indicates the intention to call it in its response, along with the necessary arguments.



Flow



Exposing Functions to the LLM

```
# This is the function that we want the model to be able to call
def get_delivery_date(order_id: str) -> datetime:
    # Connect to the database
    conn = sqlite3.connect('ecommerce.db')
    cursor = conn.cursor()
    # ...

# your function definition should be described using JSON Schema.
{
    "name": "get_delivery_date",
    "description": "Get the delivery date for a customer's order. Call this whenever you need to know the delivery date, for example when a customer asks 'Where is my package?'",
    "parameters": {
        "type": "object",
        "properties": {
            "order_id": {
                "type": "string",
                "description": "The customer's order ID."
            },
        },
        "required": ["order_id"],
        "additionalProperties": false,
    }
}
```

Berkeley Function Calling Leaderboard

This benchmark allows for an analysis to evaluate the performance of LLMs in terms of their ability to invoke external functions.

leaderboard consists of real-world data and will be updated periodically. For more information on the evaluation dataset and methodology, please refer to our [blog post](#) and [code release](#).

Expand/Collapse Table Last updated: 2024-06-22 [Change Log]

Rank	Overall Acc	Model	Cost (\$)	Average Latency (s)	ASR Summary	exec Summary	Relevance	Organization	License
1	90	Claude-3.5-Sonnet-20240620 (Prompt)	2.2	1.37	91.31	89.5	85.42	Anthropic	Proprietary
2	88	GPT-4-0125-Preview (Prompt)	5.25	1.97	91.22	88.1	70.42	OpenAI	Proprietary
3	87.65	Claude-3-Opus-20240229 (Prompt)	10.84	4.61	88.93	86.16	80.42	Anthropic	Proprietary
4	86.35	Gemini-1.5-Pro-Preview-0514 (FC)	0.86	1.94	87.92	83.32	89.58	Google	Proprietary
5	85.88	Gemini-1.5-Pro-Preview-0409 (FC)	0.86	1.95	87.62	82.88	88.75	Google	Proprietary
6	85.88	GPT-4-1106-Preview (FC)	5.07	5.97	88.62	81.73	80.42	OpenAI	Proprietary
7	85.59	GPT-4-turbo-2024-04-09 (Prompt)	5.25	2.57	90.01	86.04	62.5	OpenAI	Proprietary
8	84.71	Gorilla-OpenFunctions-v2 (FC)	0.31	0.05	89.38	81.55	61.25	Gorilla LLM	Apache 2.0
9	84.65	GPT-4-0125-Preview (FC)	4.83	4.72	87.17	84.76	82.92	OpenAI	Proprietary
									Meta Llama 3



gorilla.cs.berkeley.edu



Berkeley Function Calling Leaderboard V2 (aka Berkeley Tool Calling L...)

Explore The Berkeley Function Calling Leaderboard (also called The Berkeley Tool Calling Leaderboard) to see the LLM's ability to call functions (aka tools)...

Types of Actions



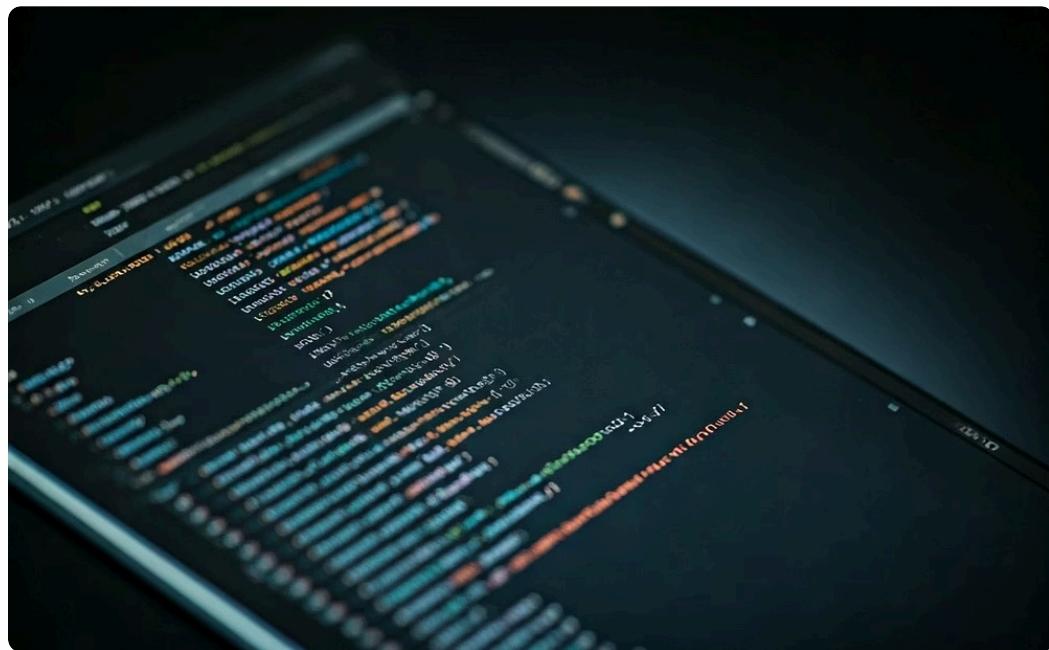
Invoke local functions

LLMs can interact with functions defined within the same environment.



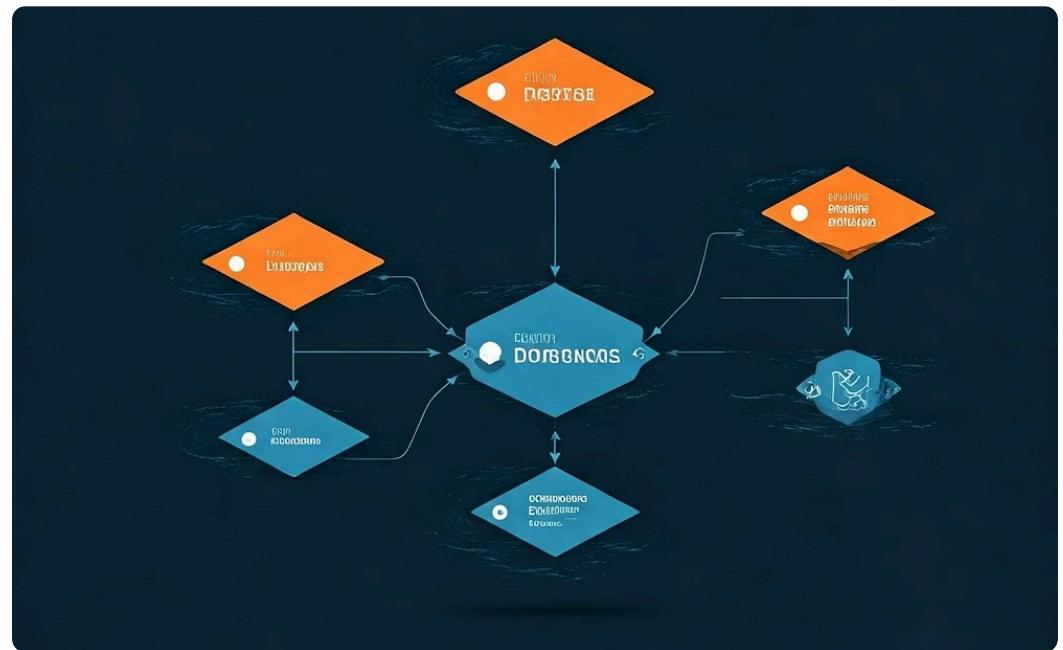
Invoke remote procedures

LLMs can execute tasks on external servers, accessing remote resources.



Generate and execute/interpret code

LLMs can generate code and execute/interpret programs in different programming languages.



Automate processes

LLMs can automate complex processes, combining different actions to complete specific tasks.

Example of a local procedure call

```
query = "What is 3 * 12? Also, what is 11 + 49?"  
  
llm_with_tools.invoke(query).tool_calls  
  
[{'name': 'multiply',  
 'args': {'a': 3, 'b': 12},  
 'id': 'call_1fyhJAbJHuKQe6n0PacubGsL',  
 'type': 'tool_call'},  
 {'name': 'add',  
 'args': {'a': 11, 'b': 49},  
 'id': 'call_fc2jVvkKzwuPWyU7kS9qn1hyG',  
 'type': 'tool_call'}]
```

```
# The function name, type hints, and docstring are all part of the tool  
# schema that's passed to the model. Defining good, descriptive schemas  
# is an extension of prompt engineering and is an important part of  
# getting models to perform well.  
def add(a: int, b: int) -> int:  
    """Add two integers.  
  
    Args:  
        a: First integer  
        b: Second integer  
    ....  
    return a + b  
  
def multiply(a: int, b: int) -> int:  
    """Multiply two integers.  
  
    Args:  
        a: First integer  
        b: Second integer  
    ....  
    return a * b
```

https://python.langchain.com/docs/how_to/tool_calling/

https://python.langchain.com/docs/how_to/custom_tools/

Invoking Remote Procedures

LLMs can access information and execute tasks in the real world through the invocation of remote procedures.

These invocations are carried out through APIs, such as the Moodle API, which allows obtaining information about virtual campus activities.

The LLM sends a request to the API, processes the received data and provides a response to the user.



Open API
Specification

Swagger

Invoking Remote Procedures

1 Evaluate the Query

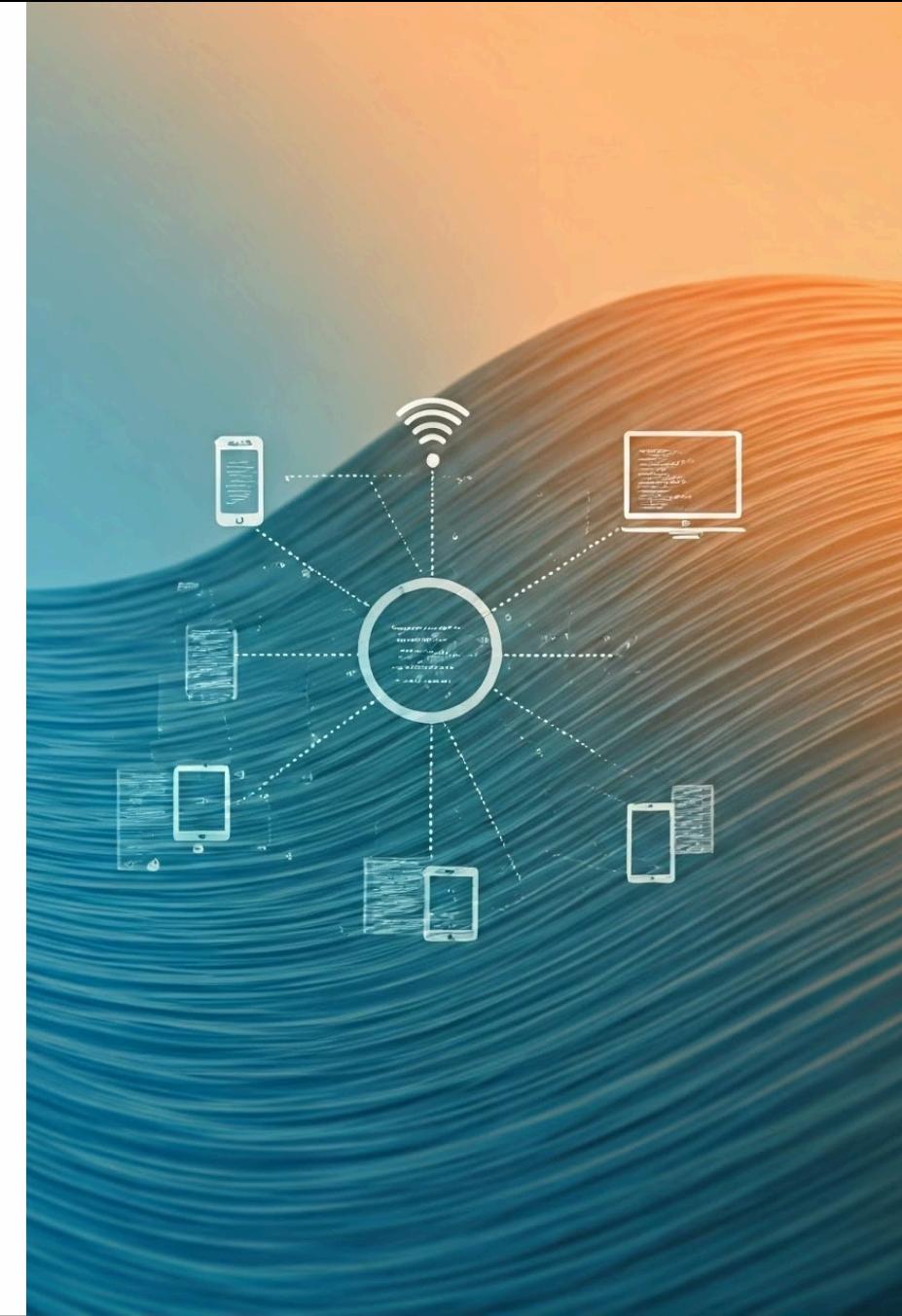
Analyse whether the user's query can be resolved using the documented API.

2 Generate an API Call Plan

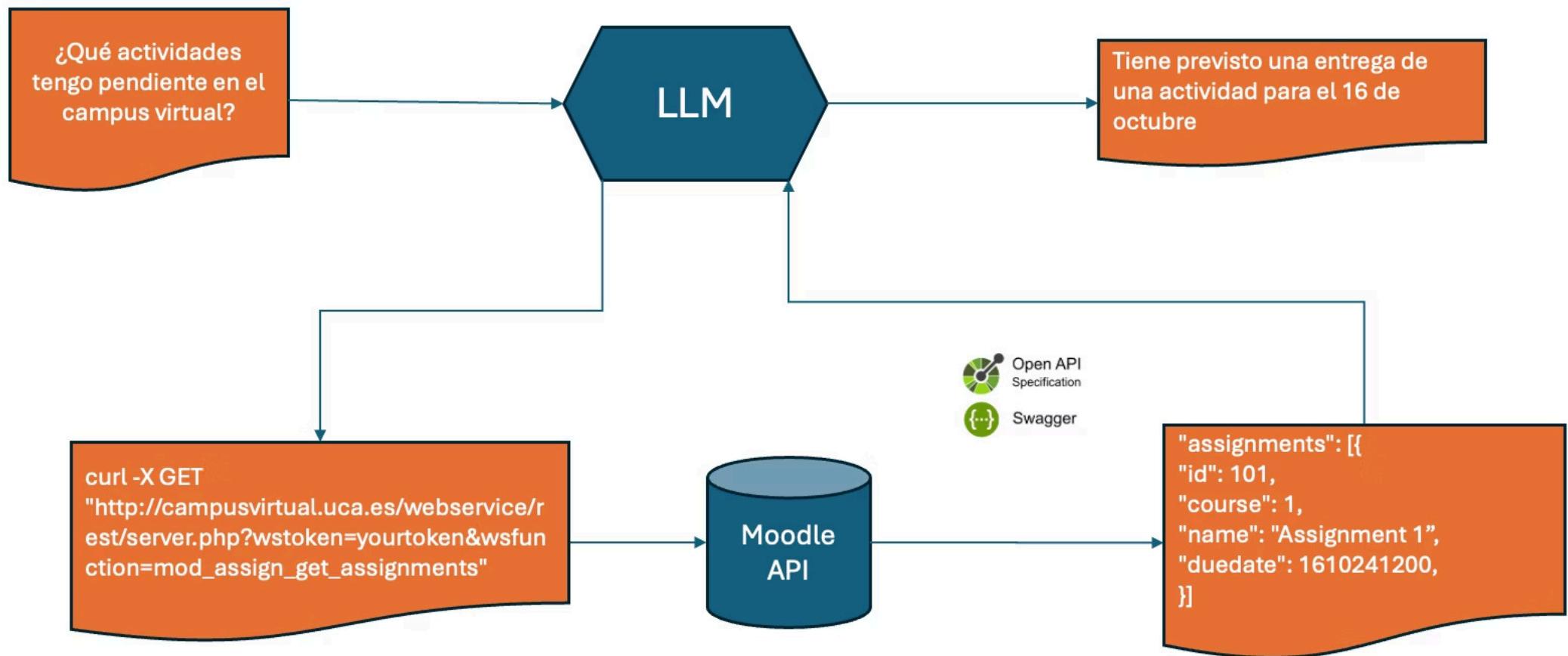
If the query can be resolved, create a plan that describes each API call and its purpose.

3 Manage Sensitive Actions

If the plan includes a DELETE call, first request user authorisation.



Remote Procedure Call Invocation Flow



OpenAPI tool system prompt

You should:

- 1) Evaluate whether the user query can be solved by the API documented below. If no, say why.
- 2) If yes, generate a plan of API calls and say what they are doing step by step.
- 3) If the plan includes a DELETE call, you should always return a request for authorisation from the User first unless the User has specifically asked to delete something.

You should only use API endpoints documented below ("Endpoints you can use:").

You can only use the DELETE tool if the User has specifically asked to delete something. Otherwise, you should return a request for authorisation from the User first.

Some user queries can be resolved in a single API call, but some will require several API calls.

The plan will be passed to an API controller that can format it into web requests and return the responses.

Here are some examples:

Fake endpoints for examples:

GET /user to get information about the current user

GET /products/search search across products

POST /users/{{id}}/cart to add products to a user's cart

PATCH /users/{{id}}/cart to update a user's cart

PUT /users/{{id}}/coupon to apply idempotent coupon to a user's cart

DELETE /users/{{id}}/cart to delete a user's cart

Example of remote procedure call

```
spotify_agent = planner.create_openapi_agent(  
    spotify_api_spec,  
    requests_wrapper,  
    llm,  
    allow_dangerous_requests=ALLOW_DANGEROUS_REQUEST,  
)  
user_query = (  
    "make me a playlist with the first song from kind of blue. call it machine  
blues."  
)  
spotify_agent.invoke(user_query)
```

<https://python.langchain.com/docs/integrations/tools/openapi/>

Generate and interpret code



Code execution

LLMs can generate code in various programming languages, such as Python, Java, and JavaScript.



Code interpretation

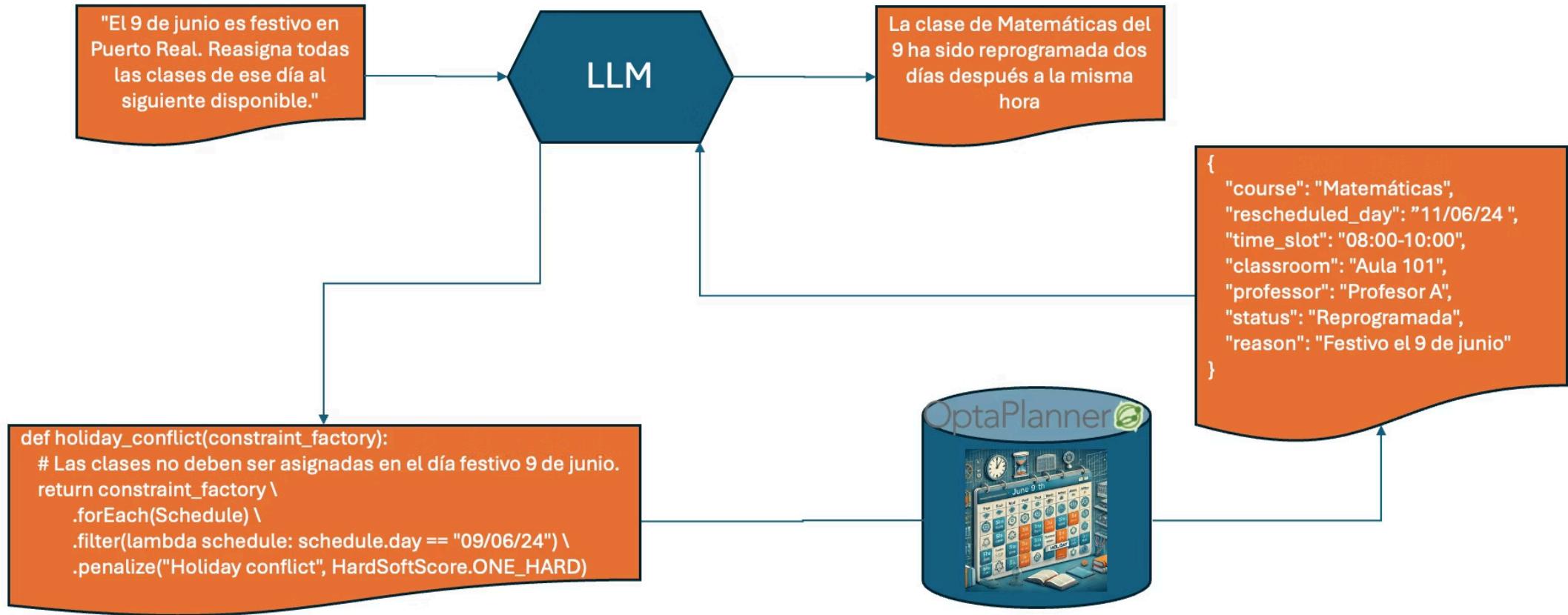
They can interpret and execute code in real-time in a protected environment, offering a more dynamic way of working with applications.



Interaction with business logic

They can interact with existing systems through code, automating tasks and simplifying complex processes.

Flow



Example of code generation and interpretation

```
python_repl = PythonREPL()
```

```
python_repl.run("print(1+1)")
```

Python REPL can execute arbitrary code. Use with caution.

```
'2\n'
```

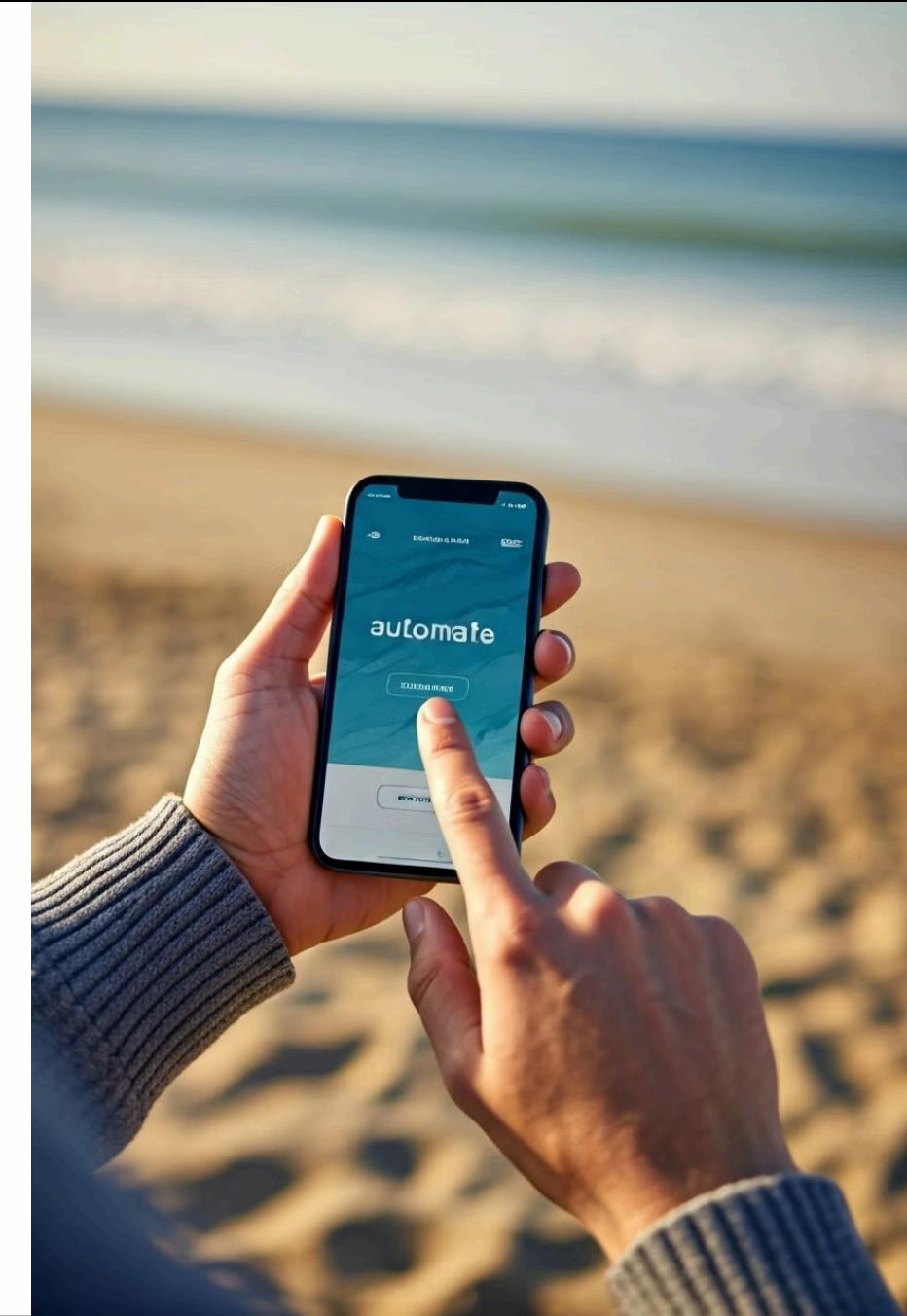
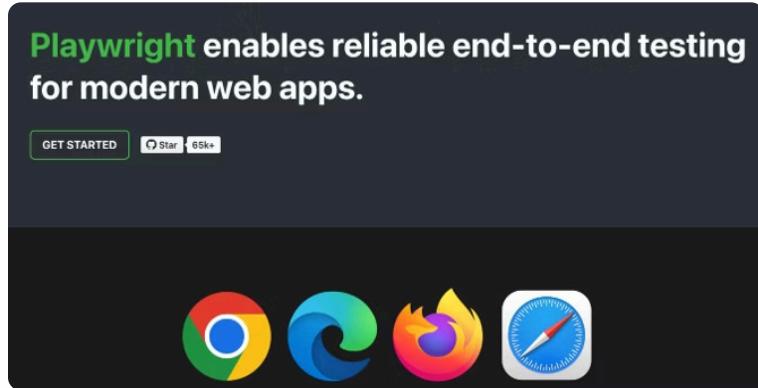
```
# You can create the tool to pass to an agent
repl_tool = Tool(
    name="python_repl",
    description="A Python shell. Use this to execute python commands. Input
should be a valid python command. If you want to see the output of a value, you
should print it out with 'print(...)'.",
    func=python_repl.run,
)
```



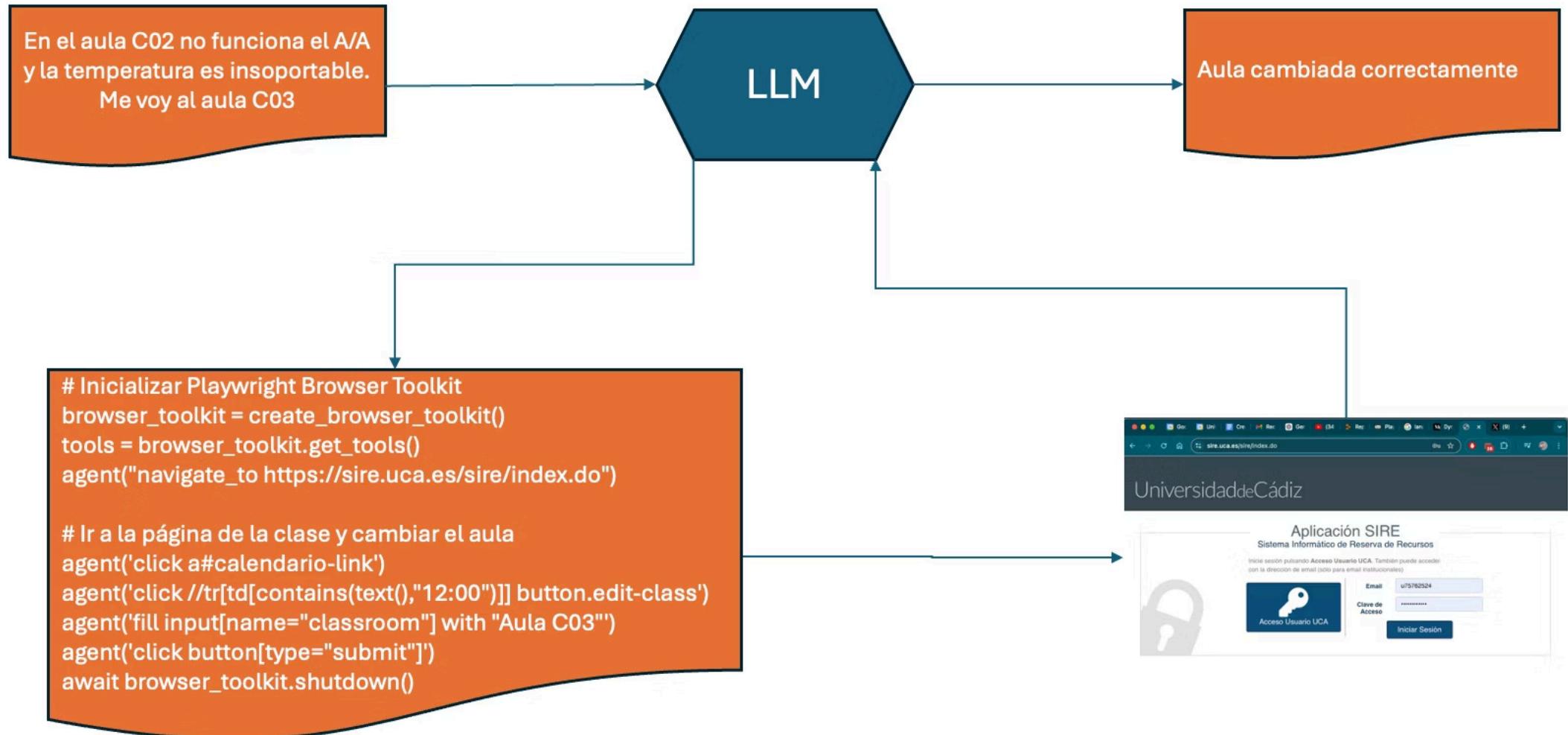
<https://python.langchain.com/docs/integrations/tools/#code-interpreter>

Automating processes

LLMs are capable of triggering the execution of task automation scripts. Libraries like Playwright enable interaction with a web browser for the retrieval and submission of data through its interface.



Flow



Example of automation in the browser

```
async_browser = create_async_playwright_browser()
toolkit = PlayWrightBrowserToolkit.from_browser(async_browser=async_browser)
tools = toolkit.get_tools()
tools
```

```
[ClickTool(async_browser=<Browser type=<BrowserType name=chromium executable_path=
  NavigateTool(async_browser=<Browser type=<BrowserType name=chromium executable_path=
  NavigateBackTool(async_browser=<Browser type=<BrowserType name=chromium executable_path=
  ExtractTextTool(async_browser=<Browser type=<BrowserType name=chromium executable_path=
  ExtractHyperlinksTool(async_browser=<Browser type=<BrowserType name=chromium executable_path=
  GetElementsTool(async_browser=<Browser type=<BrowserType name=chromium executable_path=
  CurrentWebPageTool(async_browser=<Browser type=<BrowserType name=chromium executable_path=
```

```
result = await agent_chain.arun("What are the headers on langchain.com?")
print(result)
```

<https://python.langchain.com/docs/integrations/tools/playwright/>

Ex 1. Creating a copilot





Collaboration Between Agents



Considerations When Working with Complex Agents

Multiple Scenarios

Attempting to cover all cases in a single prompt often generates suboptimal results. It is better to segment and specialise.

Additional Information

It is not always necessary to provide extra data to the LLM in each iteration. This can increase costs and reduce efficiency.

Access to Tools

Limiting the LLM's access to specific tools improves the security and efficiency of the system.

Balancing Creativity

The LLM's temperature should be adjusted according to the task: low for precision, high for creativity.

Divide and Conquer

Diverse Skills

Each agent specialises in a specific task, which optimises their performance through customised prompts.

Different LLMs

The most suitable language model is selected for each task, which improves the overall efficiency of the system.

Controlled Access

Tools and information retrievers are strategically assigned to each agent, which increases security and performance.



Chaining of Agents



Sequential Execution

Agents operate in series, passing results from one to another for complex tasks.



Conditional Execution

Decision logics are implemented to activate specific agents based on the circumstances.



Iterative Execution

Loops allow refining results or processing large data volumes efficiently.



Developer Control

The programmer maintains control of the flow, integrating LLMs as modular components.



Autonomous Agents

Full Autonomy

LLMs drive the application, making complex decisions without constant human intervention.

Hierarchy of Agents

Agents can act as tools for others, creating highly sophisticated systems.

Advanced Systems

Include loops, conditions and persistence for complex tasks and continuous learning.

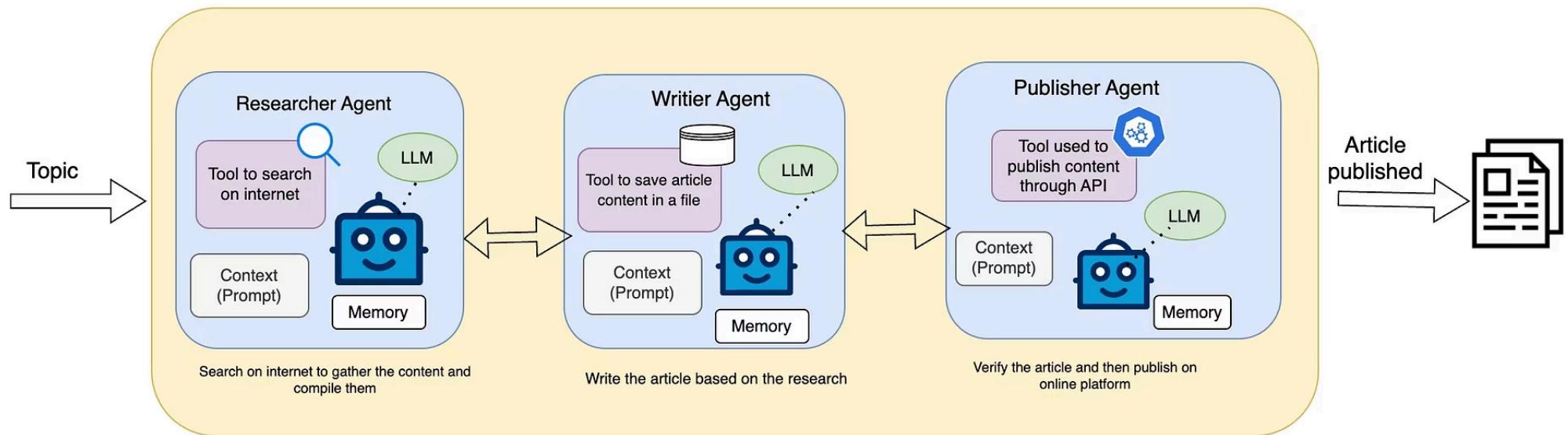
Human-in-the-loop

The option of human supervision to approve or cancel critical actions is maintained.

Workflows

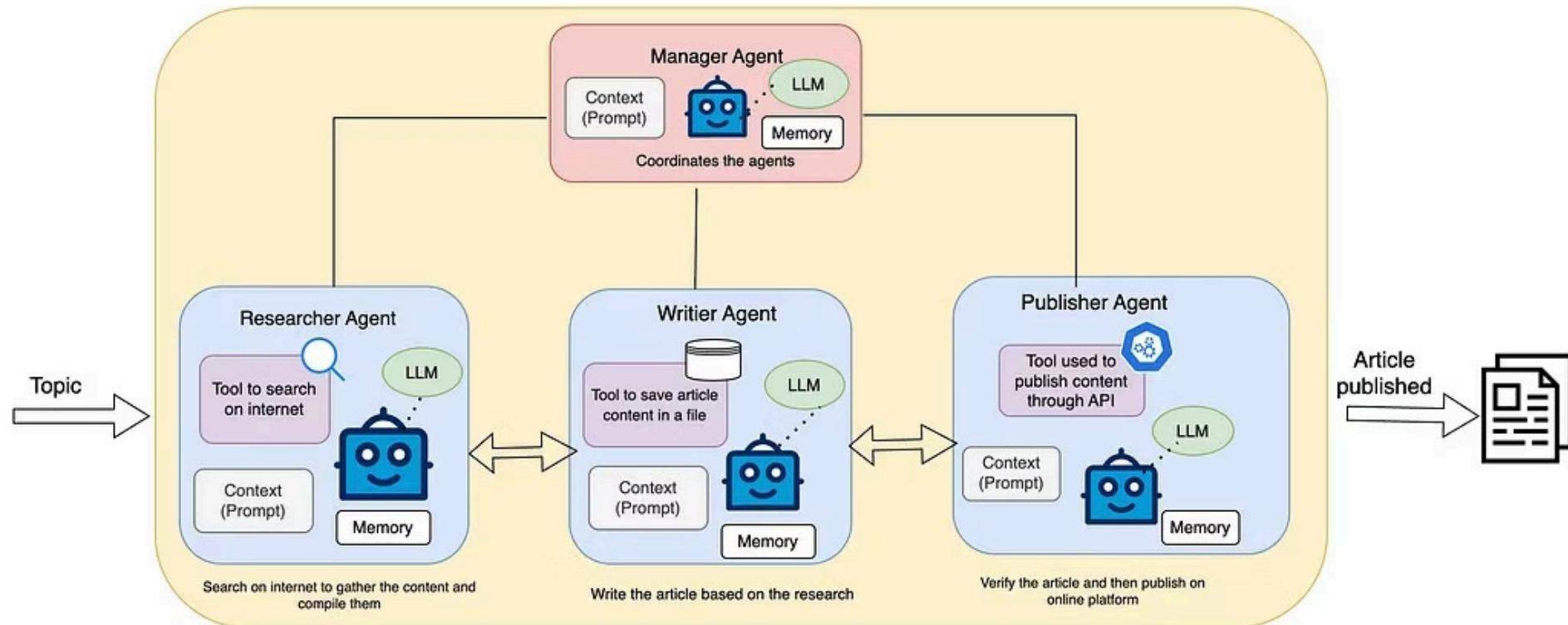
Tools and frameworks, such as LangGraph, are emerging that allow modelling agents through flow diagrams

Multiagent architecture. Sequential



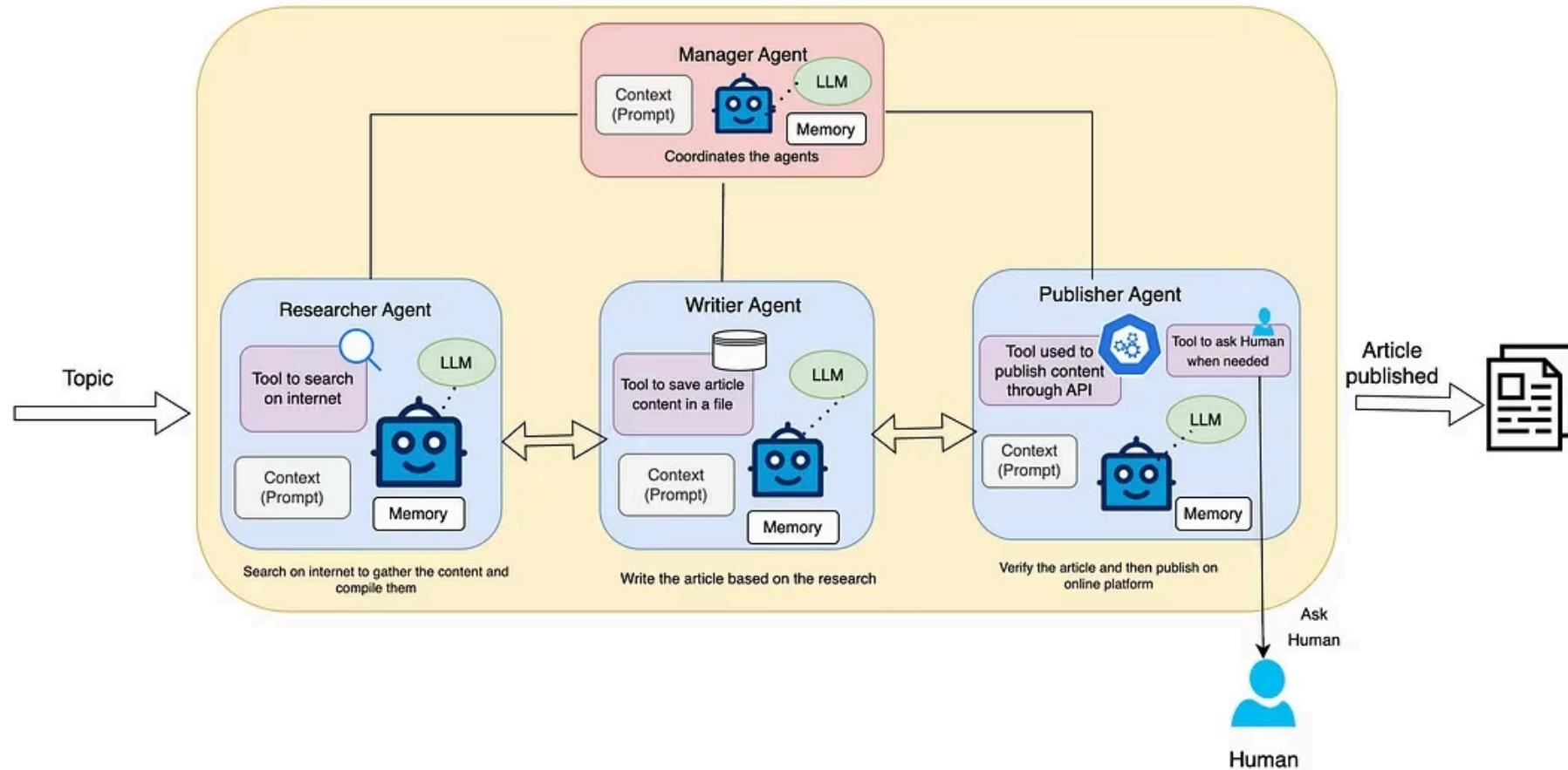
Source: <https://medium.com/@pallavisinha12/understanding-lm-based-agents-and-their-multi-agent-architecture-299cf54ebae4>

Multiagent architecture. Hierarchy



Source: <https://medium.com/@pallavisinha12/understanding-lm-based-agents-and-their-multi-agent-architecture-299cf54ebae4>

Multiagent architecture. Human-in-the-loop



Source: <https://medium.com/@pallavisinha12/understanding-lm-based-agents-and-their-multi-agent-architecture-299cf54ebae4>

IDEs for Agentic AI

Jules (Google)

Jules is an **asynchronous coding agent** powered by Gemini 2/2.5, designed to handle multi-step tasks in your codebase.

- **Context-aware:** clones your repo into a secure Google Cloud VM to fully understand project structure
- **Task automation:** writes tests, fixes bugs, bumps dependencies, builds features
- **Workflows:** accepts prompts or issue labels, providing summaries and pull request patches



```
/ Objective: Change UserOnboarding to use a state machine for steps
```

```
lass UserOnboarding {  
constructor(user) {  
this.user = user;  
  
this.steps = {  
welcome: { next: 'profile_setup' },  
profile_setup: { next: 'preferences' },  
preferences: { next: 'done' },  
done: { ne  
  
this.currentStep = 0;  
}  
  
async nextStep() {
```

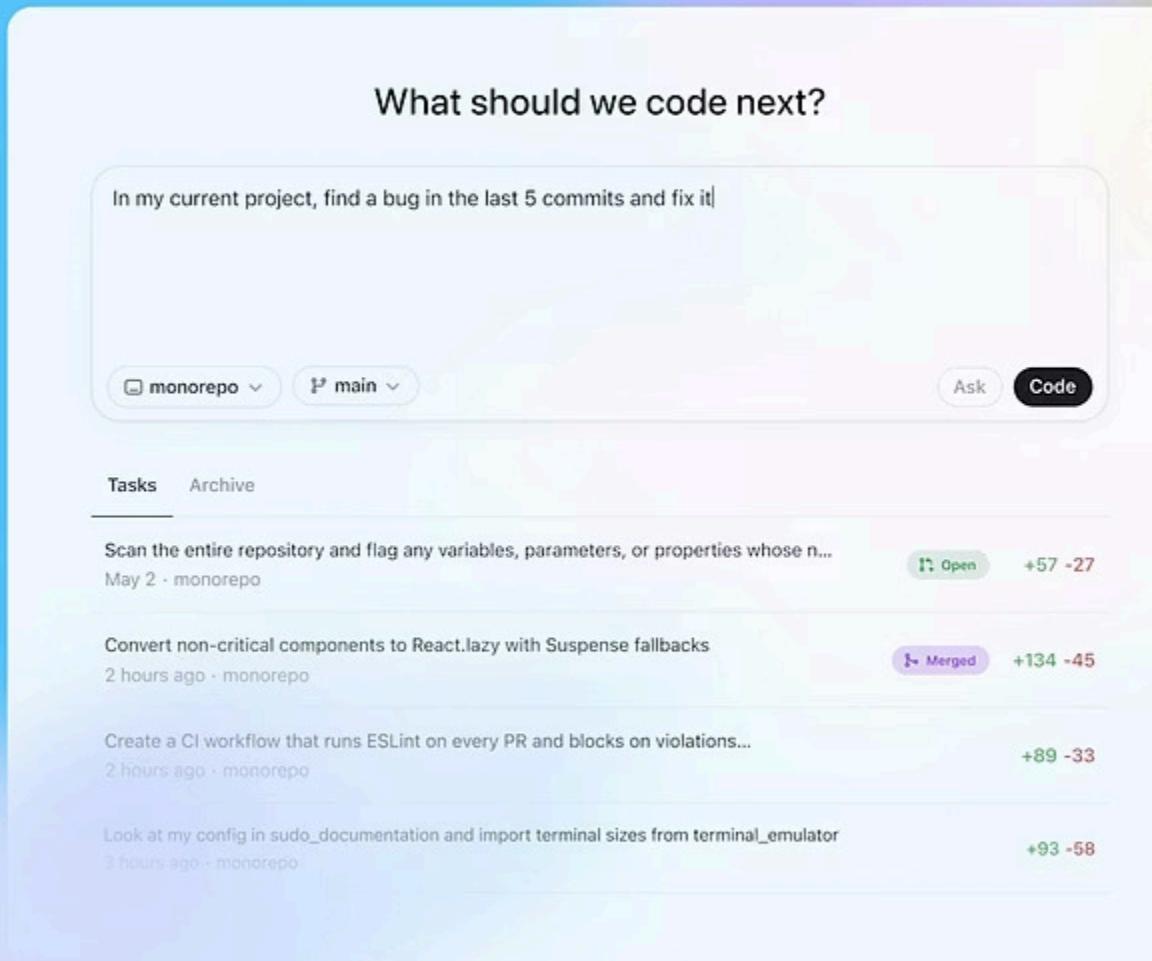
```
if (this.currentStep < this.steps.length -1) {
```

```
this.currentStep++;  
// TODO: Trigger analytics
```

Codex Agent (OpenAI)

Codex is a **cloud-based multi-task software engineering agent** available from within ChatGPT.

- Works across **multiple sandboxed tasks** simultaneously
- Can **write features, fix bugs, propose PRs, run tests iteratively**
- **Available** to ChatGPT Pro, Team, Enterprise now; soon Plus/Edu 
- Also offers an **open-source CLI agent** for local repo interaction



What should we code next?

In my current project, find a bug in the last 5 commits and fix it

monorepo main

Ask Code

Tasks Archive

Scan the entire repository and flag any variables, parameters, or properties whose n...
May 2 · monorepo Open +57 -27

Convert non-critical components to React.lazy with Suspense fallbacks
2 hours ago · monorepo Merged +134 -45

Create a CI workflow that runs ESLint on every PR and blocks on violations...
2 hours ago · monorepo In Progress +89 -33

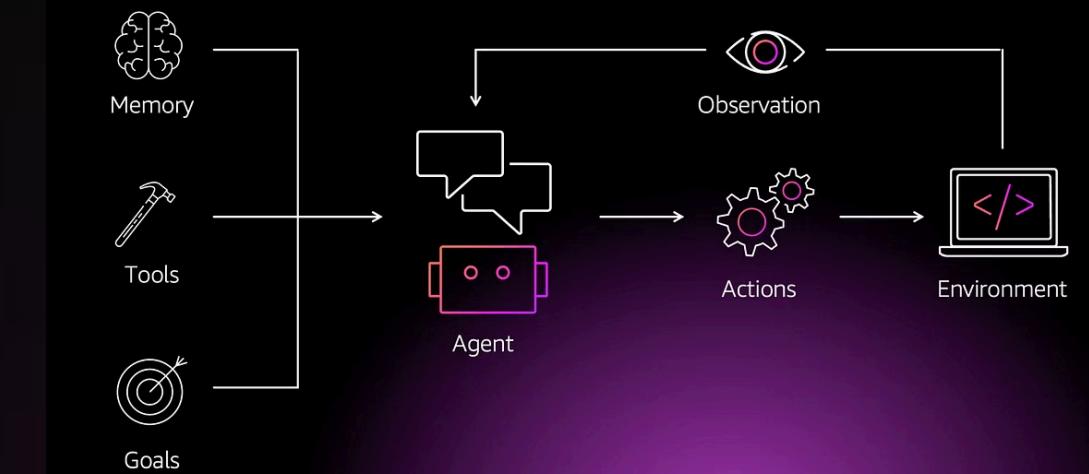
Look at my config in sudo_documentation and import terminal sizes from terminal_emulator
3 hours ago · monorepo In Progress +93 -58

AWS Agentic AI (Amazon)

Amazon Q Developer brings **agentic coding** directly into IDEs like VS Code, JetBrains, etc.

- Integrates in-IDE **agent chat + local file execution + AWS CLI access**
- Supports **switching between planning chat and build-time actions**
- Can read/write files, run shell commands, query AWS resources (e.g., list S3 buckets)

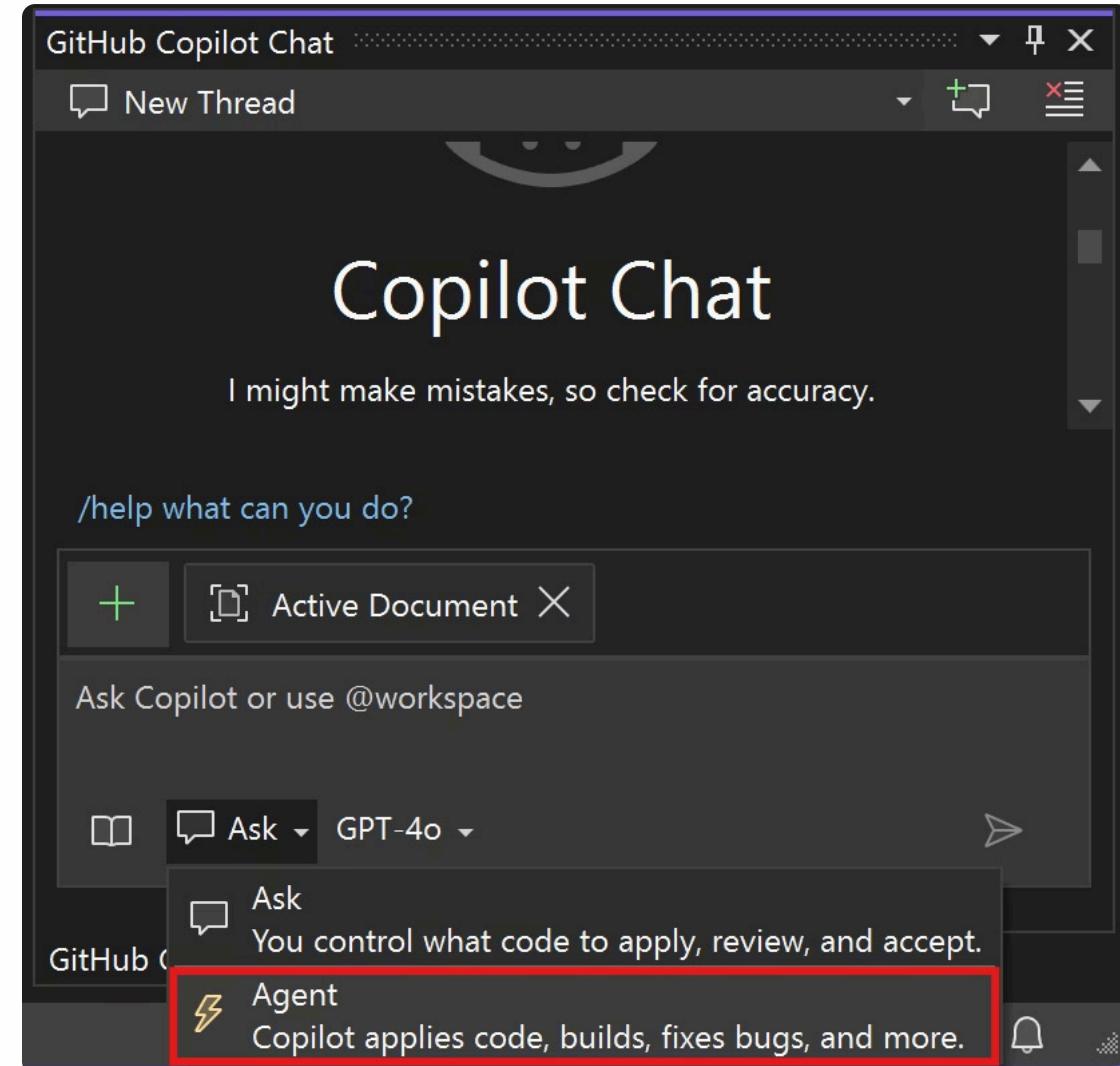
What is agentic AI?



GitHub Copilot Agent Mode (Microsoft)

Copilot Agent Mode elevates Copilot to a **fully autonomous coding agent** integrated with GitHub Actions.

- Triggered by assigning issues or toggling Agent Mode in IDE
- Clones repo in GitHub Actions VM, analyzes code, implements changes, iterates until completion
- Generates draft PR with commit history, logs its reasoning, reacts to repo activity

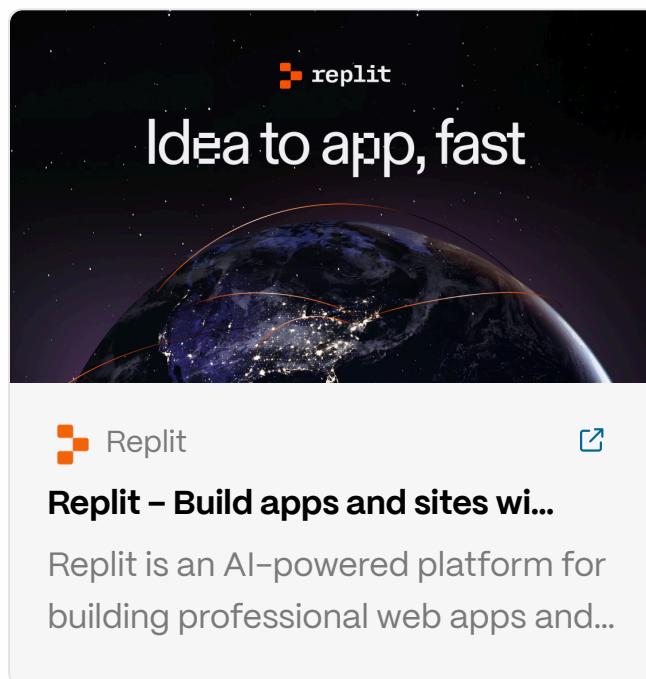


OK, but are there any free IDEs to test?

"Free" IDEs

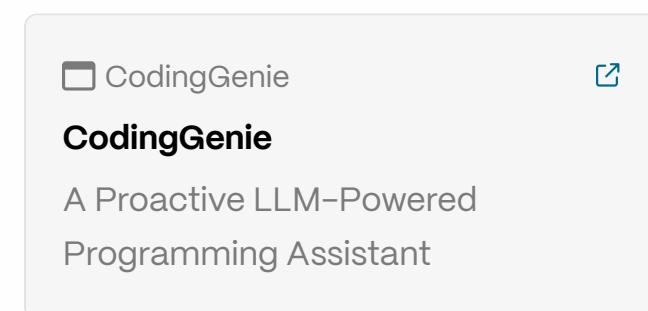
Replit Agent – Browser-based AI agent platform

- Web IDE + **natural-language agent** that builds apps or scripts
- Free tier supports basic functionality; paid plans unlock advanced features
- Enables describing app in natural language → generates working code
- Ideal for rapid prototyping, especially in educational settings



CodingGenie (Open-Source)

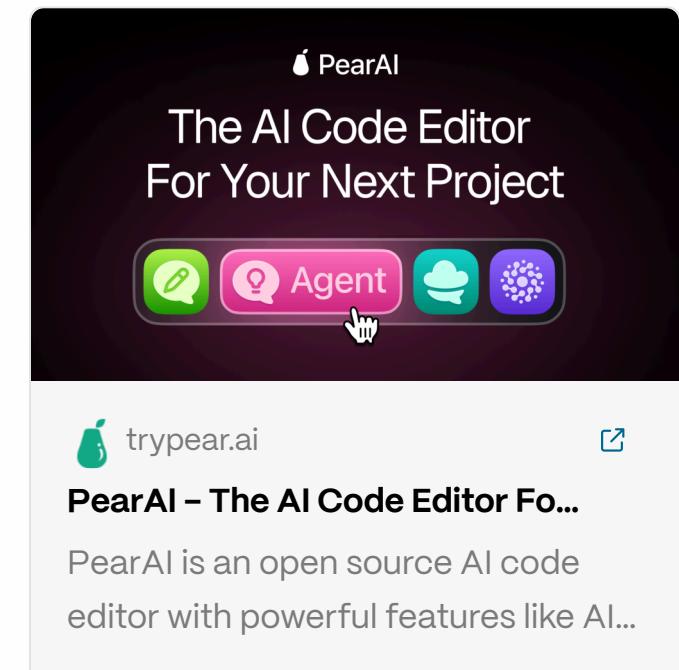
- A proactive LLM-powered programming assistant
- 🚀 **Open-source** and free: GitHub repo available (Apache-2.0)
- Context-aware suggestions (bugs, tests, features) **proactively offered**
- Demo videos + docs included; designed to integrate into local IDEs



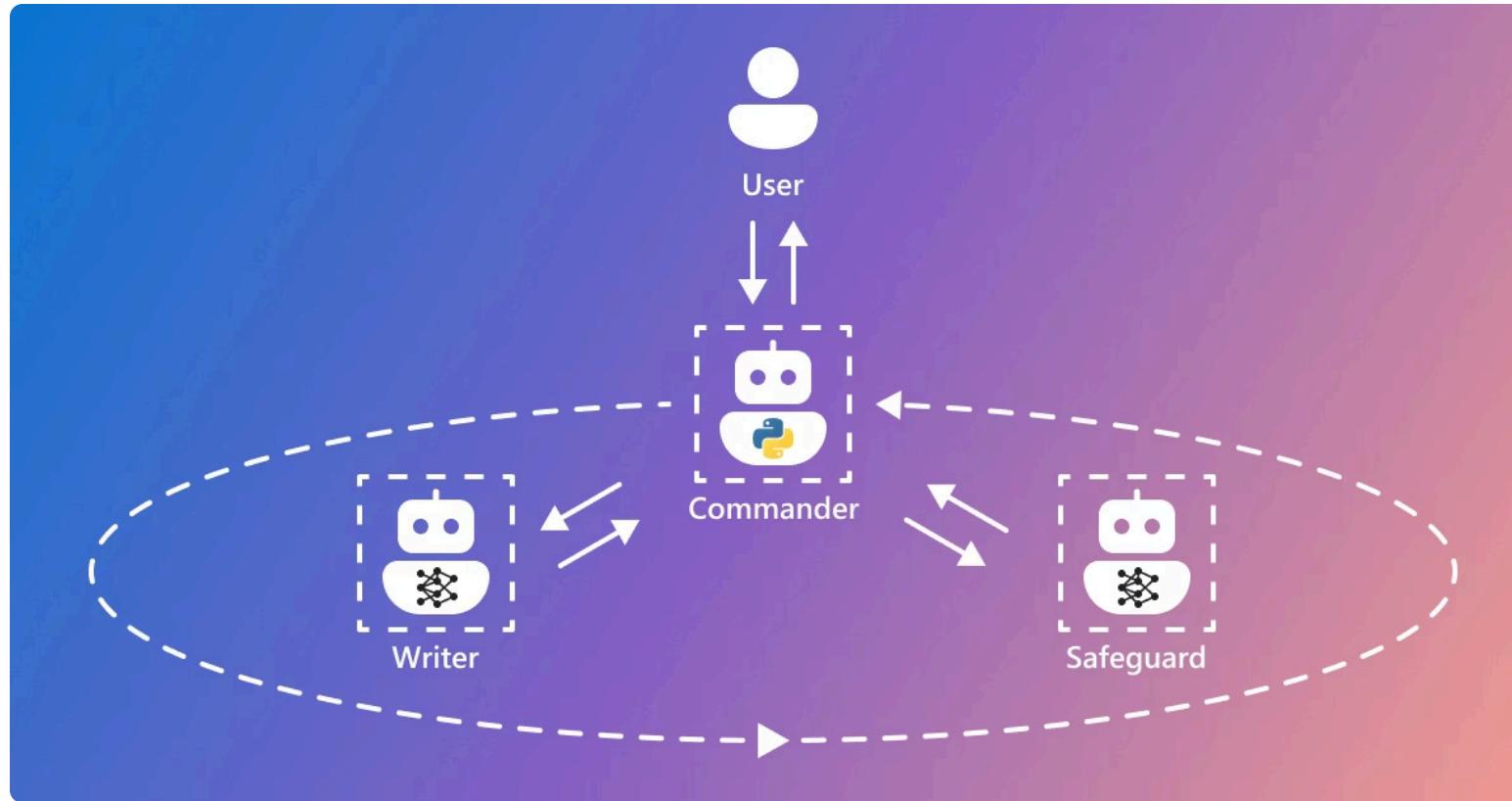
PearAI

Open-source AI code editor

- Features include **AI chat**, debugging, and context-aware coding
- Router selects best-performing coding models automatically
- Downloadable with free features; may offer premium enhancements later



Bonus resource!



**MICROSOFT
AUTOPEN**

<https://microsoft.github.io/autogen/stable//index.html>

Summary

This presentation has explored the landscape of LLM agents, from their origins to the present day.

We have delved into the growing autonomy of agents, examining their architectures .

The tools and platforms that enable the development and application of LLM agents have been highlighted.