

Project #1

Agentic antics

Deadline: 16.11.2026, 12:00

Modus operandi This is the first out of four semester projects. For each project you will typically have at least 3 weeks. A course requirement is to pass at least 3 projects (the fourth can be replaced by the quiz in lecture, details on the lecture website). To pass a project, you have to submit it by the deadline via email to

nevermann@itp.uni-frankfurt.de and
LozanoGomez@physik.uni-frankfurt.de

Furthermore, you have to present the project in a short (~ 10-15 min.) presentation in the tutorial session after the deadline and be able to answer questions. The projects are different from a typical exercise sheet: they are deliberately open, allowing you to get creative. For the projects, you may use any tools of your choice. But be mindful, you must be able to explain both the functionality and the code of your project.

Introduction

Think of some task of yours that is repetitive and really annoying to do. Could an AI agent improve your life? In this project, you will apply the Python skills you acquired in lecture to implement your own AI agent that will hopefully make your life just a little better.

A large language model (LLM) on its own is a pure text machine: text goes in, text comes out. It cannot read a file, query a database, or check tomorrow's weather. An AI agent adds a second component, the harness, which is simply a computer program that sits between the user, the LLM and external tools. The harness lets the LLM use tools repeatedly (in a tool loop) until the LLM comes up with a solution to the given task.

Agent code: Understanding the tool loop

In lecture, you saw a small example for an AI agent with tools allowing the agent to manage files [L1]. Before working on your own agent, you should improve your understanding on the basic agent from the lecture. In Alg. 1, the code from the lecture is written in pseudocode: instead of a programming language, the core logic of the program is written in plain English with a high level of simplifications. Read through the pseudocode and compare with the implementation from the lecture.

Algorithm 1 Pseudocode of the agent from the lecture

Require: system prompt S , tool descriptions D , tool implementations

```
1: procedure MAIN
2:    $messages \leftarrow [(\text{system}, S)]$ 
3:   loop ▷ one iteration per user message
4:      $u \leftarrow \text{READUSERINPUT}$ 
5:     if  $u = \text{quit}$  then
6:       break
7:     end if
8:     append  $(\text{user}, u)$  to  $messages$ 
9:      $\text{RUNAGENT}(messages)$ 
10:  end loop
11: end procedure

12: procedure  $\text{RUNAGENT}(messages)$ 
13:  loop ▷ the tool loop: one iteration per LLM call
14:     $r \leftarrow \text{LLM}(messages, D)$ 
15:    append  $(\text{assistant}, r)$  to  $messages$ 
16:    if  $r$  contains no tool calls then
17:      print  $r.\text{text}$ 
18:      return ▷ final answer
19:    end if
20:    for all tool calls  $(name, args)$  in  $r$  do
21:       $result \leftarrow \text{EXECUTE}(name, args)$  ▷ done by the harness
22:      append  $(\text{tool}, result)$  to  $messages$ 
23:    end for
24:  end loop
25: end procedure
```

Getting your LLM-API ready

You are probably used to interacting with LLMs through a chatbot website. For implementing agents, however, your harness code must interact with the LLM through an API [L2]. Recall Sheet #1, where you worked on getting API access to an LLM. For the implementation of your agent, you will need that access. If you have not yet got the API access working, do that before start working on the agent.

Building the agent

Before you start building an agent, you should think about what the agent should be able to help you with. Anything is possible here, get creative. Depending on what you

choose, you need to define a different set of tools. You might use external APIs¹, such as arXiv, Wikipedia, Deutsche Bahn or OpenStreetMap, or define tools that interact with your computer, such as browsing files [L1]. You could also write tools that allow the LLM to access certain programs on your computer: maybe a tool that gives the LLM access to plotting in Matplotlib. If you need further inspiration, discuss the matter with a chatbot of your choice.

Task 1 – Implementing the agent

Implement an agent in Python. The agent should contain a minimum of 4 non-trivial tools. As a starting point, you can use the agent implementation from the lecture [L1]. Test your agent with a few demo examples. What improvements can you make to the agent from the lecture?

Remarks and presentation

A few practical remarks before you start:

- **Keep your API keys secret.** Never write them directly into your code, upload them to GitHub, or show them in your presentation. Store them in an environment variable or in a separate file that is excluded from version control. Also don't spam external APIs and adhere to the usage policy.
- **Limit the loop.** The tool loop from the lecture runs until the LLM decides to stop. A confused model could loop forever and exceed your LLM-API limits, so set a maximum number of iterations.
- **LLM code execution** Depending on the agent you choose to build, you give an LLM access to run arbitrary code on your machine. Be aware of that.

Task 2 – Presentation

To pass the project, you have to present your solution with a few demo examples in the class after the submission deadline and be able to explain the logic and the code. The format of the presentation is up to you. You can make slides, create a Jupyter Notebook, simply show interesting parts of your code or something completely different. You could also let a chatbot generate pseudocode, similar to Alg. 1, from your agent and use that in your presentation to explain and improve your own understanding of the most important concepts.

¹To avoid confusion: API is a generic term for a structured interaction between an application and a computer program. In Task 2 you learned how to use an LLM API. In this case the application is the LLM, but e.g. weather services also provide APIs. Here, the application is not an LLM but the weather service.