Bachelor Thesis

Enhancing Data Workflows and Reproducibility with LLM Agents

How effective are LLMs with RAG and Agents in improving data analysis pipelines in terms of effectiveness and accuracy in astrophysics?

Submission date:	February 3, 2025
Author:	Enrico Tom Leonhard Strauß
Cadastral number:	19103
First Reviewer:	Prof. Dr. rer. nat. André Grüning
Second Reviewer:	Dr. rer. nat. Arman Khalatyan

HOST Hochschule Stralsund
University of Applied Sciences

AIP

Zusammenfassung / Abstract

Diese Arbeit stellt einen Machbarkeitsnachweis zur Nutzung von LLMs unter Verwendung von modernen Technologien dar, um die automatische Generierung von Workflows für die Plattform für Datenanalyse REANA zu realisieren. Das Thema der Arbeit lautet „Enhancing Data Workflows and Reproducibility with LLM Agents“, wobei die Frage „How effective are LLMs with RAG and Agents in improving data analysis pipelines in terms of effectiveness and accuracy in astrophysics?“ beantwortet werden soll. Die Machbarkeit eines solchen Systems anhand einer Prototypimplementierung soll überprüft werden. Diese Arbeit stellt einen solchen Prototypen vor und zeigt die Probleme auf, welche noch gelöst werden müssen, um ein produktives System aufzubauen. Die Arbeit enthält einen Einblick sowohl in die theoretischen Grundlagen, auf welchen die Implementierung beruht, als auch in die praktische Nutzung des Prototypens.

Bitte beachten Sie, dass diese Arbeit in englischer Sprache verfasst ist. Lediglich diese kurze Zusammenfassung wurde in deutscher Sprache verfasst, da dies von der Hochschule Stralsund vorausgesetzt wurde.

This thesis presents a proof of concept for using LLMs with modern technologies to realise the automatic generation of workflows for the data analysis platform REANA. The topic of the thesis is "Enhancing Data Workflows and Reproducibility with LLM Agents", where the question "How effective are LLMs with RAG and Agents in improving data analysis pipelines in terms of effectiveness and accuracy in astrophysics?" should be answered. The feasibility of such a system is to be tested by means of a prototype implementation. This thesis presents such a prototype and shows the problems that still need to be solved to build a productive system. The work contains an insight into both the theoretical principles on which the implementation is based and the practical use of the prototype.

This thesis will be written in English. Only the first summary has been written in German since the University of Applied Sciences "Hochschule Stralsund" presupposes this.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In modern astrophysics, petabytes of data must be processed and analyzed to create figures and publish articles and press releases. The Astrophysical Institute Potsdam (AIP) uses REANA (definition in Section 2.2) to perform large calculations and generate illustrations from vast amounts of observational or simulated data, utilizing Kubernetes clusters. Users can create, manage, and execute workflows through a command-line client. However, the platform's users have diverse IT backgrounds. While many scientists possess IT-related knowledge, they may not be familiar with REANA, and some users have limited IT experience.

The ability of Large Language Models (definition in Section 2.1.1) to process, structure, and analyze large data sets can significantly enhance research efficiency and reproducibility. This thesis explores how Retrieval-Augmented Generation (definition in Section 2.1.2) and autonomous agent-based approaches (definition in Section 2.1.3) can improve data workflows, focusing on REANA. REANA is a reproducible analysis platform that enables scientists to run containerized data analysis pipelines on remote compute clouds.

A typical scenario is shown in Figure 1.0.1. A scientist or public relations person needs to generate an illustration for a press release. They use various data sources and run code on the data. Often, the person preparing the press release has not written the scientific article, and the code used for data analysis and figure production is outdated. Although REANA could support this task, it presents a steep learning curve for first-time users.

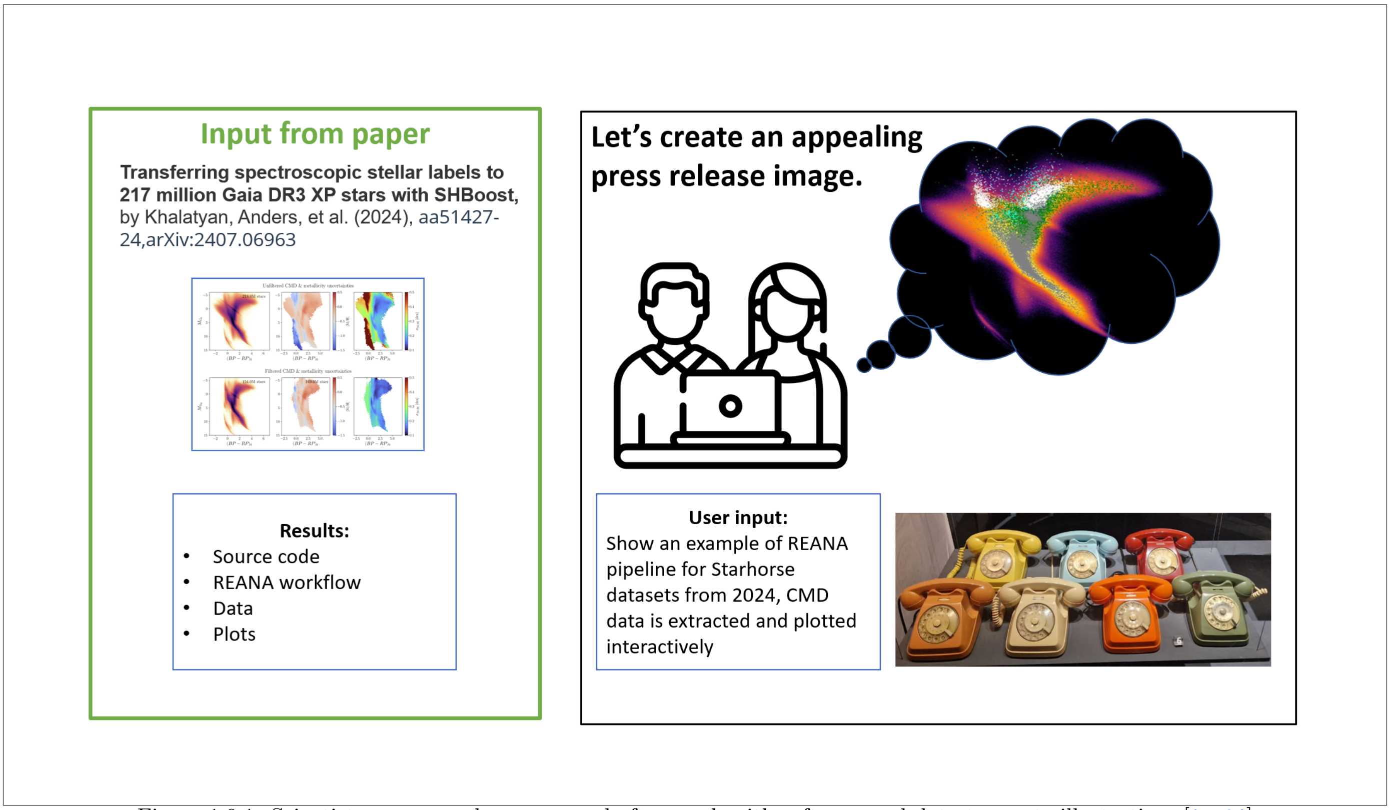

Figure 1.0.1: Scientists or press release personal often work with software and data to create illustrations.[AKP24]

In this bachelor project, the new developments in the field of LLMs are used, as well as RAG and Agents, to provide an easier handling of REANA workflows. The study aims to assess the effectiveness and accuracy of LLM-driven workflow automation within astrophysical data analysis. By implementing a prototype system that integrates modern machine learning frameworks, such as FlowiseAI, LangChain, and Qdrant, the thesis investigates the potential of these technologies in automating the generation and validation of REANA workflows. Furthermore, challenges such as infrastructure setup, model evaluation, and workflow validation are examined to determine the feasibility of a fully automated system. Based on that the question "How effective are LLMs with RAG and Agents in improving data analysis pipelines in terms of effectiveness and accuracy in astrophysics?" should be answered. Overall, the thesis should serve one overarching theme "Enhancing Data Workflows and Reproducibility with LLM Agents".

To solve the thesis question, the developed prototype should be able to automate the generation of REANA workflows, which are defined by yaml files, based on a user prompt. Figure 1.0.2 shows a possible solution for that is an agent system, that produces the results with help of a supervisor or planner and multiple workers.

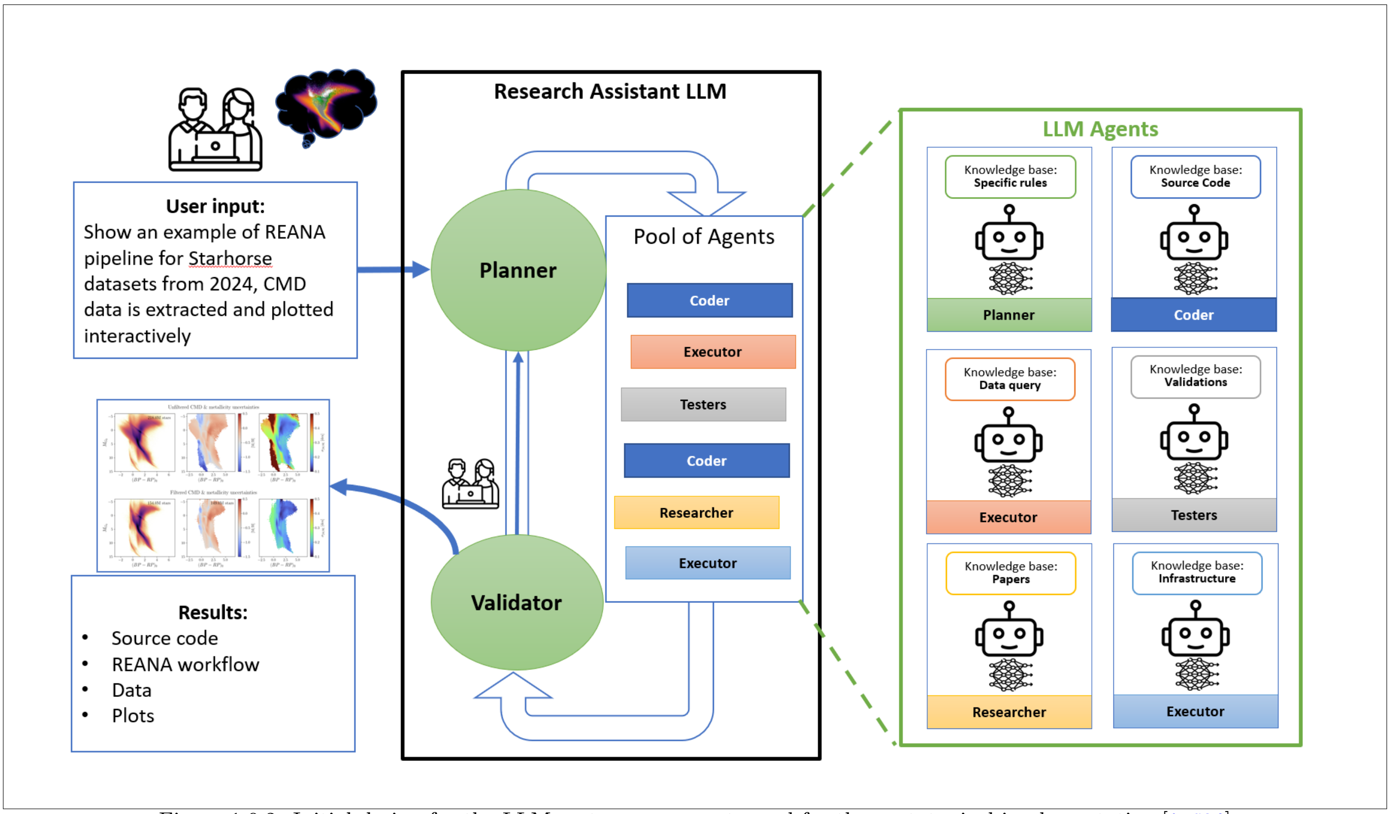

Figure 1.0.2: Initial design for the LLM system uses agents pool for the prototypical implementation.[AKG24]

The main objective of this prototype is to demonstrate through the unique integration of RAG and Agents in connection with REANA how large the impact of this new technologies is related to the scientific field of astrophysics. To evaluate this different tests will be performed. These tests include an evaluation, which is based on precise specifications of responses of several implementations related to a predefined question catalogue, which includes the generation of whole REANA workflows, but also contains theoretical questions. Through a combination of theoretical exploration and practical implementation, this work contributes to the ongoing discourse on AI-assisted scientific computing. It not only highlights the strengths of current LLM-driven automation techniques but also identifies key areas that require further development to achieve a robust and scalable solution.

The appendix highlights selected features of the used code base and is referenced in the thesis. The whole code base is available at a public GitHub repository[1]. Notice that this repository has no commit history, but the examiners of this thesis are able to access the private repository with the full commit history.

[1] https://github.com/etlstrauss/bachelor-thesis-public

Chapter 2

Theory

This chapter provides the explanation of natural languages processing basics and a description of the REANA data analysis platform. Furthermore, it contains references to libraries and frameworks used for the implementation of the prototype.

2.1 Technical fundamentals

To enable the reader to follow the developments in this thesis this section provides some information about general technical fundamentals.

2.1.1 Large Language Models (LLMs)

Large Language Models (LLMs) are systems, which are able to understand and generate a natural language. The basis for LLMs are machine learning techniques, which are used to train these models with a massive amount of data.[WALMcs24]
The development of LLMs began with the understanding of NLP (natural language processing) in the 1950s. Till the end of the last century, NLP moved from a clear logical approach for analysis of the already existing linguistic data, to the development of first systems which are able to predict words on the basis of similar data. The final breakthrough to the present state-of-the-art technology was the introduction of Neural Networks (NN) and Transformer Architecture, which are the basis of today's Generative Pre-Trained Transformers (GPTs).[WALMcs24] In the curse of this thesis LLMs will be also referred as models.

Quantization

Based on the hardware limitations described in Section 3.1.1 it was only possible to use small models locally. Another way is the quantization of LLMs, because a quantized LLM had a reduced resource consumption compared to a model that is not quantized. Quantization is the process of transforming the weights of an LLM. The weights of an LLM are values, which are describing the connection within the Neural Network of

it. Normally, LLMs will be developed with fp16 or fp32 weight sizes. To reduce the necessary resources to run a model, the weights will be transformed to a smaller bit size. This will cause precision loss.[EgaAtEl24, LinAtEl24]
To show the impact of quantization of a model in terms of its size, let's look onto the "qwen-coder2.5:32b-instruct", open weight model provided by Alibaba Cloud[HQw25]. The size of this model in the fp16 format is 66 GB, according to the Ollama (definition in Section 2.3.1) model registry. The Ollama model registry is a platform where the files for the LLMs, which are runnable with Ollama, are accessible. The same model with an quantization of q8 (or int8) is 35 GB.[OQC24] The precisions loss, that comes with this, differs with regard to the field of application.

GGML and GGUF Formats

The main problem in the beginning of LLMs development was that it was complicated to share models and make them runnable on generic hardware. One example for that is a gaming PC with a build in Nvidia graphic card. Therefore, Georgi Gerganov invented the GGML file format. GGML is a tensor library which guarantees high performance, as the tensors will be stored in a specific way, but the configuration was very complex and had to be done "manually", which took a lot of time.[IGGUF24, HFGG24]
The follow-up GGUF format fixes some of GGMLs limitations, for example compatibility issues, and makes it possible to add new features to an existing model and to use older models.[IGGUF24]

Most models in the Ollama registry, which will be used in this thesis, are quantized based on GGML. Additionally, models from Hugging Face[HF25] (a platform that provides numerous easily accessible LLMs for local deployment) that are compatible with Ollama are quantized using GGUF. Therefore, we will focus only on these methods.

2.1.2 Retrieval-Augmented Generation (RAG)

The Retrieval-Augmented Generation (RAG) paradigm can be used to reduce "hallucinations"[1] of LLMs, which can cause misleading results, and to "extend" the (mostly not changeable) knowledge base of pre-trained LLMs. Overall, the idea is to get more information of a specific topic and give this as context to an LLM. There are different "stages" of RAG, but to explain the concept of RAG it is enough to take a look at a "naive RAG".[GaoEtAl24]
The process of this particular RAG implementation is very simple. A set of documents about a specific topic will be vectorized with a special model (at this point embedding models will be used. They are very small but especially designed to vectorize data), the vectors of this content and the content itself will be loaded into a vector store. This part of the process is named "indexing". To use these information, a user query is be vectorized, too. Now, with both sides (vectors of the query and vectors in the store) a similarity search can be made. After the search in the vector store, the best matchings, based on the search algorithm, will be routed to

[1] Hint from the author: The wording "hallucinations" is probably misleading, because LLMs are not creative or on an direct way "intelligent". That means that "hallucinations" are more like a misinterpretation by the models, for example based on poor training data quality.[IHal25]

the LLM as context for the user prompt. After this "retrieval" the LLM generates output based on the context and the user query.[GaoEtAl24]
This process is important, because the context the LLM can use is probably very limited by hardware resources. That is why only the top "k" entries of the vector store should be used as context for the user query. Top "k" is a specific term of this scientific field. It describes the first "k" entries which are found within a vector store. Figure 2.1.1 below illustrates this process.

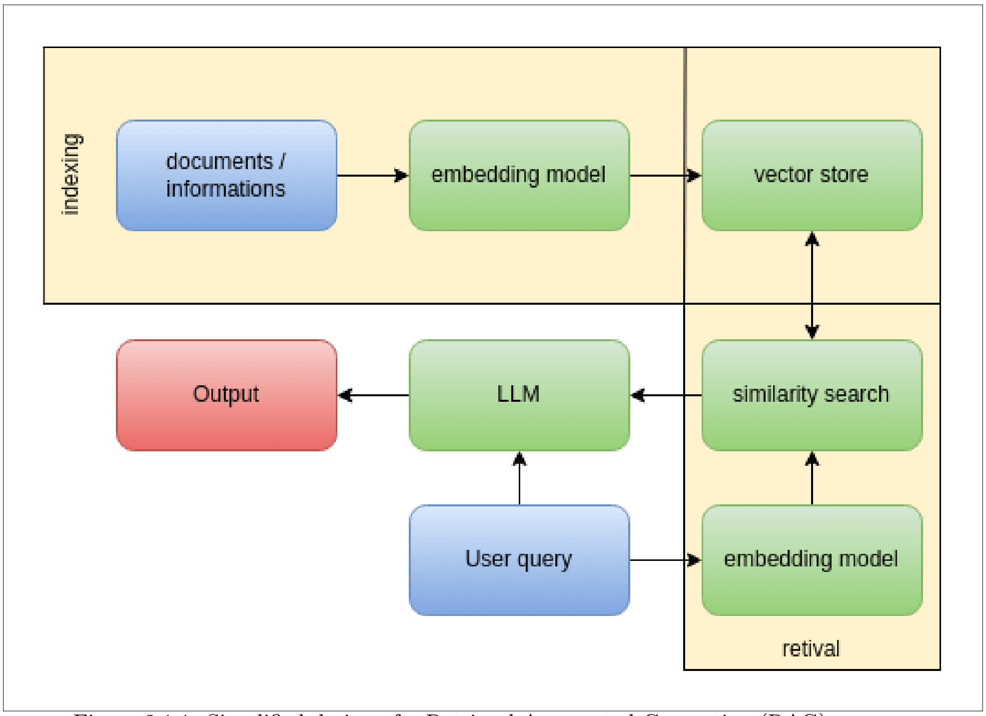

Figure 2.1.1: Simplified design of a Retrieval-Augmented Generation (RAG) process.

2.1.3 Agents and Agent Systems

This subsection describes the concept of agents and different agent systems, which will be differentiate into 3 types.

Simple Agent

An agent is nothing else then a smart system, which can use tools or iterate itself based on a LLM. Agents can have a memory to store older conversations. Because of this the agent can "learn" from the user or other agents. This functionalities increase the knowledge base, enables the ability to interact with other software and to bypass the "cutting edge" of a model without fine-tune it.[IWAA24] The "cutting edge" describes the point in time on which the last data, to train the model, was collected. Normally this is a fixed date.

Multi Agents

An multi agent system, illustrated in Figure 2.1.2, is an expansion of the agent concept. There are different approaches to implement such systems. One of the easiest ways to do this is to create a system which contains a supervisor and several workers. The supervisor and the workers are agents. Every worker can be specialized for a task or topic, uses different tools and probably separate memories. The supervisor must handle the conversation or cooperation between the workers and decides when a user's task is completed. Any single agent can use a different model in the background, regarding different topics and tasks. Furthermore, the process of the conversation between the agents is not sequential, but an iterative process.[FLMA24]

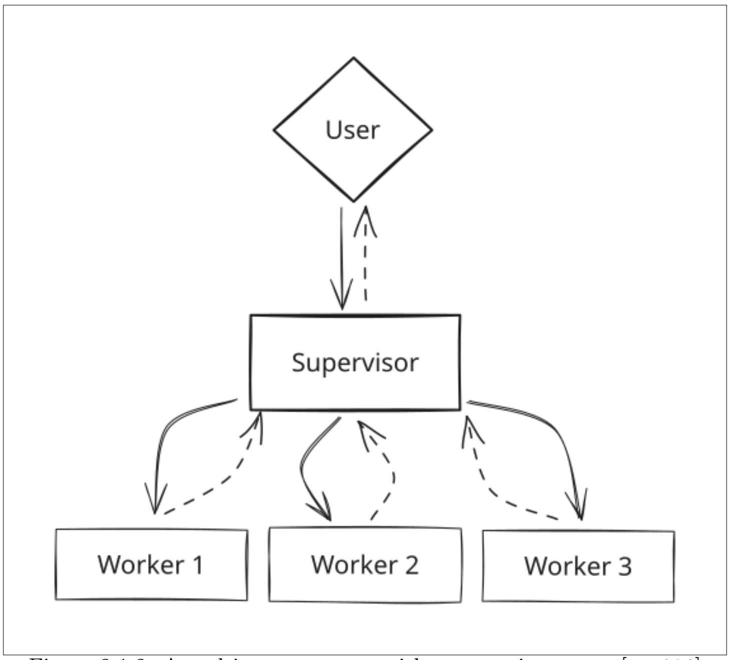

Figure 2.1.2: A multi-agent system with a supervisor agent.[FLMA24]

Sequential Agents

Sequential agent systems differ only in one point compared to multi agent systems. They are not able to iterative on their own volition. The goal of this systems is to create a clear workflow. The agents can use tools, memory and different models, but they are built sequentially. Therefore, an agent or component is only able to run one time. The only exception is, that a repetition of the workflow is possible, when the creator of the workflow uses a loop directly in the workflow.[FLSA24] To get a better understanding of this, take a look at the visualisation in Figure 2.1.3. As example imagine an agent system that should iterate itself to improve the results, but it was noticed that the resource consumption was too high. In this case it would be reasonable to create a system of sequential agents that runs predefined improvements instead of non-traceable once, like in an multi agent system.

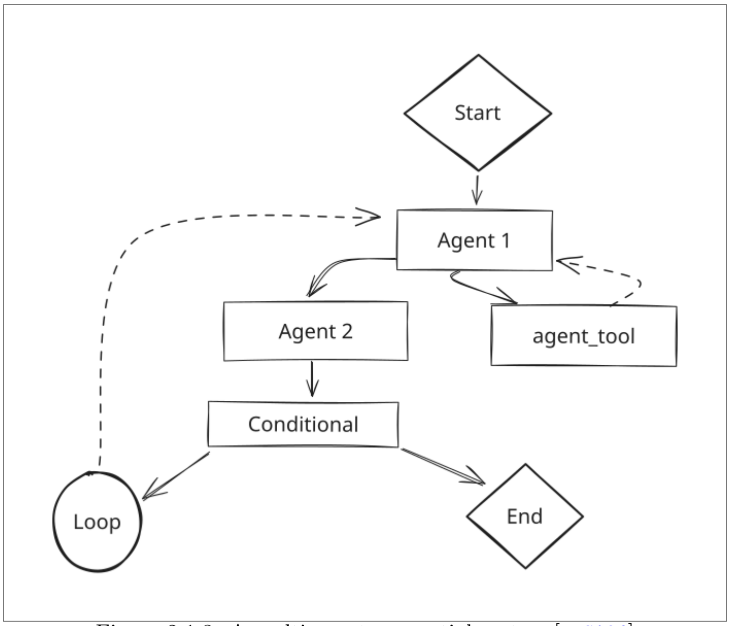

Figure 2.1.3: A multi-agent sequential system.[FLSA24]

2.2 A Reproducible Analysis Platform REANA

REANA is a reproducible analysis platform allowing researchers to run containerised data analyses on remote compute clouds.[GHR24]

For this thesis the REANA platform instance at AIP was used as it is with no modifications. For this thesis, only the definition of a REANA workflow is of importance. The definition of a workflow is done by a yaml file. This yaml file contains different REANA specific parameters, like input files, steps which will be executed and output files. Figure 2.2.1 displays an example "reana.yaml" file. This example is provided in a tutorial series by PUNCH4NFDI[2] written at the AIP. The workflow runs a basic hello world script.

```yaml
inputs:
  files:
    - helloworld.py
workflow:
  type: serial
  specification:
    steps:
      - environment: 'docker.io/library/python:3.10-bookworm'
        kubernetes_memory_limit: '100Mi'
        kubernetes_job_timeout: 60 # seconds
        commands:
        - python helloworld.py
```

Figure 2.2.1: A REANA workflow configuration file which is used to create a REANA workflow.[P4NR24]

Furthermore, REANA provides a web interface and a command line client. The client itself is a Python library, which can be directly used. Also, the library "reana-commons" will be used in this thesis.

[2]PUNCH4NFDI is the German Particles, Universe, NuClei and Hadrons consortium for National Research Data Infrastructure (NFDI), https://punch4nfdi.de.

2.3 Machine Learning (ML) Libraries, Frameworks and Platforms

This section provides references to state-of-the-art Machine Learning (ML) libraries, frameworks and platforms that are relevant for this thesis. Notice that not all components were used for the final prototype.

2.3.1 Ollama

Ollama can be understand as a simple usable platform to run LLMs locally with a limited selection of models. The preselection of models and their configurations make the usage quite easy without a deep understanding of the configurations.[Olla25, MEOL24] The main disadvantage is, that the selection of models is limited.

2.3.2 CrewAI

CrewAI describe itself as "cutting-edge framework for orchestrating role-playing, autonomous AI agents."[CDS24]. The Python based Framework is used to orchestrate and run different Agents. The agents collaborate autonomous as a "crew". The main components are the agents, their tasks and tools. The agents can use defined tools to solve the specific tasks.[CIBM24]

2.3.3 LangChain

LangChain is a framework, which contains different libraries to develop, monitor and deploy whole applications. The libraries which can be used are langchain-core, langchain-community, langchain, LangGraph, LangServe and LangSmith.[LCI24] A detailed description of provided libraries is available at[L1].

2.3.4 LlamaIndex

"LlamaIndex is a framework for building context-augmented generative AI applications with LLMs including agents and workflows."[LII2024] This Framework contain some important components to build RAG systems. The components are data connectors, data indexes and engines. With the predefined agents and evaluation integrations it should be possible to monitor and build workflows.[LII2024]

2.3.5 FlowiseAI

FlowiseAI and LangFlow are so called low/zero code tools. When FlowiseAI is started, it provides a web interface that enables the simple creation of workflows via the click and drop principle. [FI24] FlowiseAI intergrades base utilities[FIU24], but also LangChain[FILC24] and LlamaIndex[FILI24] are (indirectly) integrated.

2.3.6 LangFlow

LangFlow is a visual framework that can be used to build agent and RAG systems. It is open source and runs Python. The simple and user-friendly structure makes it perfect for prototyping.[LFW24]

2.3.7 Open WebUI

Open WebUI is a complex but user-friendly interface to interact with different ML models and different runners (like OpenAI, Claude and Ollama). It brings a useful user management and stores the data locally on the machine. This feature can play an important role in the future due to data protection regulations. The interface itself is accessible as webpage and will be run within a Docker container.[OH24]

2.3.8 Qdrant

Qdrant is a vector store, which is capable to run similarity searches within the vectorized data and to store this kind of data. Qdrant uses different distance metrics. The data itself will be vectorized by a separate embedding model and is forwarded over an API to the vector store. Qdrant can be run in a Docker container, which also provides a web interface.[QWiQ24]
The one larger disadvantage of this vector store is that every collection has a fixed vector dimension size. If a new embedding model needs to be used, it is very likely that all data must be vectorized again.

Chapter 3

Practical

This chapter should provide an overview about the development process. To provide a look onto the development process the development environment will be explained and the first setup of the already introduced tools is provided. Also, this chapter will explain which tools will be used, and why. Furthermore, an overview of large changes in the environment and challenges will be pointed out. At the end of this chapter an evaluation of different models was done to find the best fitting model and some examples for the used tools were given.

3.1 Hardware and Software Environment

This section describes the hardware and software environment. The environment defines the possibilities and limitations of the prototype.

3.1.1 Hardware

The project is developed on a separate virtual machine (VM), which is connected to a dedicated Nvidia Quadro RTX 8000. This graphic card has a GPU memory of 48 GB GDDR6, a bandwidth of 672 GB/s and has a Single-Precision Performance of 16.3 TFLOPS or a Tensor Performance of 130.5 TFLOPS. To get more information about it look at the data sheet.[L3] The Dual-CPU is a 2x24 core Intel(R) Xeon(R) Gold 6252 CPU @ 2.10GHz. The storage for the OS is located on a simple 107GB SSD. The models will be stored on 2 separate 480GB SSDs.

3.1.2 Software

The VM has Debian 12 as operating system. The software packages include Nvidia drivers, Ollama, Docker and Docker compose.

3.2 Infrastructure Setup

Figure 3.2.1 represents the final infrastructure. The llm001 VM hosts different services to build a functional and (nearly) productive ready system. As mentioned above, Ollama is installed in the OS system level. Ollama uses two separate SSDs to store the models and directly accesses the graphic card. Further software components are provided via several Docker containers. The core component is FlowiseAI. FlowiseAI is connected to Ollama to use LLMs. Following chapters will describe in detail how FlowiseAI is connected to Qdrant and PostgreSQL to provide RAGs. A simple RAG implementation which using Qdrant and PostgreSQL is explained in Section 3.6.1. As user interface Open WebUI was chosen. To call the API of FlowiseAI via Open WebUI, the setup of an Open WebUI pipeline container was mandatory. Open WebUI can also be connected directly to Ollama.

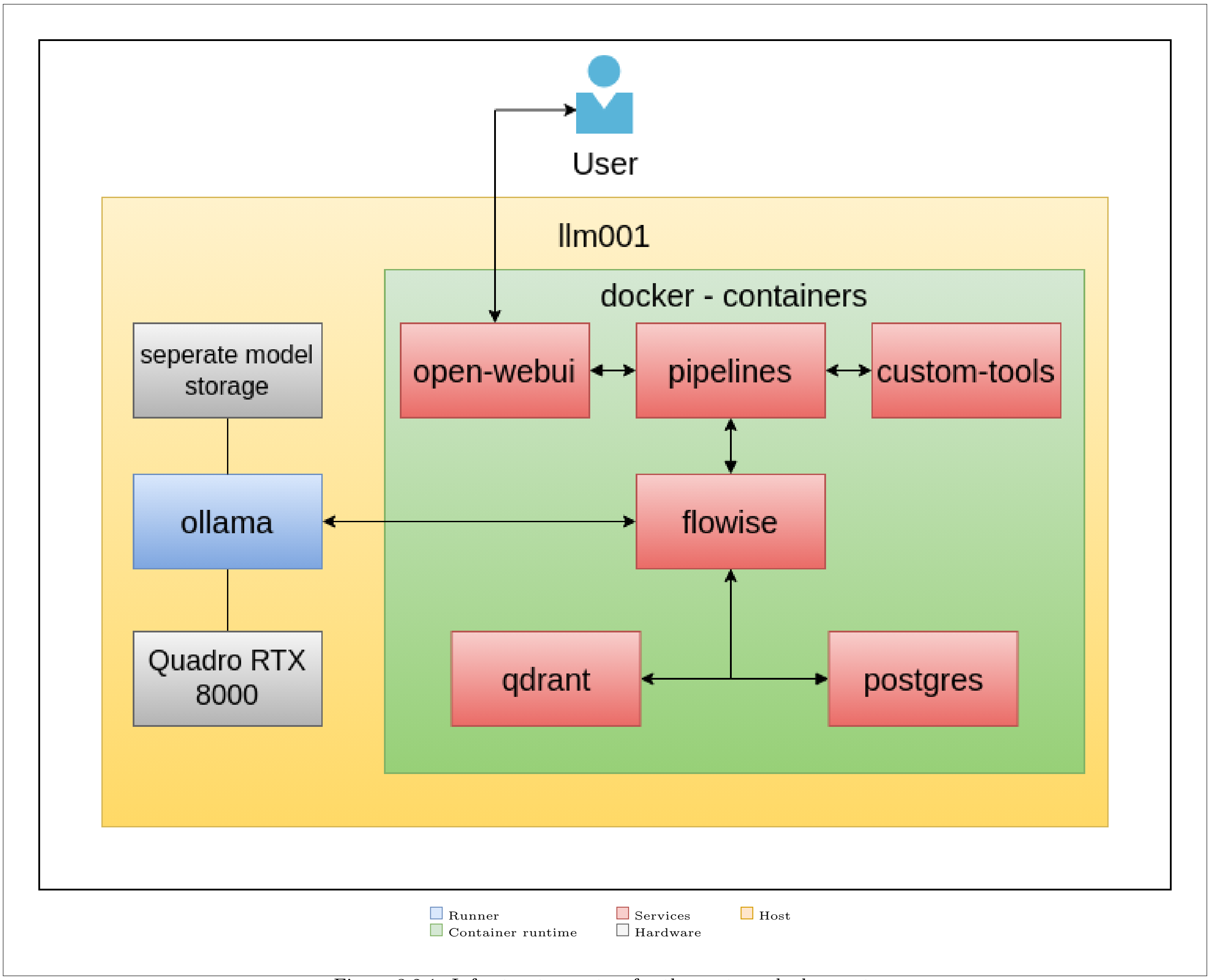

Figure 3.2.1: Infrastructure setup for the protype deployment.

3.3 Setup and Usage

This section explains the first setup and the tools used by the first setup. This do not match the final system, but it shows the approach to implement initial ideas and serves the purpose to get a basis to select tools for the prototype, this will be explained in Section 3.4 in more detail.

3.3.1 Ollama

As described in previous section, Ollama was already pre-installed on the VM. Regarding the central role of Ollama, here the installation instructions for Debian 12 which might differ for different Linux distribution. This applies throughout the thesis.

```
curl -fsSL https://ollama.com/install.sh | sh
```

This command installs or upgrades the Ollama setup. To check if the service is running use "`systemctl status ollama`". To run a model use "`ollama run model:tag`". Following reference contains a list of available models[L2].

3.3.2 FlowiseAI

To use FlowiseAI clone the GitHub repository to a local machine and build an image from the Dockerfile[FGS24]. Although there is the possibility to install FlowiseAI over npm, but the decision to use Docker was made, because it is easier to reproduce results and that many deprecation warnings occurred during the npm installation.

```
git clone https://github.com/FlowiseAI/Flowise.git
docker build --no-cache -t flowise .
```

Start FlowiseAI as Docker container[FGS24]. The web interface of FlowiseAI can be accessed via browser with "`http://localhost:3000`" URL.

```
docker run -d --name flowise -p 3000:3000 flowise
docker ps
```

3.3.3 CrewAI and LangChain

The setup of CrewAI and LangChain is much easier. The only things which are necessary is a Python environment and the pip installations of the 2 tools[LCQ24, CGS24, LFGS24].

```
python3 -m venv <path-to-venv>
pip install langchain
pip install crewai
pip install 'crewai[tools]'
```

3.3.4 LangFlow

In order to install LangFlow we activate an Python virtual environment and running following commands:[LFGS24].

```
pip install langflow
python -m langflow run
```

The LangFlow web interface will be accessible via a browser. After the startup LangFlow should open a new tab by itself. When this does not happen, please follow the steps in the terminal, where you had started LangFlow.

3.3.5 Open WebUI

As there are no specifications for the initial setup, Open WebUI is started by the default "docker run" command from the Open WebUI documentation.[OGS24]
This command will start the container and if not existent on the host will pull the Open WebUI image.

Here a short explanation of the "docker run" parameters. The port forwarding is specified. The default configuration expects to use all GPUs of the host. Notice that the default configuration can possibly affect the efficiency of some other services on the specific host like Ollama. The data volume on host of Open WebUI is specified as well, to preserve data if a container crashes or if a container is migrated to a different host.

```
docker run -d -p 3000:8080 --gpus all --add-host=host.docker.internal:host-gateway -v
↪   open-webui:/app/backend/data --name open-webui --restart always
↪   ghcr.io/open-webui/open-webui:cuda
```

Pipelines

In addition to the setup of Open WebUI, a pipeline container needs to be setup. Open WebUI connects to the container to run user specified pipelines, for example to communicate with the FlowiseAI APIs. This connection has to be set via the Open WebUI web interface in the admin panel under the entry "connections".[OP24]

```
docker run -d -p 9099:9099 \
  --add-host=host.docker.internal:host-gateway \
  -v /root/docker-data/pipelines-openwebui:/app/pipelines \
  --name pipelines \
```

```
  --restart always \
  -e REANA_SERVER_URL=https://reana-p4n.aip.de \
  ghcr.io/open-webui/pipelines:main
```

3.3.6 Qdrant

The Qdrant vector store is pulled as the latest Qdrant image and mounted the internal storage of Qdrant onto a specific directory on the host machine. The Qdrant web interface is accessible at "`http://localhost-or-IP:6333/dashboard`". The API endpoint to load data into the store is located at "`http://localhost-or-IP:6333`".[QLQ24]

```
docker run -p 6333:6333 -p 6334:6334 -v $(pwd)/qdrant_storage:/qdrant/storage:z
↪  qdrant/qdrant
```

3.4 Choice of Tools

Previous section gave an overview of the state-of-the-art methods and tools. Since not all tools should be used, this section discusses the reasoning behind the tool choices for the practical implementation of the prototype.

3.4.1 FlowiseAI vs. Langflow

First, FlowiseAI rebuilds the prototypes with usage of LangChain in the background. Second, it is also capable to load the documents into a vector store, which is a mandatory requirement. For this reason FlowiseAI provides a specific integration in the web interface or an own workflow can implemented. This makes the testing phase simple. Therefore, Qdrant was used in a Docker container setup.
Furthermore, FlowiseAI provides manageable APIs for the workflows. This way, the interactive web interface can be bypassed. The chatbots can connect to a website, but they also can connect to Open WebUI. This also creates an advantage to check outputs and run tests.
The missing user management is currently a main disadvantage of FlowiseAI, but there are some open GitHub requests. This project does not require this feature, but it could be important in the future for the productive setup.
Due to the predominant advantages FlowiseAI was chosen as a main tool for this work.

3.4.2 User Interface

Open WebUI provides an implementation for the Docker and Docker compose setup in their GitHub repository. Also, the connectable pipeline container is an advantage, as it allows to run Python code as a model in the user interface. These were the reasons for choosing Open WebUI regarding time and the general usability.

3.4.3 Vector Store

A vector store was required to implement RAG into the prototype. The Qdrant vector store provides a simple setup and a reliable management. A separate PostgreSQL database keeps track of the records to avoid duplicates entries in connection with FlowiseAI.

3.4.4 Docker and Docker Compose

Docker and Docker compose is a widely used reliable engine for development and productive environments. The usage of data mounts guarantees data integrity. As described in Section 3.5.1, Docker compose replaced simple Docker setups during the development. Docker compose manages multi-container applications in a "`docker-compose.yaml`" file. The code is available in the GitHub repository or in the Appendix C.

3.5 Development Environment and Challenges

This section discusses the successive development of the environment and arising challenges.

3.5.1 Docker to Docker Compose

The usage of separate Docker containers for each service soon arrived at its limits in a larger project. Therefore, the setup was migrated to Docker compose, making the management, documentation and usage of Docker containers a way simpler. The specific yaml files are accessible in Appendix C.

3.5.2 Challenges

Upgrading Ollama

As the field of LLMs experience a surge of new developments almost daily, several new models were published during the development process and some of them were supported by Ollama. The upgrading process requires to run the installation command again but overwrites the previous configuration. The solution is to backup either the service or the configuration file before the upgrade and load/apply the service or the configuration file respectively.

Mounting Storage

It is paramount to careful handle the data mounts while development to prevent data loss and to handle storage size. The "`/etc/fstab`" disk configuration file lists the mounts to automatically mount specified disks in Linux OS at each reboot. If not handled carefully, new pulls of a model can land on the OS disk partitions and overfill them.

Vector Dimensions of a Collection

Our vector store can hold different collections with different configurations. At the beginning several different embedding models were tested, to find one that has a good performance with reduced resource consumption. Every embedding model gives its output in different dimensions. Sometimes the responses from the LLM with RAG context appeared to be very bad. It turned out, that the indexing of the collection was done by a different model, which results in inaccurate responses. Additionally, attention has to be paid while uploading data with a model that did not match the vector dimensions of the collection, because the upload can fail or even worse, the responses by using this data are bad.

Open WebUI Title Generation

Open WebUI is quite like the OpenAI chat interface. On the left side of the interface, there are small buttons with autogenerated headlines to the content of the specific chat. A specific "Task Model" needs to be used to generate the headlines, otherwise Open WebUI might not deliver results or even crash. The setting to configure a "Task Model" can be changed in the admin panel of the Open WebUI web interface.

FlowiseAI Bugs

During this work FlowiseAI was under active development. To prevent it from crashing, it helped to create "dummies" behind tool nodes (see Section 3.7.2). For a productive system, it might take longer time to produce an errorless setup, still it is good for prototyping.

3.6 Model Evaluation

This section examines the evaluation of some models, which where suitable for this project due to their performance and accessibility. All scripts, configurations, reviews and the question catalogue are contained in Appendix A and B.

As expected, in the beginning the models did not understood the concept of REANA or how to build REANA workflows. To determine which model is to be used, an evaluation was performed. The responses of the models with and without RAG context are compared based on a question catalogue. Bar charts are used to illustrate the results which are also provided in a list form in Appendix B. As a note, at the time of this evaluation it was already evident, based on the first usage within this project, that the model "qwen2.5-coder:32b-instruct-q8_0"[OQC25] seemed to generate very good results. This needed to be confirmed through the evaluation results. Alternatively, the model that performs best in this evaluation had to be chosen.

3.6.1 Configuration Details

The vector store contains 8 different yaml files which describe different REANA workflows. Only 4 of them where routed to the tested model as context per question via a RAG implementation, based on the question itself. These files are provided as tutorials and are accessible in [P4NR24]. No further context was given. The temperature was always set to 0.3 to achieve an reasonable but independent result with regard to the provided context. The temperature configuration of the LLM defines the creativity of the answers. With low temperature the answers become more reproducible.[ITem25]

Figure 3.6.1 describes a RAG workflow, which provides an API with which an upload of any file type is possible via a curl request. This API was used by the *vectorization.py* script. The workflow consists out of 5 parts. A text splitter, the configuration of an Ollama embedding model, a record manager to prevent duplicate records, the definition what type is uploaded (in our case an file loader) and finally the configuration to the vector store.

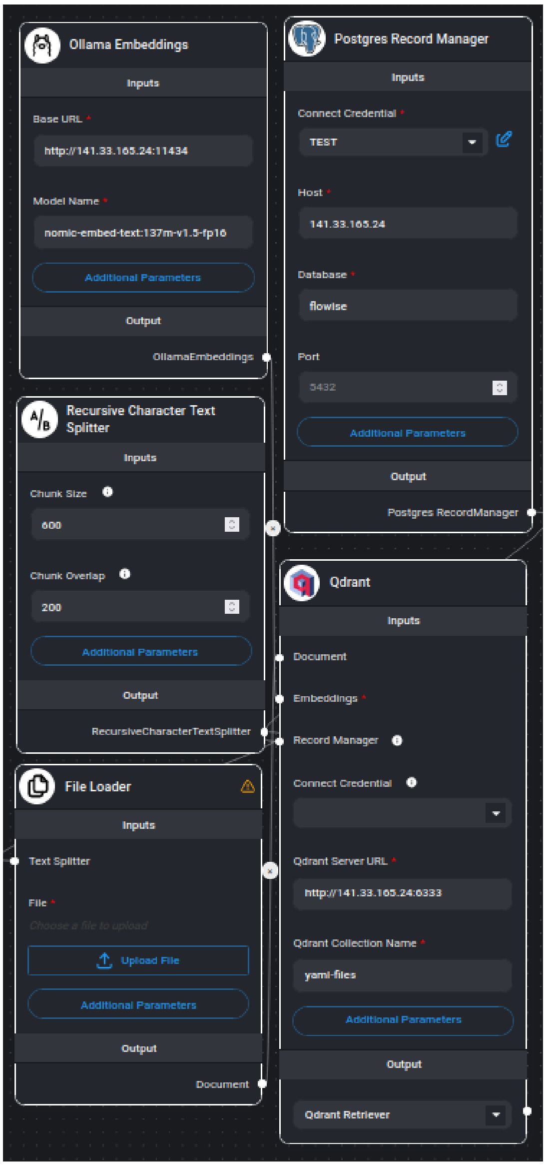

Figure 3.6.1: Implementation of a FlowiseAI workflow to upload files to Qdrant via API.

Without RAG the questions are sent directly to Ollama using the *model_evaluation-_request.py*. Figure 3.6.2 shows a FlowiseAI workflow, which uses RAG and contains 4 components (vector store, embedding model, chat model and conversational chain). The embedding model should be the same as used for the vectorization and a chat model is defined. A conversational chain (designed by FlowiseAI based on LangChain) uses the context of the vector store.

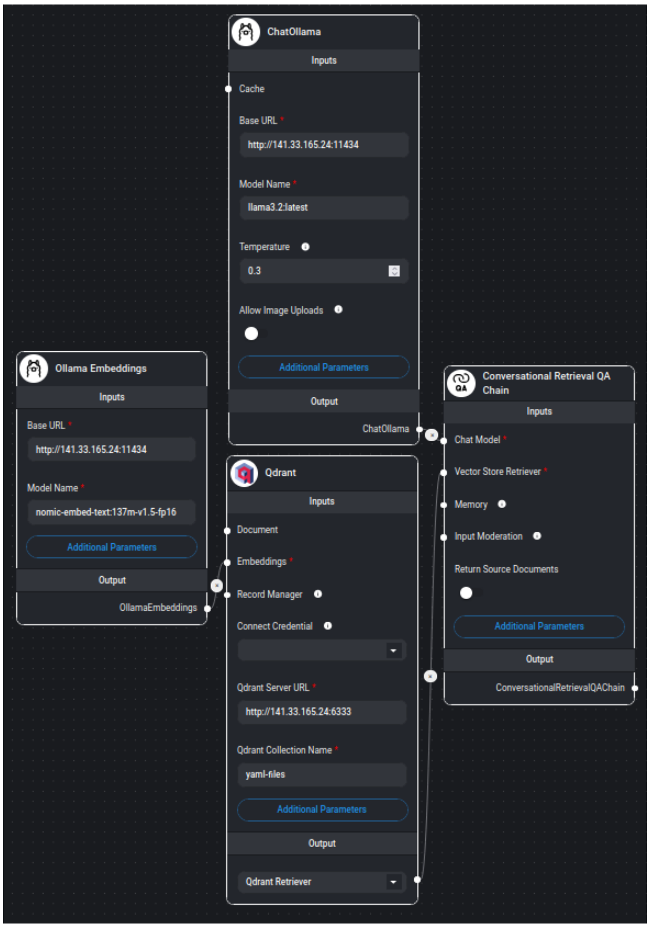

Figure 3.6.2: Implementation of a FlowiseAI workflow for a RAG.

Type of Evaluation

The ratings of the results were carried out by hand to ensure the fulfilment of the task, to create a functioning REANA workflow yaml file. The better results were given a higher score. Nonsense responses were rated 0. Better responses but which did not provide a proper REANA configuration file were rated 1. Correct answered catalogue questions and a proper REANA workflow configuration were rated 2. The same rating system was applied to questions, which do not require the generation of yaml files

This method of evaluation is difficult to apply for other projects as an additional evaluation of a specific task at hand needs to be done, in this case, to run or validate a REANA workflow.

3.6.2 Model Evaluation Overall

Figure 3.6.3 shows how the models perform for the provided question catalogue. In the right bar chart RAG was used and, in the left, direct calls to Ollama were sent. It is evident, that the usage of RAG has a large positive impact onto the responses of the LLMs. As a consequence, RAG was included into the final workflow.

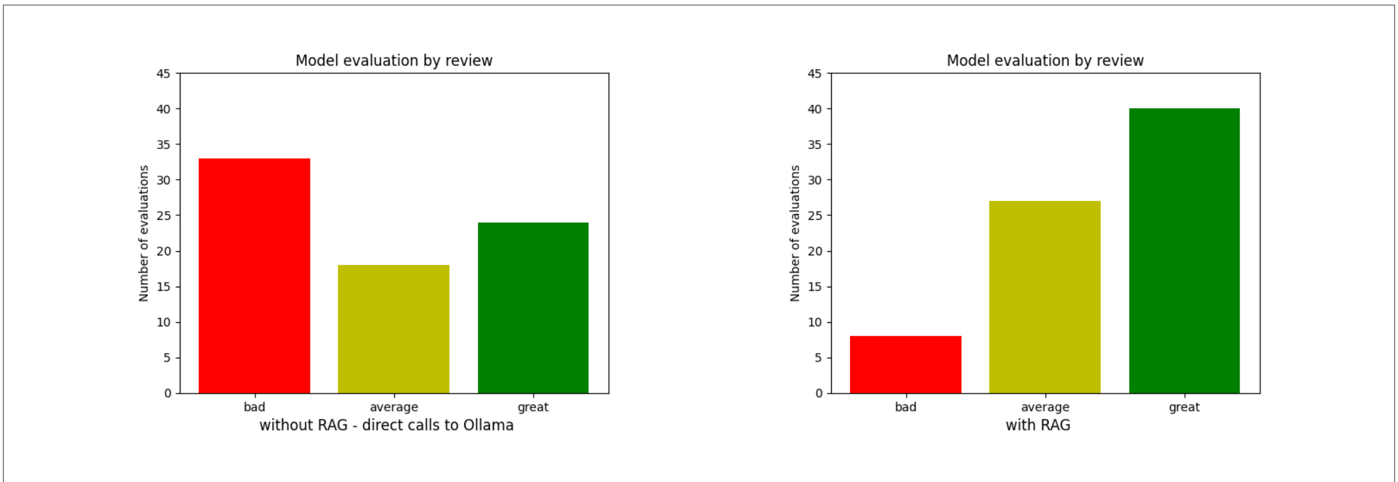

Figure 3.6.3: Comparison of the model performance summarized by rating to illustrate the impact of RAG.

3.6.3 Model Evaluation by Model

In Figure 3.6.4 we can see that the model "qwen2.5-coder:32b-instruct-q8_0" produces the best responses to the given question catalogue. Please notice that in comparison to Figure 3.6.5 the benefit of RAG has not such an impact on this model. But it is possible to provide a clear format, other information like environmental images and specific parameters for REANA workflows via RAG.

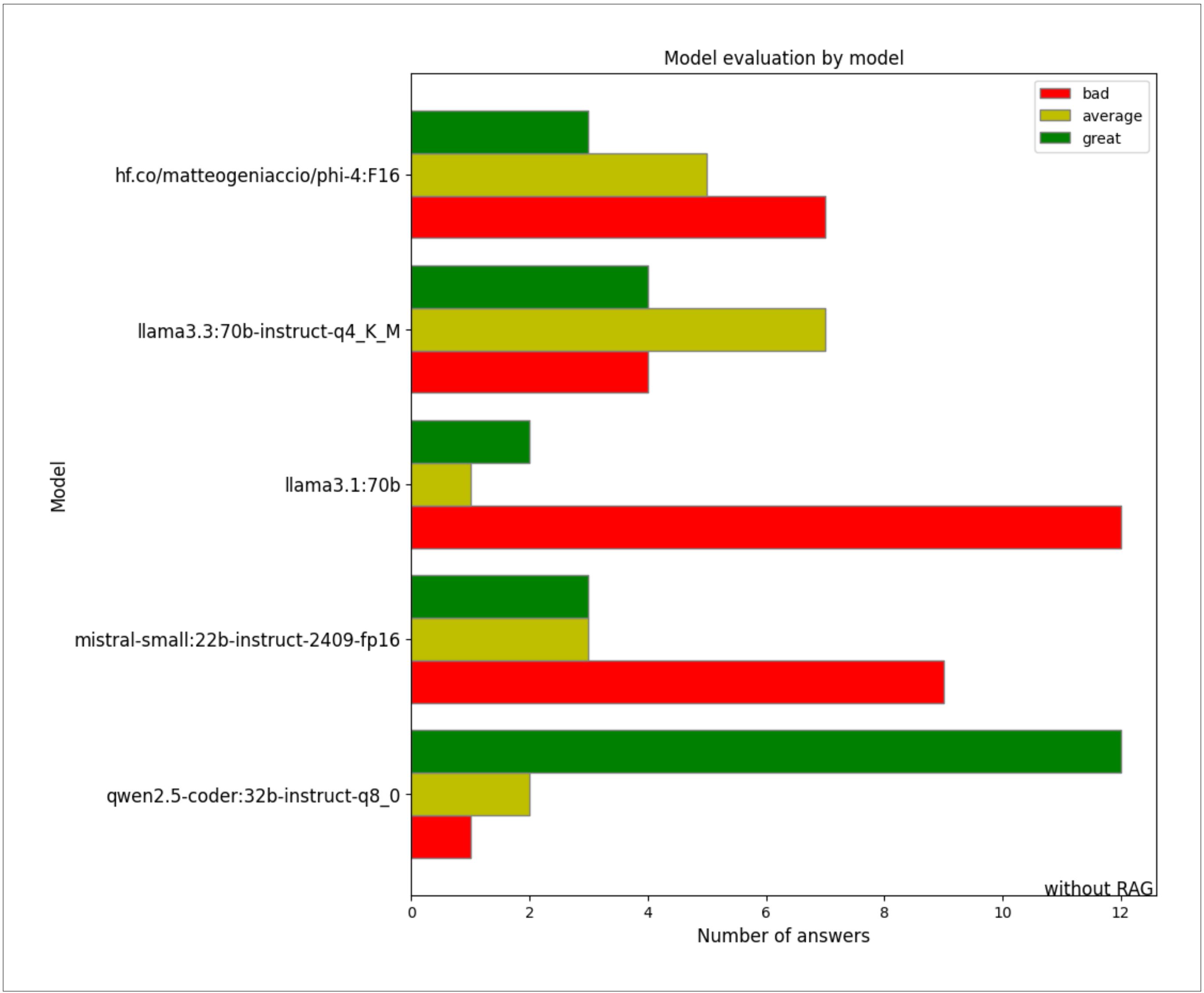

Figure 3.6.4: Model performance for LLM models without RAG.

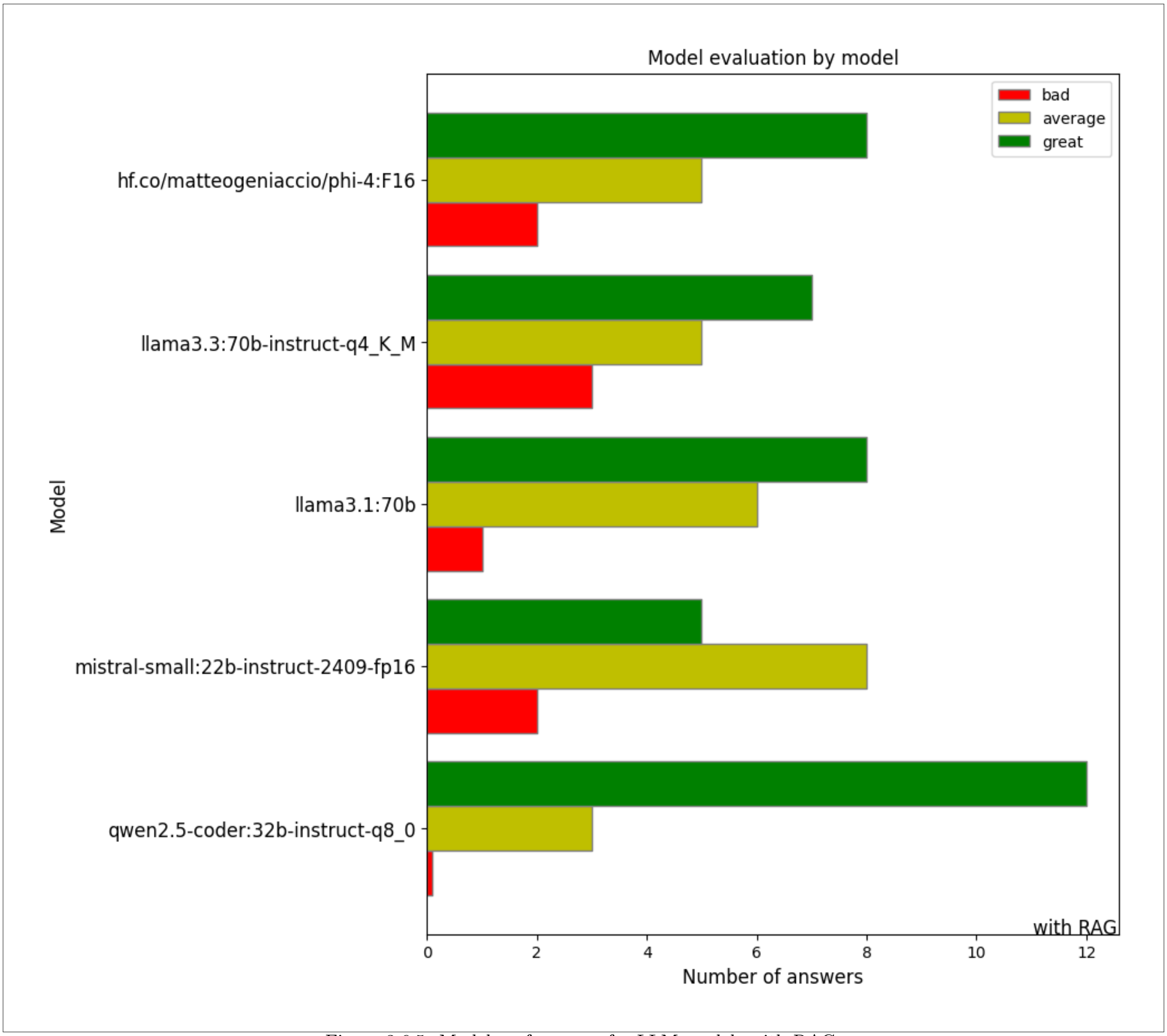

Figure 3.6.5: Model performance for LLM models with RAG.

3.6.4 Model Evaluation by Model in Detail

The tasks and questions were divided into 3 categories. The first category contains general questions like "What is REANA?" or "How can I start a REANA workflow?". The second category encompassed the task to create workflows and scripts. This category will be split into 2 subtopics, "programming" tasks and tasks where specific context is already present via RAG. As mentioned before the comparison of the Figures 3.6.6 and 3.6.7 implies a large impact by the usage of RAG. It is especially noticeable for questions where specific context from the vector store is given. Here, the focus lies on the promising "qwen2.5-coder:32b-instruct-q8_0" model. The results are even more impressive because no prompt engineering was used, and the model was provided only with user input and the vector store context.

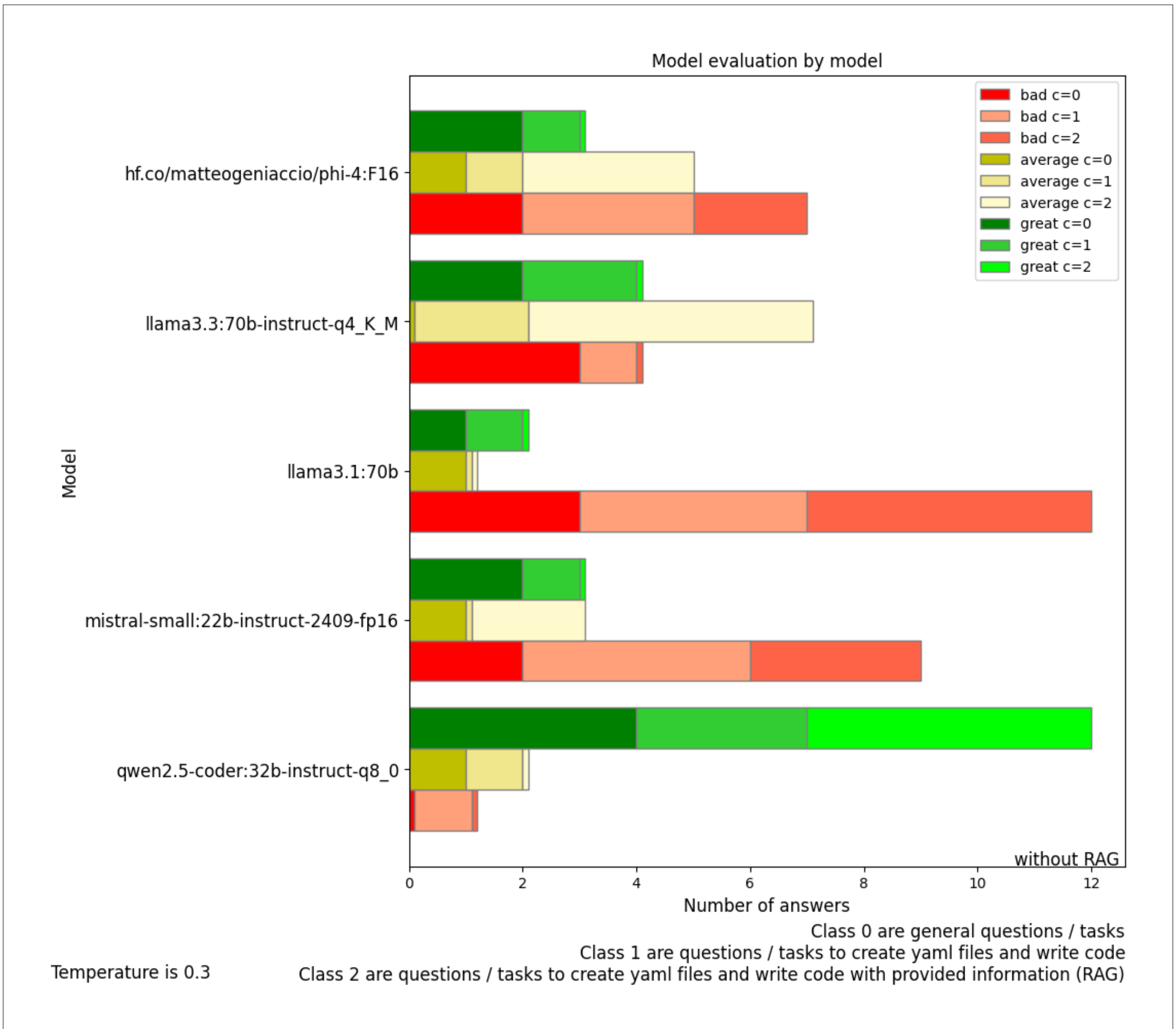

Figure 3.6.6: Model performance for LLM models without RAG by task categories.

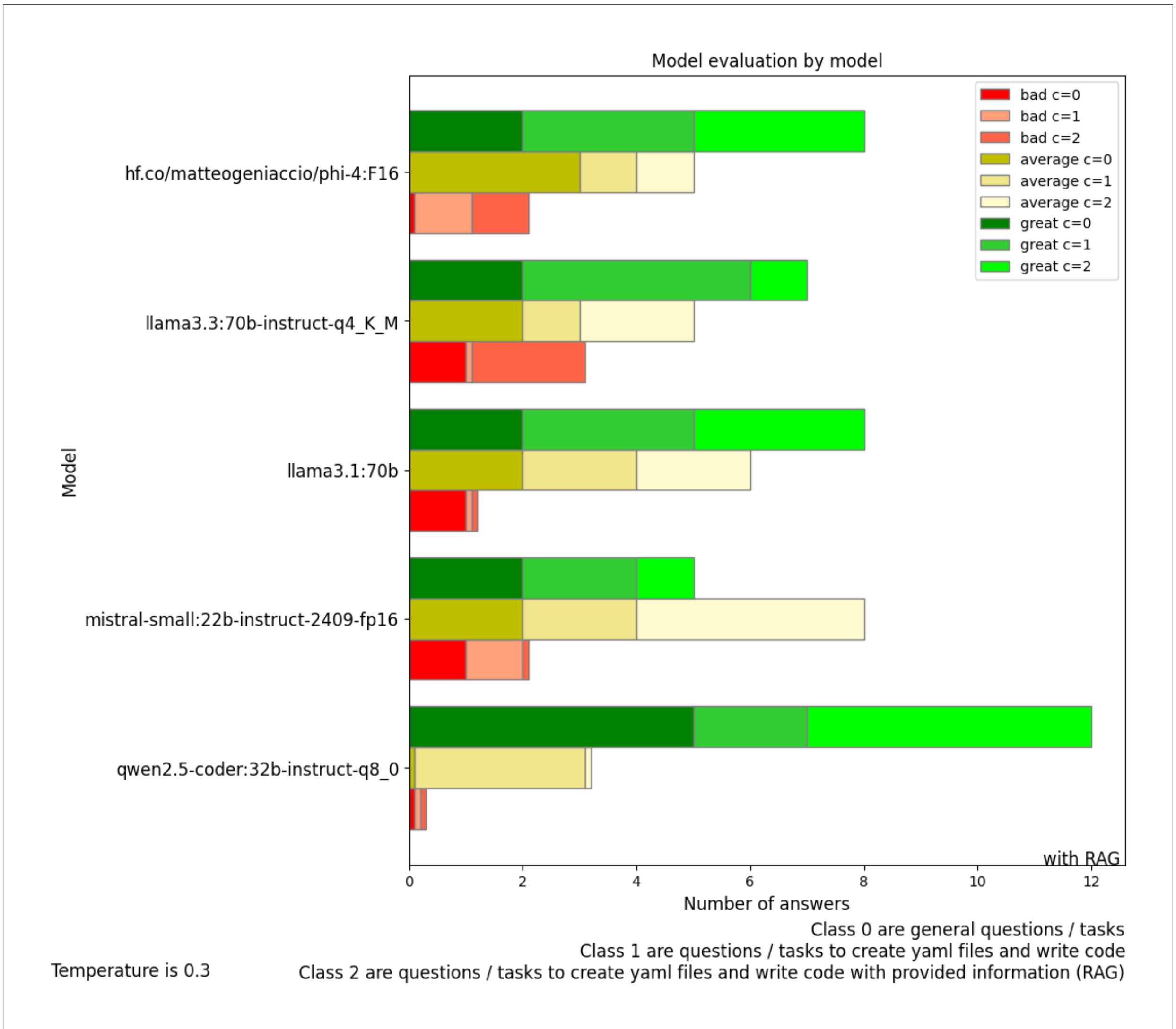

Figure 3.6.7: Model performance for LLM models with RAG by task categories.

3.6.5 Validation

The evaluation confirmed the assumption for the "qwen2.5-coder:32b-instruct-q8_0" model to fit well the given task and to answer the questions catalogue. Accordingly, this model is used in the final workflow/agent system.

3.7 First Steps with FlowiseAI and Open WebUI

This section explains the first steps with FlowiseAI and Open WebUI. To get a basic understanding of FlowiseAI and Open WebUI, there are three main sources of information, which were also used for this thesis.

- A YouTube playlist [LvZF124] for the FlowiseAI general explanations
- The Open WebUI homepage [OH24] helps with the setup of the Open WebUI and the specific pipeline container.
- Different GitHub projects and issues [OwuGh24], [OGD24] and [OGE24] for code examples and pipelines

The first implementation included a chatbot based on the pipeline in [OGD24] which connected a FlowiseAI workflow to Open WebUI and was mentioned in Section 3.7.1. Next, Python code was run via a pipeline [OGE24].

3.7.1 Python Runner with Matplotlib

The following approach extends the first examples with the usage of Matplotlib. The FlowiseAI workflow generate Python scripts and the Open WebUI pipeline runs these scripts. Since plots are not shown in Open WebUI, the Python script have to encode the plot as base64 encoded string and returns it in markdown format.

FlowiseAI Workflow

Figure 3.7.1 shows the steps. First, the starting point and the base model "qwen2.5-coder:32b-instruct-q80" were set. After that the Python files were generated by an LLM node with the following prompt. The user input was given by a separate prompt value.

```
You should write python files. Please only give the clear python code. Format the code
↪  correctly, but please markdown formats. ONLY CODE! No annotations or explanaitons! And
↪  ONLY ONE PYTHON FILE - NOTHING ELSE
```

After that the output of this node was routed to a conditional agent, which decides either the script uses Matplotlib or not. If Matplotlib was not used, the next node should only pass the generated code as output. If Matplotlib was used, a different node rewrites the script to generate the base64 encoded string, with which the display of the plot is possible. To achieve this the prompt below was used.

```
You will get some code to format. It is a python script, that contains matplotlib and
↪  especially plt.show(). You should format it to markdown readable. Therefore, you have
↪  to save the figure into an buffer, format it into base64 and write it into an specific
↪  markdown format + output it. Here an example how that can look like:
# write to buffer
buffer = io.BytesIO()
plt.savefig(buffer, format='png', bbox_inches='tight')
```

```
buffer.seek(0)
# Encode the image as Base64
img_base64 = base64.b64encode(buffer.read()).decode('utf-8')
buffer.close()
# make it markdown readeble
markdown_image = f"![Figure](data:image/png;base64,{img_base64})"
print(markdown_image)  # Output Markdown for embedding
```

The first script was provided like this. The part "{code}" will be replaced with the actual input.

```
This is the actually code:
{code}

IMPORTANT: JUST GIVE THE PYTHON CODE. I WANT NO EXPLANATIONS!
```

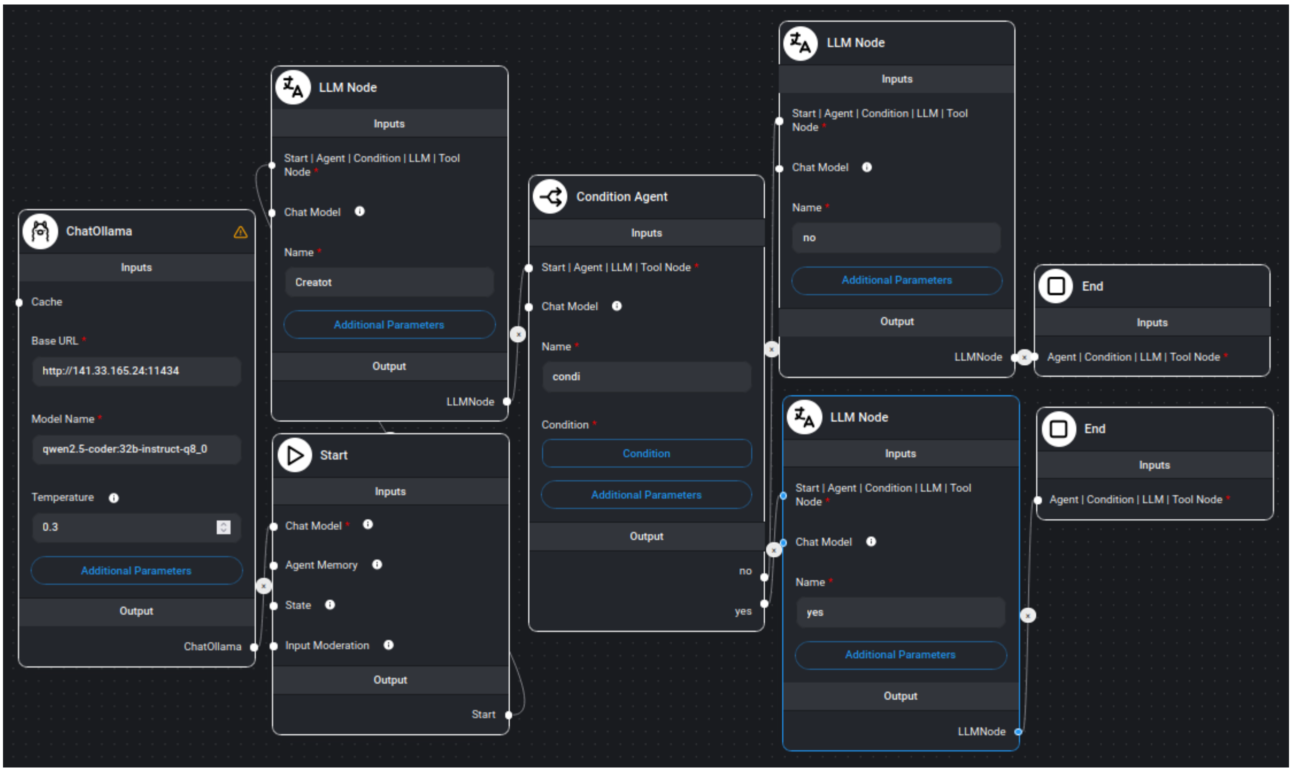

Figure 3.7.1: A FlowiseAI workflow to generate Python scripts incl. the detection of Matplotlib.

Open WebUI Pipeline

The next step was to connect the previous FlowiseAI workflow via its API to Open WebUI and run the Python code. The whole pipeline is accessible in the Appendix D. The parts of the pipeline of particular interest are the connection to FlowiseAI, the implementation of a function to run the Python code and the definition to give Open WebUI a markdown formatted output.
In the code snippet below the pipeline makes a request to the FlowiseAI workflow. The "request" is a built in the Python functionality.

```
url = 'http://141.33.165.24:8000/api/v1/prediction/5467f902-a69f-4bb1-8db7-40a70da88d64'
headers = {'Content-Type': 'application/json'}
data = {'question': user_message}
response = requests.post(url,  json=data, headers=headers)
```

The "execute_python_code" function runs the Python script as subprocess and returns the output.

```
def execute_python_code(self, code):
    try:
        result = subprocess.run(
            ["python", "-c", code], capture_output=True, text=True, check=True
        )
        stdout = result.stdout.strip()
        return stdout, result.returncode
    except subprocess.CalledProcessError as e:
        return e.output.strip(), e.returncode
```

Next, the function is used to request and to build a markdown formatted output. The Python code is executed which is contained in the response. After that the output from the FlowiseAI workflow is set to the first view the actual script and then the output of it. Open WebUI receives the output and displays it to the user.

```
stdout, return_code = self.execute_python_code(response.json()['text'].split('```python')
↪  [1].split('```')[0])
output = response.json()['text'] + '\n script output:\n' + stdout
return output
```

Example

Here the results of the end user output are shown, one with Matplotlib (Figure 3.7.2), and one without (Figure 3.7.3).

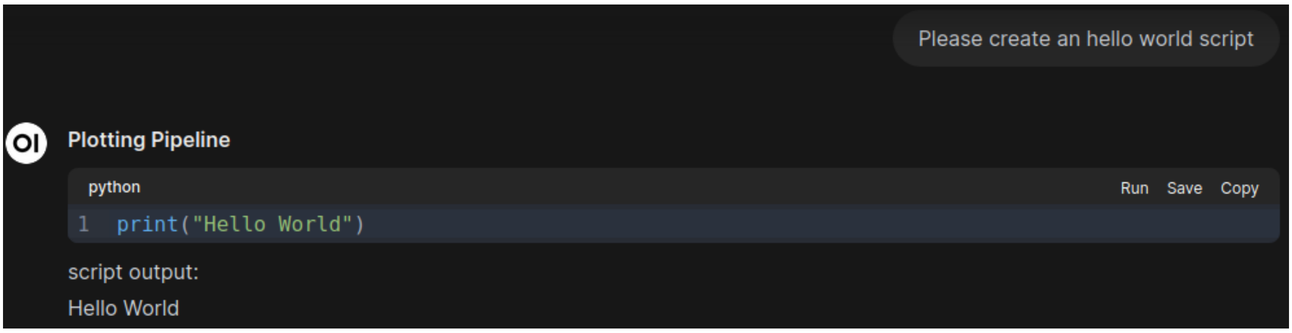

Figure 3.7.2: Open WebUI response to run a Python script without Matplotlib.

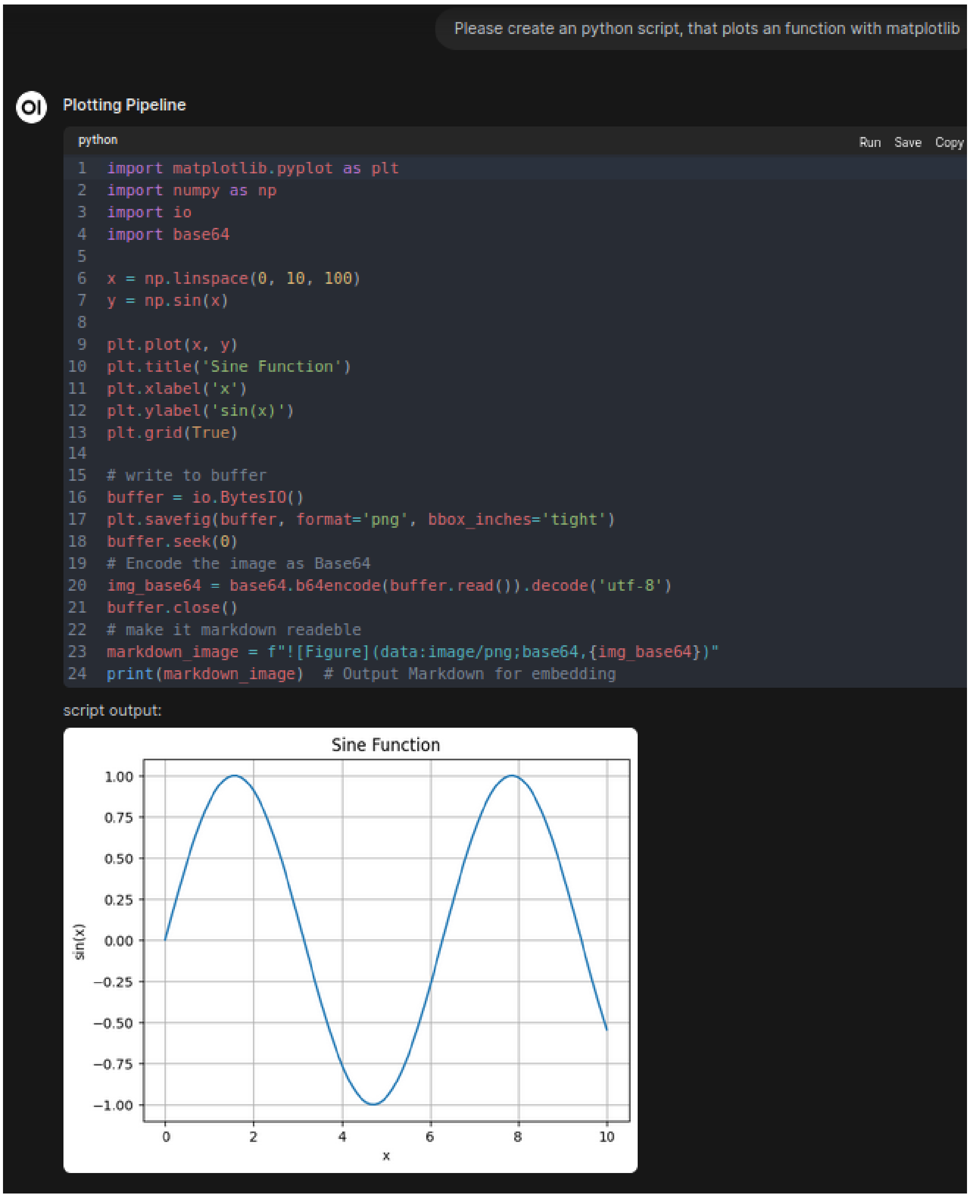

Figure 3.7.3: Open WebUI response to run a Python script with Matplotlib.

Summary

The presented implementation shows the capabilities of FlowiseAI in connection with a Open WebUI pipeline, which executes Python code. It provides the proof of concept, that the thesis question can be solved by the suggested methods. It also makes clear that many different tools, implementations and prompt engineering are required for the solution.

3.7.2 Implementation to Generate and Run REANA Workflows

Based on the acquired knowledge the next step was to build a system to generate and run REANA workflows. The example provides the system prompts. Further input is given via prompt values.

FlowiseAI Workflow

The extended FlowiseAI workflow is divided into parts (Figure 3.7.4, Figure 3.7.5, Figure 3.7.6).
The first part only shows the starting point and the settings of the base and embedding model. The base model, which is used by all nodes as default was set to "qwen2.5-coder:32b-instruct-q8_0" and as embedding model, which was used to vectorize the user prompt to use RAG, "nomic-embed-text:137m-v1.5-fp16" was chosen.

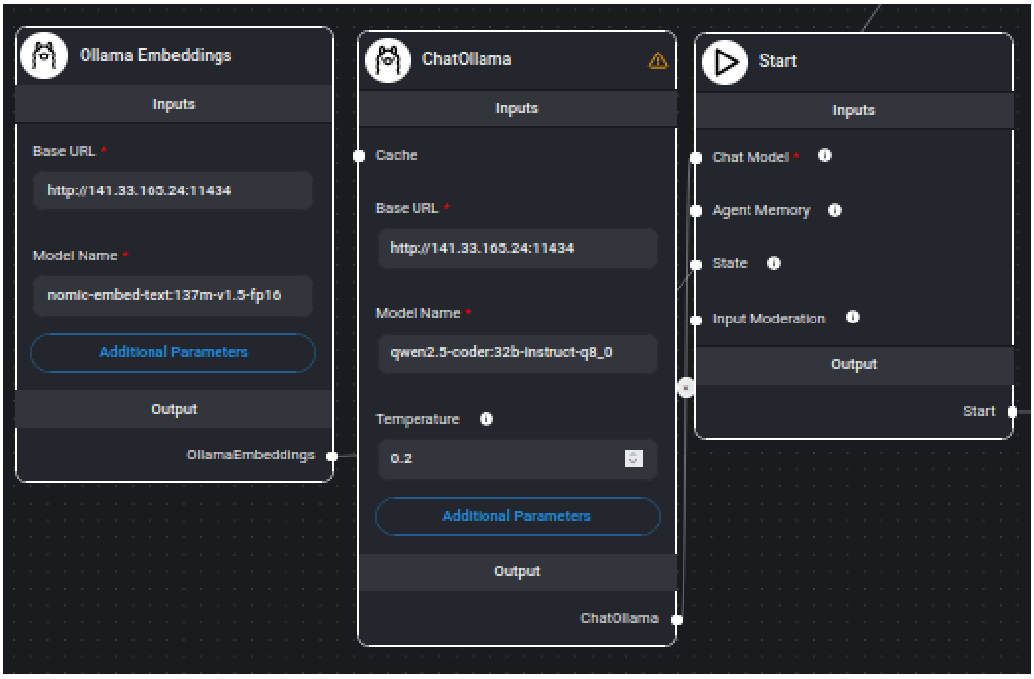

Figure 3.7.4: A FlowiseAI workflow to generate a REANA workflow, Part 1 of 3

The second part explains the usage of the tools. An LLM and a tool node use a retriever to get context from a vector store. A retriever component in FlowiseAI can be understood as link between the tool node and the vector store configuration. The vector store contains the different yaml files, as explained in Section 3.6. Notice that the functionality of this system can only be guaranteed, if a "credentials.py" file with a REANA token is located in "/app" in the pipeline container due to the required authorization. REANA URL has to be set as environmental variable within the Docker container in the "docker-compose.yaml" file for the pipeline container. Here is the prompt of the first LLM node.

```
You are an assitent to create reana workflows as yaml files.
Please generate your answer based on the user input.
PLEASE ONLY CREATE THE YAML AND PYTHON FILES. NOTHING ELSE!

IMPORTANT: JUST GIVE THE PYTHON CODE. I WANT NO EXPLANATIONS!
```

The tool node prompt will be set to the default FlowiseAI value and the retriever tool prompt reads:

```
This Retriever is connected to a vector store. This vector store contains example yaml
↪  files for REANA workflows.
```

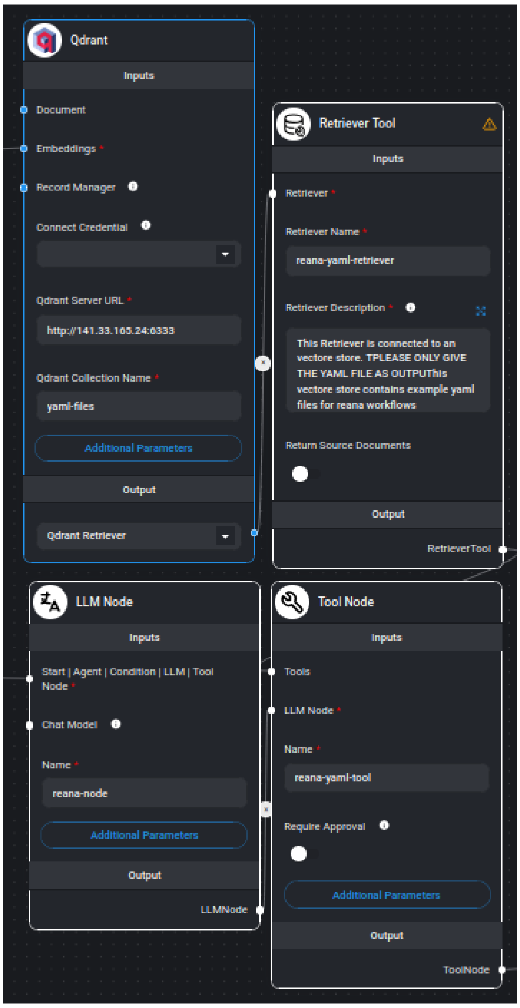

Figure 3.7.5: A FlowiseAI workflow to generate a REANA workflow, Part 2 of 3

In the end two LLM nodes were used to generate the answer. The "dummy" passes the tool node output and bypasses some of the FlowiseAI bugs. Here is the prompt for the next step.

```
DO NOT RESPOND. DO NOT CALL TOOLS. JUST PASS THE INPUT FORWARD!
```

The second node actually generates the response with the following prompt.

```
Based on the given input, please generate a reana workflow yaml file based on the users
↪ input. Please use this as enviroment if possible
↪ 'gitlab-p4n.aip.de:5005/p4nreana/reana-env:py311-astro.9845'. Don't run pip commands!
↪ Please fit your anweser to the given example. Don't add other paramerts. Note that the
↪ paramter "version" have do be string type(version: '0.13').
PLEASE ONLY GIVE THE YAML FILE AND PYTHON FILES AS OUTPUT. NOTHING ELSE!
Format should be:
```
code
```
```
code
```
...
```

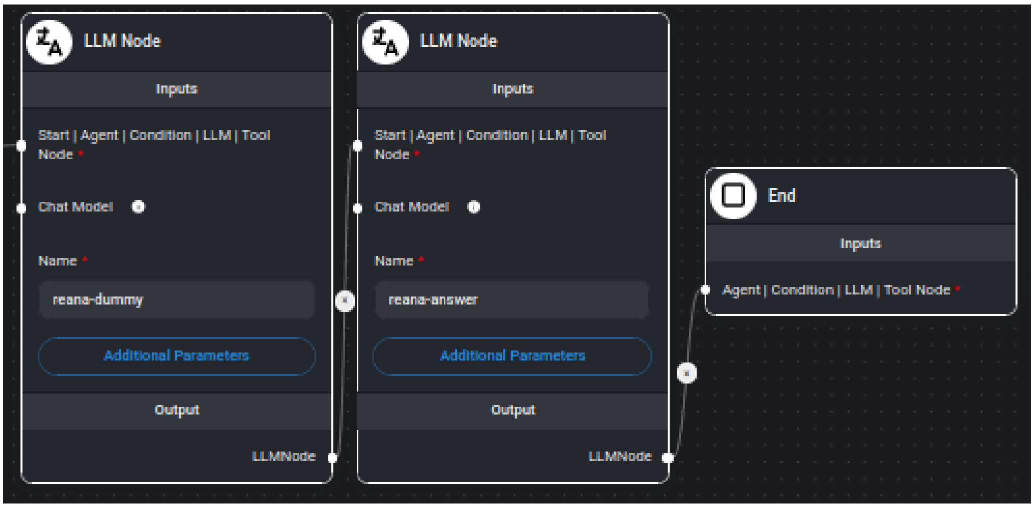

Figure 3.7.6: A FlowiseAI workflow to generate a REANA workflow, Part 3 of 3

Open WebUI pipeline

Open WebUI is connected to the extended workflow in the same way as before. This subsection explains the main functionalities and the markdown formatting for the output. The complete pipeline is again available in the Appendix D.

This new pipeline needs much more imports which require further libraries' dependencies. The dependencies are specified in the pipeline.

```python
import reana_commons.validation.utils as rcv
import yaml
import re
import reana_client.api.client as rcl
import credentails
```

The core of this pipeline are the REANA command line tools. These tools are programmed in Python and their functions can be used in any Python script. The following code shows how the response of the LLM can be split to get the different files, which are needed to run a REANA workflow. The split is only possible since the workflow generates a specific file format. The first file is a definition of a REANA workflow file in yaml format. This definition will be validated with the function "validate_reana_yaml". If this succeeds, the workflow is created with the "create_workflow" command. Then the Python files will be uploaded with the "upload_file" command. Finally, the workflow is started with the "start_workflow" command. At this point, the REANA workflow runs on a REANA cluster and the results can be accessed via the REANA web interface.

```python
validation = False
reana_file = {}
files = []
file_list = []
try:
    files = re.split(r'```.*?\n', text)
    files[len(files)-1] = files[len(files)-1].replace('````', '')
    files = list(filter(lambda item: item not in ['\n', ''], files))
    reana_file = yaml.safe_load(files[0])
    file_list = reana_file["inputs"]["files"]
    if(len(rcv.validate_reana_yaml(reana_file)) < 1):
        validation = True
except Exception as e:
        print("Error:", e)
        validation = False

if(validation):
```

```
    rcl.create_workflow(reana_file, "hello", credentails.reana_api_key)
    for i in range(1, len(files)):
        rcl.upload_file("hello", files[i], file_list[i-1], credentails.reana_api_key)
    rcl.start_workflow("hello", credentails.reana_api_key, None)
    return "# This yaml specifications should work and will be run\n" + text
else:
    return "# The response is not a valid REANA YAML. Please generate again. \n" + text
```

To create an end user response, different outputs are generated if the run is successful. The user receives a short message that the REANA workflow was started. If the generation did not work or the validation failed, the user also receives a response.

Example

Figure 3.7.7 shows how a request via Open WebUI generates and starts a REANA workflow, which uses 2 python files to create a plot of an sinus function.

Figure 3.7.7: Open WebUI response to create a REANA workflow and start a job on the REANA cluster.

The 2 following Figures (3.7.8, 3.7.9) are showing, that the workflow ended successfully and the plot was saved as well.

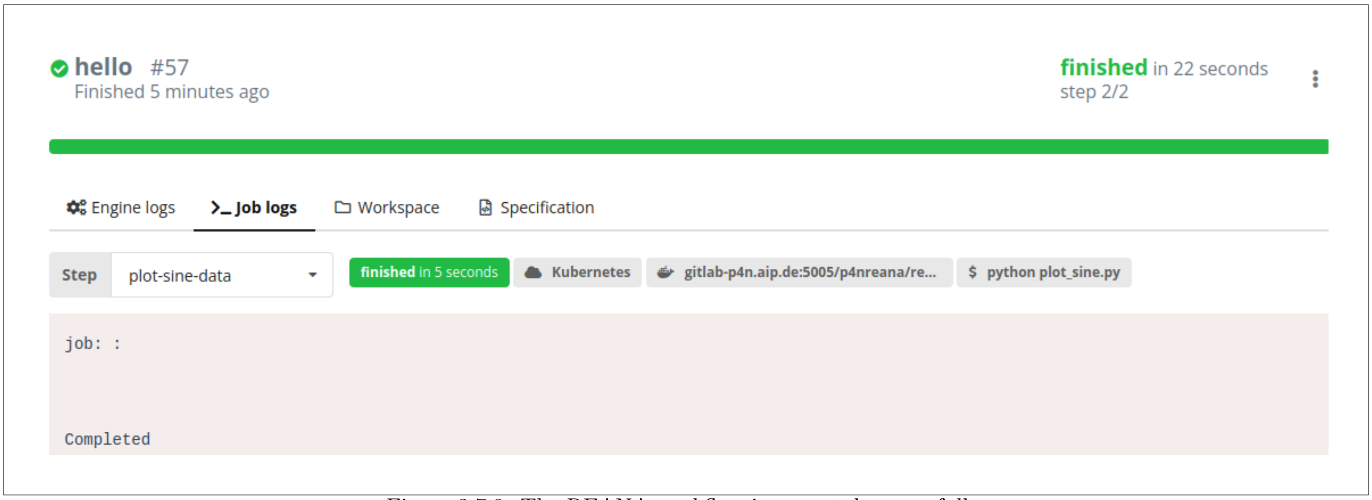

Figure 3.7.8: The REANA workflow is executed successfully.

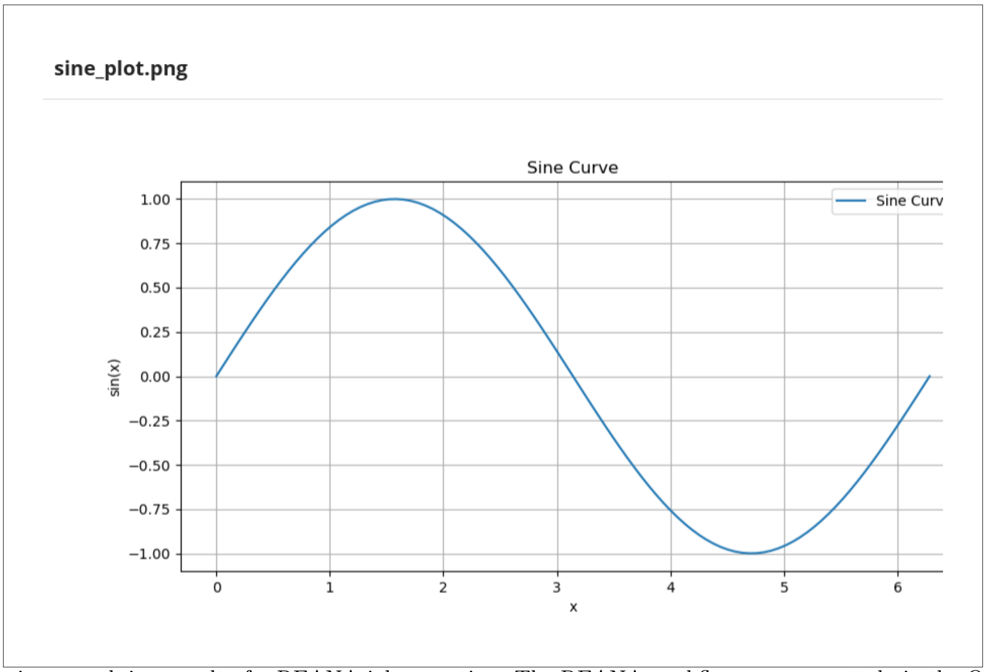

Figure 3.7.9: A sinus graph is a result of a REANA job execution. The REANA workflow was generated via the OpenWebUI web interface and successfully executed on the REANA platform.

Chapter 4

Final Setup

This section describes the final prototype. The proposed solution for the automated generation of REANA workflows using LLMs in connection with state-of-the-art technologies provides the proof of concept and can be used as a starting point for the development of productive systems. The prototype has following main tasks:

1. generate and modify a REANA workflow,
2. upload the generated workflow to REANA platform,
3. create a GitLab repository for the REANA workflow configuration files,
4. answer general questions
5. give hints when the system is wrongly or harmfully used.

Figure 4.0.1 illustrates a sequential processing of the request via agens and LLMs on a basis of a FlowiseAI workflow as described in Section 4.1.2.

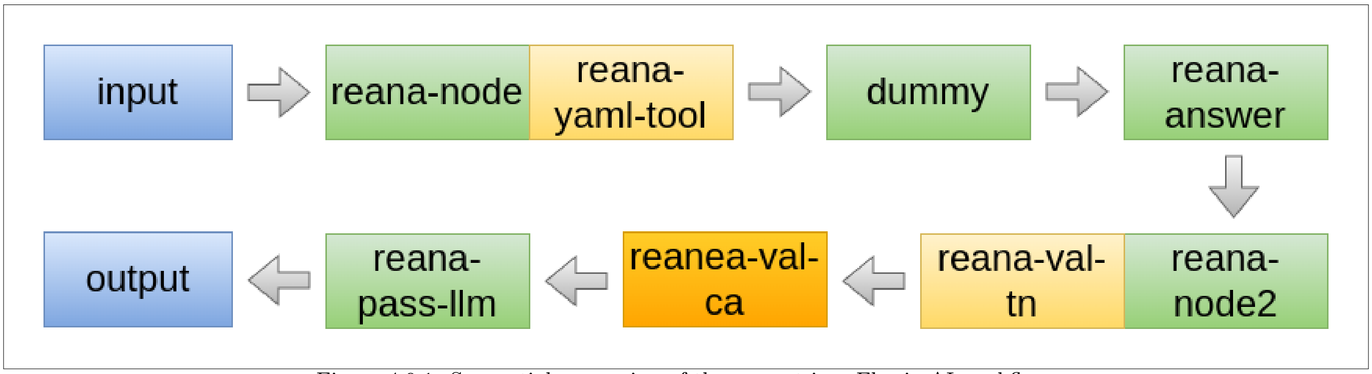

Figure 4.0.1: Sequential processing of the request in a FlowiseAI workflow.

The implementation of the tasks is discussed in the next sections. More examples and figures are provided in Appendix E.

4.1 FlowiseAI Workflows

The final setup is built on top of 2 different FlowiseAI workflows. The first one generates and adjusts REANA workflows. The other one provides an endpoint for the Open WebUI pipeline. The source code for the workflows is accessible in the GitHub repository[1].

4.1.1 Output Standard Format

The first workflow generates different REANA workflows in a specific format. The output of this workflow should be:

````
```
yaml file
```
```
python file
```
```
python file
```
.
.
.
````

The specific output is achieved through prompt engineering.

4.1.2 REANA Generation Workflow

First, the base model "qwen2.5-coder:32b-instruct-q8_0" for this workflow is defined and the "state" for variables is set. The starting point is connected to the first LLM node. The LLM node uses the tool node. The RAG process requires a retriever, a vector store configuration and a specific embedding model, "nomic-embed-text:137m-v1.5-fp16" model in this case. To stabilize FlowiseAI, the output is passed through a "dummy" node, as described in Section 3.7.2.
Next, the yaml and Python file(s) for the REANA workflow are generated by a more complex prompt. The markdown string is saved in the pre-defined state/variable, so that another LLM cannot change the original. The string is validated by the next LLM and the tool node. A validation tool was developed for this purpose and is described in Section 4.3.2. The conditional agent decides if the validation was successful or not based on the validation output. If the validation failed the follow-up LLM node will address the user to rephrase the task or question. If the validation was successful, the follow-up LLM node will forward the markdown output string with the generated yaml and Python files.

[1] https://github.com/etlstrauss/bachelor-thesis-public

Last, the LLMs nodes are connected to an endpoint component. The Figure 4.1.1 provides an abstraction of this FlowiseAI workflow.

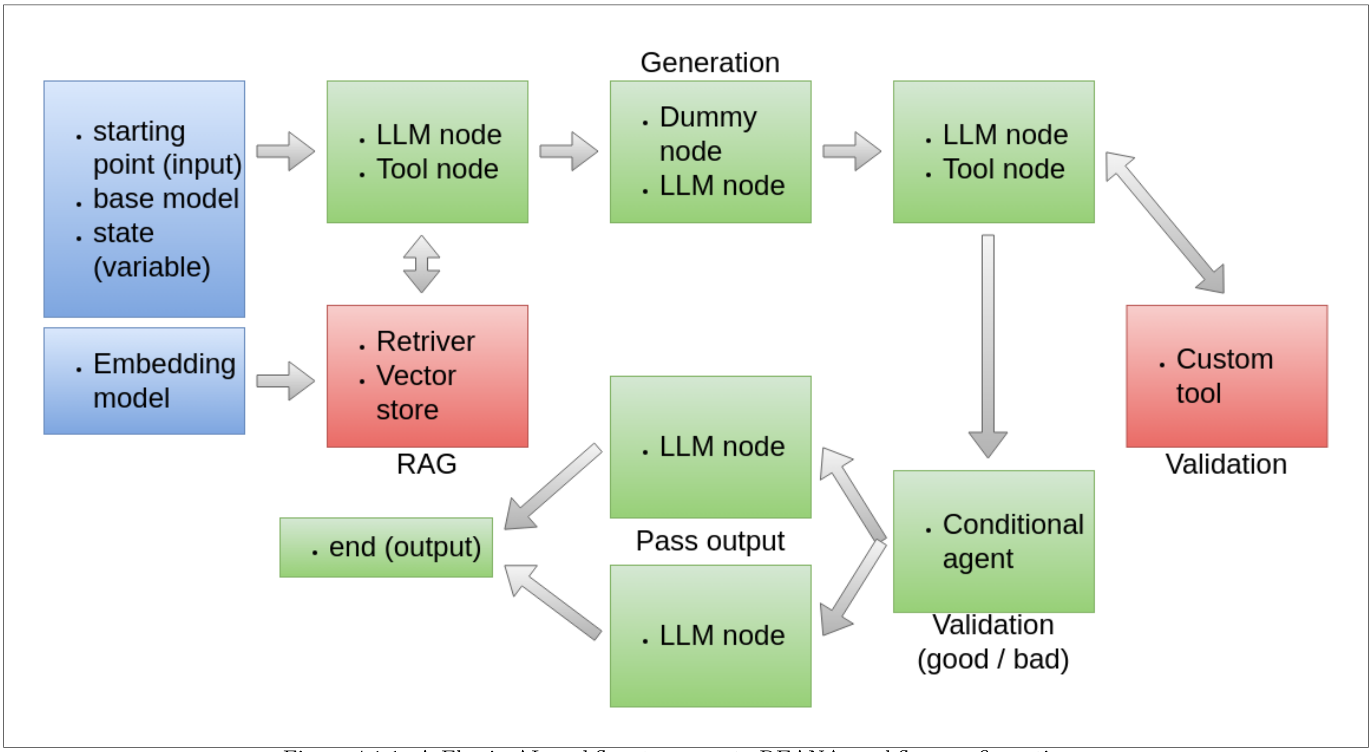

Figure 4.1.1: A FlowiseAI workflow to generate REANA workflow configurations.

4.1.3 Open WebUI Endpoint Workflow

The Open WebUI endpoint workflow, which was abstracted in Figure 4.1.2, provides the API to connect the Open WebUI pipeline container to FlowiseAI. The user has four options to choose from: to generate a REANA workflow, to upload/start a workflow to/on REANA or to create a GitLab repository with the created workflow. Furthermore, the system could process questions from the question catalogue. A drawback of this system is that if no REANA workflow was initially requested or provided by the user, the LLM node might invent a fake REANA workflow, by creating a GitLab repository or start a REANA workflow.

First, as for the previous workflow, the base model "qwen2.5-coder:32b-instruct-q8_0", a state and a starting point are set. The first LLM node is powered by a specific guard model "granite3-guardian:8b-q5_K_M". This model checks the user input for harmful content or jail breaks. If the guard model is triggered, the user runs an empty node. If the query passes through the guard, a conditional agent behind a "dummy" node decides if which of the listed options should be chosen. The LLM node and the tool node are connected behind every decision of the conditional agent. Custom tools or the previous REANA generation workflow is used.

A separate LLM node plus the "dummy" node are handling the output and connect to an end component. The output is in these cases a generated or adjusted REANA workflow, a link to a Gitlab repository or an ID of the REANA workflow run on the REANA platform. The question node output will not be processed with other nodes.

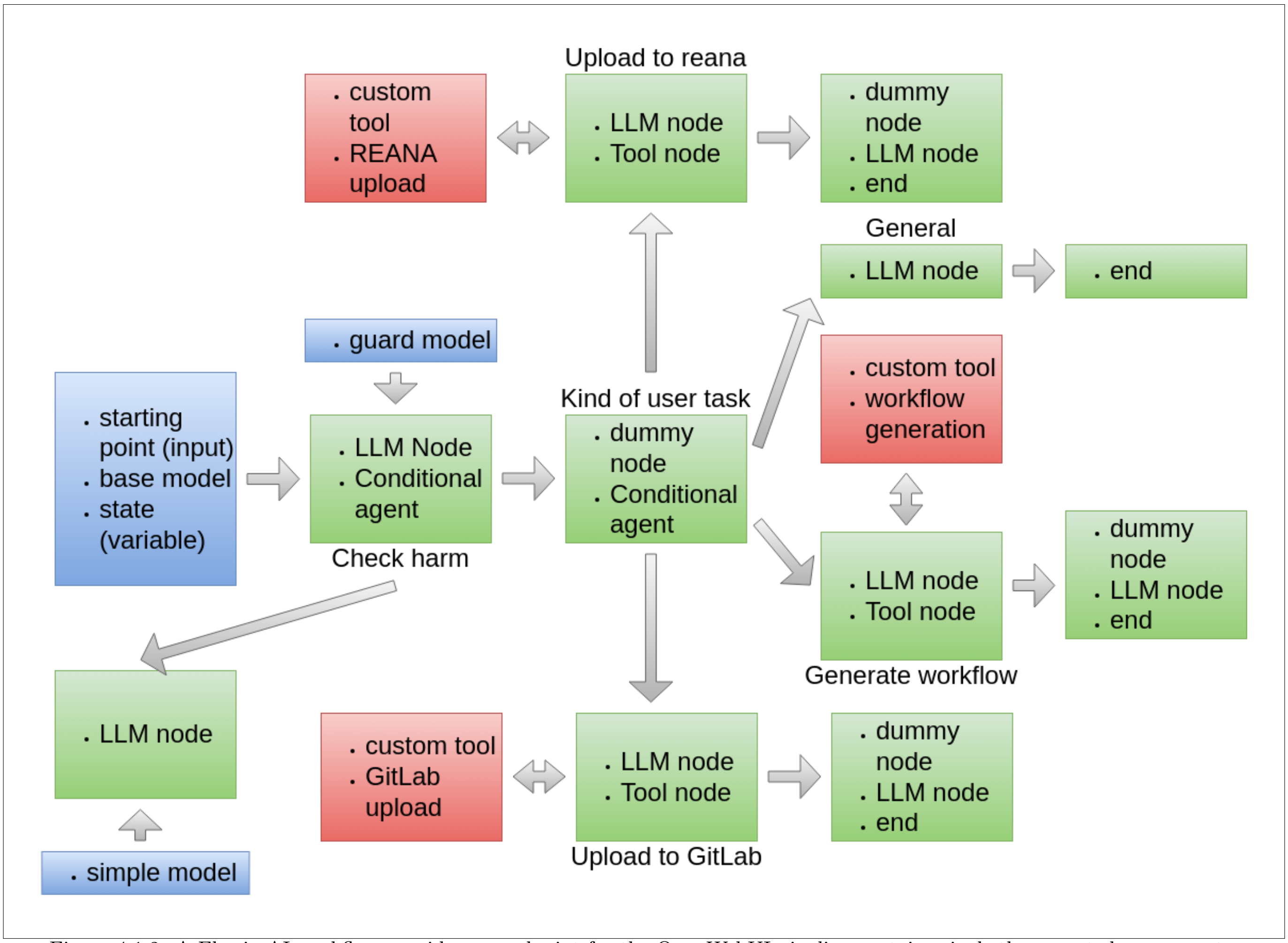

Figure 4.1.2: A FlowiseAI workflow provides an endpoint for the OpenWebUI pipeline container incl. the executed components.

4.2 Open WebUI Pipeline

The Open WebUI pipeline connects the Open WebUI endpoint workflow from the FlowiseAI Docker container to Open WebUI and is accessible in the Appendix E. The last message is appended to the user input as context and is sent to FlowiseAI as an API call. The following code block demonstrate how this was done using the initial "user_message" and the list of messages, which will be provided by Open WebUI.

```
if(len(messages) > 1):
            user_message = user_message + "\n History:\n" +
            ↪  messages[len(messages)-2].get("content")
```

4.3 Custom Tools

This section describes the custom tools implemented within this thesis project. The tools are provided inside a docker container.

4.3.1 General Structure

The docker container is run via "docker compose". It is built with a Dockerfile. The Python libraries are installed on top of the image and are included in the "requirements.txt". Furthermore, a specific credential file provides REANA platform and Gitlab tokens and/or ssh key for authorization. These components are specified in the "docker-compose.yaml" file, displayed in Figure 4.3.1. Also an environmental variable defines the URL of the REANA service at the AIP. "host.docker.internal" is mapped to "host-gateway" to enable the communication with the container APIs. The container restart policy is set to always so that the container always restarts after a crash or a reboot of the host.

```yaml
services:
  aiptools:
    build:
      context: .
      dockerfile: Dockerfile
    image: aiptools:latest
    ports:
      - 5000:5000
    environment:
      - REANA_SERVER_URL=https://reana-p4n.aip.de
    extra_hosts:
      - host.docker.internal:host-gateway
    restart: always
```

Figure 4.3.1: Docker compose yaml file of tool container, which exposes the internal structure to understand the parameters, which are necessary to guarantee the functionality of the container.

The Docker container image is based on the official "python:3.12-slim" image. Figure 4.3.2 shows the Dockerfile, which sets the working directory to "/app" and uses "apt-get update/install" to install 2 REANA specific libraries and "gitpyton". Also it copies the "app.py", "credentails.py" and "requirements.txt" (described in Figure 4.3.3) files into the working directory. The "requirements.txt" Python libraries are installed via pip. The port 5000 is exposed and the app is started.

```
FROM python:3.12-slim

WORKDIR /app

```

```
RUN apt-get update && \
    apt install build-essential -y

RUN apt-get update && apt-get install -y git

COPY app.py /app
COPY credentails.py /app
COPY requirements.txt ./

RUN pip install --no-cache-dir -r requirements.txt
RUN pip install flask

EXPOSE 5000

CMD ["python", "app.py"]
```

Figure 4.3.2: Tool container Dockerfile.

```
reana-client
reana-commons
gitpython
```

Figure 4.3.3: "requirements.txt" file contains the Python library dependencies.

4.3.2 Tools

The "app.py" file imports the validation tool, reana_client, further general libraries, the credentials and initializes "flask".

```
from flask import Flask, request, jsonify
from reana_commons.validation import utils as rcv
import reana_client.api.client as rcl
import yaml
import re
import git
import os
import uuid
import credentails

app = Flask(__name__)

[functions]
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0')
```

REANA Workflow Validation

The validation tool is a wrapper around the "reana-client" library. It extracts the data out of the request via a specific Json key and the string is loaded as "yaml" format as expected by the "reana-client" library. The "validation" function of the "reana-client" library is used to verify the given REANA workflow. Either the function returns "validation:True" or "validation:False".

```
@app.route('/validate', methods=['POST'])
def validate():
    try:
        data = request.get_json()
        reana_file = data.get('reana_file')
        print(len(rcv.validate_reana_yaml(yaml.safe_load(reana_file))))
        if(len(rcv.validate_reana_yaml(yaml.safe_load(reana_file)))<1):
            return jsonify({'validation': True})
        else:
            return jsonify({'validation': False})
    except Exception as e:
        return jsonify({'validation': False})
```

Upload to REANA Platform

Next task was to upload the generated workflow and to try to run a job on the REANA platform.
First, a UUID is generated and indicates that the workflow is LLM generated. Since the input request is standardised, it can be split using regular expressions. Similar to the "validation" function, "reana_client" and the credentials are used to upload the configuration file incl. the Python files and the job might start on the REANA platform, which needs to be checked in the REANA dashboard with the job ID. The LLM generates a response based on the "reana_client" output.

```
@app.route('/full_upload', methods=['POST'])
def full_upload():
   try:
        random_id = "llm-gen-" + str(uuid.uuid4())
        data = request.get_json()
        files = re.split(r'```.*?\n', data.get('content'))
        files[len(files)-1] = files[len(files)-1].replace('````', '')
        files = list(filter(lambda item: item not in ['\n', ''], files))
        reana_file = yaml.safe_load(files[0])
```

```
        rcl.create_workflow(reana_file, random_id, credentails.reana_api_key)
        file_list = reana_file["inputs"]["files"]
        for i in range(1, len(files)):
            rcl.upload_file(random_id, files[i], file_list[i-1],
            ↪  credentails.reana_api_key)
        rcl.start_workflow(random_id, credentails.reana_api_key, None)
        return jsonify({'status': True, 'worflow_id': random_id})
   except Exception as e:
    return jsonify({'status': False})
```

Create a GitLab repository

The task for this tool was to create a git repository and to push the generated REANA workflow into a repository on Gitlab. The input data is processed. A git repository with the generated configuration files is created locally and then is published on AIP Gitlab. The output is the URL for the repository on Gitlab and the status of the call.

```
@app.route('/push_to_gitlab', methods=['POST'])
def push_to_gitlab():
    try:
        def initialize_local_repo(repo_path, remote_url, token):
            # Create the directory if it doesn't exist
            if not os.path.exists(repo_path):
                os.makedirs(repo_path)

            # Initialize the repository
            repo = git.Repo.init(repo_path)
            print(f"Initialized a new git repository at {repo_path}")


            data = request.get_json()
            files = re.split(r'```.*?\n', data.get('content'))
            files[len(files)-1] = files[len(files)-1].replace('```', '')
            files = list(filter(lambda item: item not in ['\n', ''], files))
            reana_file = str(files[0])
            with open(f'{repo_path}reana.yaml', "w") as f:
                    f.write(reana_file)
            repo.index.add([f'{repo_path}reana.yaml'])
            file_list = yaml.safe_load(reana_file)["inputs"]["files"]
            for i in range(1, len(files)):
```

```
            with open(f'{repo_path}{file_list[i-1]}', "w") as f:
                f.write(str(files[i]))
            repo.index.add([f'{repo_path}{file_list[i-1]}'])

        repo.index.commit("Initial commit")
        print("Created initial commit")

        origin = repo.create_remote('origin', remote_url)
        print(f"Added remote: {remote_url}")
        origin.push(refspec="master")
        print("Pushed to remote repository")

        return jsonify({'status': True})


    data = request.get_json()
    repo_name = data.get('repo_name')

    random_id = str(uuid.uuid4())


    initialize_local_repo(f'/app/{random_id}/', f'https://etlstrauss:{credentails.git
    ↪   lab_passwd}@gitlab.aip.de/etlstrauss/{random_id}.git',
    ↪   credentails.gitlab_token)
    return jsonify({'status': True, 'repo_url':
    ↪   f'https://gitlab.aip.de/etlstrauss/{random_id}'})
except Exception as e:
    return jsonify({'status': False})
```

Chapter 5

Evaluation of the Setups

The goal of this thesis implementation was to make the usage of LLMs for creation of the REANA workflows more user friendly and simpler. Several points need to be considered for the evaluation of the proposed setups. First, the quality of the responses is tested vs. direct curl requests to LLMs. Second, the time consumption is evaluated.

5.1 Response Evaluation

Figure 5.1.1 shows, how the final setup with the FlowiseAI and Open WebUI endpoint, include the "qwen2.5-coder:32b-instruct-q8_0" LLM performs in comparison to requests via curl LLM requests. The figure is available in list form in Appendix F

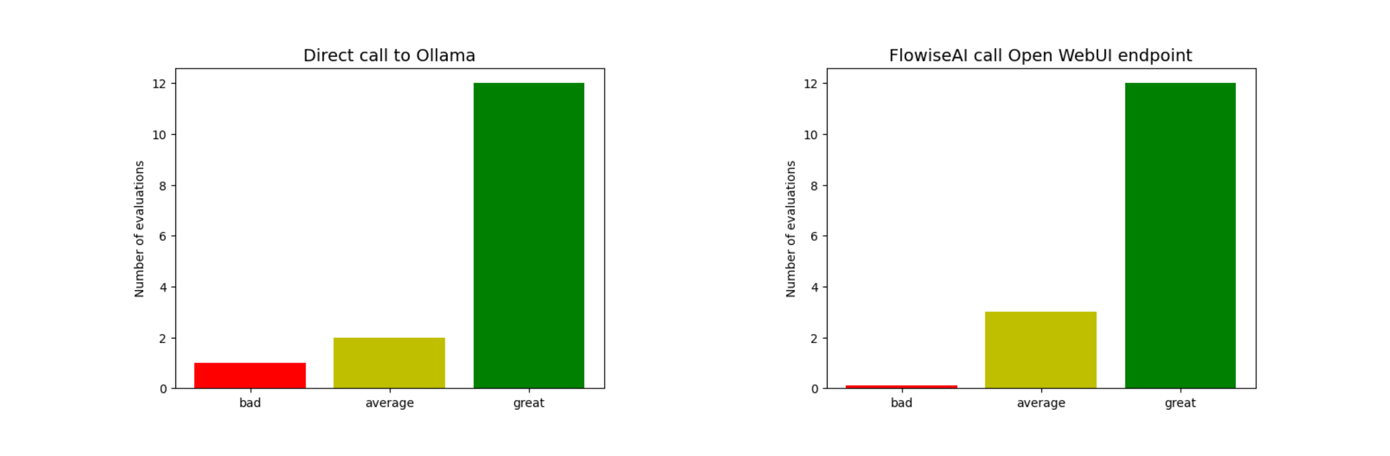

Figure 5.1.1: Model performance diagram for Ollama direct calls with the "qwen2.5-coder:32b-instruct-q8 0" LLM model vs. final prototype API calls to the final prototype.

5.2 Time Consumption

As reference for the time consumption a direct call to Ollama is used. Table 5.2.1 shows that the RAG FlowiseAI setup is faster than the reference value of 33s. The answers of the specialised workflows are shorter, please see Appendix E for references. It can be tentatively assumed that specialised systems are more time effective.
For the time evaluation a script "time-consumption.py" was used. The LLM models were already loaded by Ollama and therefore did not affect the time consumption.

	Time(in s)
Direct Ollama call	33.04
Simple RAG call	12.09
FlowiseAI call	64.03

Table 5.2.1: Time consumption comparison

Chapter 6

Conclusions

A prototype which uses LLMs with RAG and Agents was implemented and evaluated to analyse the question of this thesis "How effective are LLMs with RAG and Agents in improving data analysis pipelines in terms of effectiveness and accuracy in astrophysics?". The prototype can automatically generate REANA workflows. This proves that LLMs setups are capable to fulfil the given task. The system also achieves high accuracy for the given questions catalogue. The generated REANA workflows are available via GitLab repositories, and a job can be run on the REANA cluster.

6.0.1 Benefits and Limitations

First, the initial setup was built with a pool of agents in connection with FlowiseAI, but was discarded since this did not produced the expected results. Instead, a system with sequential agents was built.

The main advantage of this implementation is that all data and software are located on a server at the AIP. This is important for the data protection regulations. All parts of the system do not connect to external resources and can be exchanged and verified. As the logs are also available locally, the debugging and tuning process is simplified as well.

As the software are run on premise, the system administrator has the challenge to handle all the resources and configuration locally as well. The speed of the execution is of course also limited by local resources and the expertise present on premise. Given the nature of LLMs, the output might differ from time to time. The resource usage was limited to one graphic card with a GPU memory of 48GB GDDR6.

6.0.2 Next Steps

This thesis has demonstrated the feasibility of using LLMs, RAG, and agent-based systems to enhance the automation and reproducibility of data workflows. Through the development and evaluation of a prototype, it has been shown that integrating AI-driven tools can streamline the generation and validation of REANA workflows, offering a potential paradigm shift in computational research.

The results indicate that while LLM-based automation significantly improves efficiency, several challenges remain. Issues related to workflow validation, execution reliability, and integration with existing research infrastructures need to be addressed before such systems can be deployed at scale. Additionally, as LLMs and associated technologies continue to evolve, further research is required to optimize performance, improve interpretability, and ensure data integrity.

Future work should explore the potential of distributed AI architectures, enhanced security mechanisms, and adaptive learning models to further refine the automation process. By addressing these challenges, the integration of LLMs in scientific workflows could pave the way for a new era of intelligent, reproducible, and efficient research methodologies.

Chapter 7

Acknowledgments

First of all, I thank the Astrophysical Institute Potsdam (AIP) and the University of Applied Science "Hochschule Stralusnd", represented respectively by Dr. rer. nat. Arman Khalatyan and Prof. Dr. rer. nat. André Grüning for the possibility to accomplish this bachelor thesis project in such a large scope. I would also like to thank Anastasia Galkin and Guido Kuhlmann for the editing help.

Bibliography

[AKG24] Dr. Arman Khalatyan, *The problem of creating REANA workflows.*

[AKP24] Dr. Arman Khalatyan, *Solution approach for generating REANA wokrlfows.*

[CDS24] *crewAI.* [Online]. Available: https://docs.crewai.com/ (visited on 09/24/2024).

[CGS24] *Installing crewAI - crewAI.* [Online]. Available: https://docs.crewai.com/getting-started/Installing-CrewAI/ (visited on 09/24/2024).

[CIBM24] *What is crewAI? — IBM.* [Online]. Available: https://www.ibm.com/think/topics/crew-ai (visited on 09/24/2024).

[EgaAtEl24] K. Egashira, M. Vero, R. Staab, J. He, and M. Vechev, *Exploiting LLM Quantization*, arXiv:2405.18137 [cs], Nov. 2024. DOI: 10.48550/arXiv.2405.18137. [Online]. Available: http://arxiv.org/abs/2405.18137 (visited on 01/02/2025).

[FGS24] *Get Started — FlowiseAI.* [Online]. Available: https://docs.flowiseai.com/getting-started (visited on 09/20/2024).

[FI24] *Introduction — FlowiseAI.* [Online]. Available: https://docs.flowiseai.com/ (visited on 09/24/2024).

[FILC24] *LangChain — FlowiseAI.* [Online]. Available: https://docs.flowiseai.com/integrations/langchain (visited on 09/24/2024).

[FILI24] *LlamaIndex — FlowiseAI.* [Online]. Available: https://docs.flowiseai.com/integrations/llamaindex (visited on 09/24/2024).

[FIU24] *Utilities — FlowiseAI.* [Online]. Available: https://docs.flowiseai.com/integrations/utilities (visited on 09/24/2024).

[FLMA24] *Multi-Agents — FlowiseAI.* [Online]. Available: https://docs.flowiseai.com/using-flowise/agentflows/multi-agents (visited on 01/02/2025).

[FLSA24] *Sequential Agents — FlowiseAI.* [Online]. Available: https://docs.flowiseai.com/using-flowise/agentflows/sequential-agents (visited on 01/02/2025).

[GaoEtAl24] Y. Gao *et al.*, *Retrieval-Augmented Generation for Large Language Models: A Survey*, arXiv:2312.10997 [cs], Mar. 2024. [Online]. Available: http://arxiv.org/abs/2312.10997 (visited on 11/18/2024).

[GHR24] *Home · reanahub/reana Wiki*. [Online]. Available: https://github.com/reanahub/reana/wiki/ (visited on 01/14/2025).

[HF25] *Hugging Face – The AI community building the future.* [Online]. Available: https://huggingface.co/ (visited on 02/03/2025).

[HFGG24] *Introduction to ggml*. [Online]. Available: https://huggingface.co/blog/introduction-to-ggml (visited on 01/28/2025).

[HQw25] *Qwen (Qwen)*. [Online]. Available: https://huggingface.co/Qwen (visited on 02/03/2025).

[IGGUF24] *GGUF versus GGML — IBM*. [Online]. Available: https://www.ibm.com/think/topics/gguf-versus-ggml (visited on 01/02/2025).

[IHal25] *What Are AI Hallucinations? — IBM*. [Online]. Available: https://www.ibm.com/think/topics/ai-hallucinations (visited on 02/03/2025).

[ITem25] *What is LLM Temperature? — IBM*. [Online]. Available: https://www.ibm.com/think/topics/llm-temperature (visited on 02/03/2025).

[IWAA24] *What Are AI Agents? — IBM*. [Online]. Available: https://www.ibm.com/think/topics/ai-agents (visited on 01/02/2025).

[L1] *Introduction — LangChain*. [Online]. Available: https://python.langchain.com/v0.2/docs/introduction/ (visited on 01/21/2025).

[L2] *Ollama*. [Online]. Available: https://ollama.com/search (visited on 01/21/2025).

[L3] *Https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/quadro rtx-8000-us-nvidia-946977-r1-web.pdf*. [Online]. Available: https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/quadro-product-literature/quadro-rtx-8000-us-nvidia-946977-r1-web.pdf (visited on 11/05/2024).

[LCI24] *Introduction — LangChain*. [Online]. Available: https://python.langchain.com/v0.2/docs/introduction/ (visited on 09/24/2024).

[LCQ24] *Installation — LangChain*. [Online]. Available: https://python.langchain.com/v0.1/docs/get_started/installation/ (visited on 09/24/2024).

[LFGS24] *Installation — Langflow Documentation*. [Online]. Available: https://docs.langflow.org/getting-started-installation (visited on 09/20/2024).

[LFW24] *Welcome to Langflow — Langflow Documentation*. [Online]. Available: https://docs.langflow.org/ (visited on 01/03/2025).

[LII2024] *LlamaIndex - LlamaIndex*. [Online]. Available: https://docs.llamaindex.ai/en/stable/#introduction (visited on 09/24/2024).

[LinAtEl24] J. Lin *et al.*, “AWQ: Activation-aware Weight Quantization for On-Device LLM Compression and Acceleration,” en,

[LvZFl24] L. van Zyl, *Flowise AI (2024) Tutorial - YouTube*. [Online]. Available: https://www.youtube.com/playlist?list=PL4HikwTaYE0H7wBxhvQqxYcKOkZ4O3zXh (visited on 01/06/2025).

[MEOL24] *Ollama: A Deep Dive into Running Large Language Models Locally(PART-1): — by Anoop Maurya — Medium*. [Online]. Available: https://medium.com/@mauryaanoop3/ollama-a-deep-dive-into-running-large-language-models-locally-part-1-0a4b70b30982 (visited on 11/18/2024).

[OGD24] *Is it possible to have open WebUI as a chat-interface for Flowise? · FlowiseAI/Flowise · Discussion #2581*. [Online]. Available: https://github.com/FlowiseAI/Flowise/discussions/2581#discussioncomment-10607580 (visited on 01/06/2025).

[OGE24] *Pipelines/examples/pipelines/integrations/python_code_pipeline.py at main · open-webui/pipelines*. [Online]. Available: https://github.com/open-webui/pipelines/blob/main/examples/pipelines/integrations/python_code_pipeline.py (visited on 01/06/2025).

[OGS24] *Getting Started — Open WebUI*. [Online]. Available: https://docs.openwebui.com/getting-started/ (visited on 11/04/2024).

[OH24] *Home — Open WebUI*. [Online]. Available: https://docs.openwebui.com/ (visited on 11/04/2024).

[Olla25] *Ollama*. [Online]. Available: https://ollama.com/ (visited on 02/03/2025).

[OP24] *Pipelines — Open WebUI*. [Online]. Available: https://docs.openwebui.com/pipelines/ (visited on 10/30/2024).

[OQC24] *Qwen2.5-coder:32b-instruct-q8_0/model*. [Online]. Available: https://ollama.com/library/qwen2.5-coder:32b-instruct-q8_0/blobs/50cf95c4a2f0 (visited on 01/02/2025).

[OQC25] *Qwen2.5-coder:32b-base-q8_0*. [Online]. Available: https://ollama.com/library/qwen2.5-coder:32b-base-q8_0 (visited on 02/03/2025).

[OwuGh24] *Open-webui/pipelines: Pipelines: Versatile, UI-Agnostic OpenAI-Compatible Plugin Framework*. [Online]. Available: https://github.com/open-webui/pipelines (visited on 01/06/2025).

[P4NR24] *Punch_public / reana / Tutorials · GitLab*. [Online]. Available: https://gitlab-p4n.aip.de/punch_public/reana/tutorials (visited on 01/03/2025).

[QLQ24] *Local Quickstart - Qdrant*. [Online]. Available: https://qdrant.tech/documentation/quickstart/ (visited on 10/30/2024).

[QWiQ24] *What is Qdrant? - Qdrant*. [Online]. Available: https://qdrant.tech/documentation/overview/ (visited on 11/06/2024).

[WALMcs24] *What are large language models (LLMs)? — Microsoft Azure*. [Online]. Available: https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-are-large-language-models-llms (visited on 01/01/2025).

Appendix A

Model evaluation configurations / question catalogue

A.1 Question Catalogue

id	question	kind
1	Please change the following workflow, so that the output is 'Hello, REANA!' "' # Hello World Workflow inputs: files: - helloworld.py workflow: type: serial specification: steps: - environment: 'docker.io/library/python:3.10-bookworm' kubernetes_memory_limit: '100Mi' kubernetes_job_timeout: 60 # seconds commands: - python helloworld.py "' "' # helloworld.py print('Hello, World!') "'	programming
2	What is REANA	general
3	What is REANA developed at Cern	general
4	Give me a tutorial how to create, start, run and check and reana workflow	general
5	Give me a tutorial how to create, start, run and check and reana workflow with the cli	general
6	Please list and explain all parameters which can be used in the reana yaml specifications	general
7	Write an simple hello world reana yaml workflow	programming-rag
8	Write an simple hello world reana workflow, which creates different files. Please save this files in subfolder. Therefore you can use the mkdir command.	programming-rag

9	Write an reana workflow. It should take a csv file as input and plot the data from the csv. The csv itself contains to parameters. X and Y. Draw the points an return an png.	programming-rag
10	Please use reana to plot an sinus function. Do this with two scripts. One for data generation and another one for plotting. Save the result as png.	programming-rag
11	Create an reana workflow which accesses remote data. Use this url for that `'https://s3.data.aip.de:9000/sh21pdf/gaiaedr3_sh_input_healpixlevel5_hpno-00000{str(n)}.fits.hdf5.txt`. Notice that str(n) should be from 1 to 5	programming-rag
12	Please create an reana workflow which curls an external webpage and prints it with 'rich formatting to the cli. The url of the webpage should be set by an environmental variable'	programming
13	Please create an rena workflow which querys some data from the AIP gaia ddr3 release and plots them. Save the figure as png	programming
14	Create an reana workflow which sets different parameters and print them to the cli	programming
15	Create an hello world reana workflow, which prints hello world to the cli. Don't use python! Write it in bash and directly write the 'code' into the workflow	programming

Table A.1.1: Question catalogue used for the model evaluation

A.2 Configurations

```
[Models]
list=['qwen2.5-coder:32b-instruct-q8_0', 'mistral-small:22b-instruct-2409-fp16',
↪  'llama3.1:70b', 'llama3.3:70b-instruct-q4_K_M', 'hf.co/matteogeniaccio/phi-4:F16']
[Questions]
list= [
    ["Please change the following workflow, so that the output is 'Hello, REANA!' \n```\n#
    ↪  Hello World Workflow\n    inputs:\n    files:\n         - helloworld.py\n
    ↪  workflow:\n    type: serial\n    specification:\n        steps:\n        -
    ↪  environment: 'docker.io/library/python:3.10-bookworm'\n
    ↪  kubernetes_memory_limit: '100Mi'\n            kubernetes_job_timeout: 60 #
    ↪  seconds\n            commands:\n            - python helloworld.py\n```\n```\n
    ↪  # helloworld.py\n    print('Hello, World!')\n```", "programming"],
    ["What is REANA", "general"],
    ["What is REANA developed at Cern", "general"],
```

```
    ["Give me a tutorial how to create, start, run and check and reana workflow",
    ↪  "general"],
    ["Give me a tutorial how to create, start, run and check and reana workflow with the
    ↪  cli", "general"],
    ["Please list and explain all parameters which can be used in the reana yaml
    ↪  specifications", "general"],
    ["Write an simple hello world reana yaml workflow", "programming-rag"],
    ["Write an simple hello world reana workflow, which creates different files. Please
    ↪  save this files in subfolder. Therefore you can use the mkdir command.",
    ↪  "programming-rag"],
    ["Write an reana workflow. It should take a csv file as input and plot the data from
    ↪  the csv. The csv itself contains to parameters. X and Y. Draw the points an
    ↪  return an png.", "programming-rag"],
    ["Please use reana to plot an sinus function. Do this with two scripts. One for data
    ↪  generation and another one for plotting. Save the result as png.",
    ↪  "programming-rag"],
    ["Create an reana workflow which accesses remote data. Use this url for that
    ↪  'https://s3.data.aip.de:9000/sh21pdf/gaiaedr3_sh_input_healpixlevel5_hpno-00000{s
    ↪  tr(n)}.fits.hdf5.txt'. Notice that str(n) should be from 1 to 5",
    ↪  "programming-rag"],
    ["Please create an reana workflow which curls an external webpage and prints it with
    ↪  'rich formatting to the cli. The url of the webpage should be set by an
    ↪  environmental variable'", "programming"],
    ["Please create an rena workflow which querys some data from the AIP gaia ddr3
    ↪  release and plots them. Save the figure as png", "programming"],
    ["Create an reana workflow which sets different parameters and print them to the
    ↪  cli", "programming"],
    ["Create an hello world reana workflow, which prints hello world to the cli. Don't
    ↪  use python! Write it in bash and directly write the 'code' into the workflow",
    ↪  "programming"]
]
class = ["general", "programming", "programming-rag"]
```

Figure A.2.1: Catalogue of questions and configurations for model evaluation

Appendix B

Model evaluation

B.1 Scripts for Model Evaluation Without RAG

```
import requests
import psycopg2
import toml

file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()

config = toml.load('model_evaluation.toml')

con = psycopg2.connect(
    dbname="model_evaluation",
    user="local",
    password=passwd,
    host="141.33.165.24"
)

cur = con.cursor()
cur.execute("CREATE TABLE IF NOT EXISTS model_evaluation (id SERIAL PRIMARY KEY, response
↪  VARCHAR, question VARCHAR, model VARCHAR, review INTEGER, class VARCHAR);")
con.commit()

for j in config['Models']['list']:
    for i in config['Questions']['list']:
        print(f'Model: {j} Question: {i[0]}')
```

```python
        url = 'http://141.33.165.24:11434/api/generate'
        data = {
        "model": j,
        "stream" : False,
        "prompt": f"{i[0]}",
        "options": {
            "temperature": 0.3
        },
        }


        response = requests.post(url, json=data)

        print(response)

        cur.execute("INSERT INTO model_evaluation (response, question, model, review,
        ↪   class) VALUES (%s, %s, %s, NULL, %s)", (response.json()['response'], i[0], j,
        ↪   i[1]))
        con.commit()

con.close()
```

Figure B.1.1: Script to make request directly to Ollama

```python
import psycopg2
from rich.console import Console
from rich.markdown import Markdown

file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()

console = Console()

con = psycopg2.connect(
    dbname="model_evaluation",
    user="local",
    password=passwd,
    host="141.33.165.24"
)

cur = con.cursor()
```

```python
cur.execute("SELECT COUNT(*) FROM model_evaluation;")
number_of_records = cur.fetchone()[0]

is_true = True
console.clear()
while is_true:
    console.clear()
    console.print("""
                  This is a small command line tool to review the generated responses.
                  ↪  Please answer the following questions with y or n. y for true and n
                  ↪  for false. Did you understand?
                  In the following you will see a question and a the response.
                  """)
    user_input = input(":")
    if user_input in ["true", "1", "yes", "y", "t"]:
            is_true = False
    elif user_input in ["false", "0", "no", "n", "f"]:
            is_true = True

for i in range(number_of_records):
    console.clear()
    console.print(Markdown(f"### Question:\n"))
    cur.execute(f"SELECT question FROM model_evaluation WHERE id={i+1};")
    console.print(Markdown(cur.fetchone()[0]))

    console.print("\n" * 4)

    console.print(Markdown(f"### Response: \n"))
    cur.execute(f"SELECT response FROM model_evaluation WHERE id={i+1};")
    console.print(Markdown(cur.fetchone()[0]))

    user_input = int(input(":").strip().lower())

    cur.execute(f"UPDATE model_evaluation SET review={user_input} WHERE id={i+1};")
    con.commit()
    console.clear()

con.close()
```

Figure B.1.2: Script to evaluate direct Ollama requests

```python
import pandas as pd
import psycopg2

file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()

data = [['id', 'question', 'model', 'review']]


con = psycopg2.connect(
    dbname="model_evaluation",
    user="local",
    password=passwd,
    host="141.33.165.24"
)

cur = con.cursor()
cur.execute("SELECT COUNT(*) FROM model_evaluation;")

for i in range(0, cur.fetchone()[0]):
    cur.execute("SELECT * FROM model_evaluation WHERE id=%s;", (i+1,))
    current = cur.fetchone()
    datatmp = [current[0], current[2], current[3], current[4]]
    data.append(datatmp)

df = pd.DataFrame(data[1:], columns=data[0])

file = open("model_evaluation_table.md", "w")
file.write(df.to_markdown(index=False))
file.close()
```

Figure B.1.3: Script to generate evaluation markdown table of direct Ollama calls

B.2 Analyses

[1]

[1]notice that the original file type was an jupyter notebook; the notebook was formatted to an python file to format and show it correctly in this appendix

```python
# %% [markdown]
# # Ollama model evaluation analyses
# - this notebook shows some analyses for the ollama evaluations

# %%
# import libraries
import toml
import psycopg2
import matplotlib.pyplot as plt
import numpy as np

# %%
# load configurations
config = toml.load('model_evaluation.toml')
file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()
file.close()

# %%
# Connect to the database
con = psycopg2.connect(
    dbname="model_evaluation",
    user="local",
    password=passwd,
    host="141.33.165.24"
)
cur = con.cursor()

# %%
# get the number of evaluations for each class
review_list = []
for i in range(0,3):
    cur.execute(f"SELECT COUNT(*) FROM model_evaluation WHERE review={i};")
    review_list.append(cur.fetchone()[0])
labels = ['bad', 'average', 'great']

# create the plot
plt.bar(labels, review_list, color=['r', 'y', 'g'])
```

```
plt.title('Model evaluation by review')
plt.ylabel('Number of evaluations')
plt.figtext(0.5, 0.02, "without RAG", wrap=True, horizontalalignment='center', fontsize=8)
#plt.show()
plt.savefig('model_evaluation_by_review.png')

# %%
# create empty and standard lists
review_list = [[],[],[]]
colors_list = ['r', 'y', 'g']
labels_list = ['bad', 'average', 'great']

# run sql quries to create lists for the bar chart (bad/average/great per model)
for i in range(0,3):
        tmp_list = []
        for model in config['Models']['list']:
                cur.execute(f"SELECT COUNT(*) FROM model_evaluation WHERE model='{model}'
                ↪  AND review={i};")
                eval = cur.fetchone()[0]
                if(eval == 0):
                        eval = 0.1
                tmp_list.append(eval)
        review_list[i] = tmp_list

# set the width of the bar and create subplots
barWidth = 0.275
fig, ax = plt.subplots(figsize =(9, 10))

# plot the bars by grouping them by review
br = np.arange(len(config["Models"]["list"]))
for i in range(0, len(review_list)):
        plt.barh(br, review_list[i], color =colors_list[i], edgecolor ='grey',
        ↪  label=labels_list[i], height=barWidth)
        br = [x + barWidth for x in br]

# add labels and title
plt.xlabel('Number of answers', fontsize = 12)
plt.ylabel('Model', fontsize = 12)
```

```
plt.yticks([r + barWidth for r in range(len(config["Models"]["list"]))],
↪  config["Models"]["list"], fontsize = 8)
plt.title('Model evaluation by model')

ax.annotate('without RAG',xy = (0.87, 0.007),xycoords='axes
↪  fraction',ha='left',va="center",fontsize=10)

# add legend + show
plt.legend()
plt.show()

# %%
# create empty and standard lists
review_list0 = [[],[],[]]
review_list1 = [[],[],[]]
review_list2 = [[],[],[]]
colors_list = ['r', 'y', 'g']
colors_list2 = ['lightsalmon', 'khaki', 'limegreen']
colors_list3 = ['tomato', 'lemonchiffon', 'lime']
labels_list = ['bad c=0', 'average c=0', 'great c=0']
labels_list2 = ['bad c=1', 'average c=1', 'great c=1']
labels_list3 = ['bad c=2', 'average c=2', 'great c=2']

# run sql quries to create lists for the bar chart (bad/average/great per model)
for i in range(0, 3):
        tmp_list = []
        for model in config['Models']['list']:
                cur.execute(f"SELECT COUNT(*) FROM model_evaluation WHERE model='{model}'
                ↪  AND review={i} AND class='general';")
                eval = cur.fetchone()[0]
                if(eval == 0):
                        eval = 0.1
                tmp_list.append(eval)
        review_list0[i] = tmp_list
        tmp_list = []
        for model in config['Models']['list']:
                cur.execute(f"SELECT COUNT(*) FROM model_evaluation WHERE model='{model}'
                ↪  AND review={i} AND class='programming';")
                eval = cur.fetchone()[0]
```

```
                if(eval == 0):
                        eval = 0.1
                tmp_list.append(eval)
        review_list1[i] = tmp_list
        tmp_list = []
        for model in config['Models']['list']:
                cur.execute(f"SELECT COUNT(*) FROM model_evaluation WHERE model='{model}'
                ↪  AND review={i} AND class='programming-rag';")
                eval = cur.fetchone()[0]
                if(eval == 0):
                        eval = 0.1
                tmp_list.append(eval)
        review_list2[i] = tmp_list

# set the width of the bar and create subplots
barWidth = 0.275
fig, ax = plt.subplots(figsize =(9, 10))

# plot the bars by grouping them by review
br = np.arange(len(config["Models"]["list"]))
for i in range(0, len(review_list0)):
        plt.barh(br, review_list0[i], color =colors_list[i], edgecolor ='grey',
        ↪   label=labels_list[i], height=barWidth)
        plt.barh(br, review_list1[i], color =colors_list2[i], edgecolor ='grey',
        ↪   label=labels_list2[i], height=barWidth, left=review_list0[i])
        plt.barh(br, review_list2[i], color =colors_list3[i], edgecolor ='grey',
        ↪   label=labels_list3[i], height=barWidth,  left=[sum(x) for x in
        ↪   zip(review_list0[i], review_list1[i])])
        br = [x + barWidth for x in br]

# add labels and title
plt.xlabel('Number of answers', fontsize = 12)
plt.ylabel('Model', fontsize = 12)
plt.yticks([r + barWidth for r in range(len(config["Models"]["list"]))],
↪  config["Models"]["list"], fontsize = 8)
plt.title('Model evaluation by model')
```

```
ax.annotate('Class 0 are general questions / tasks\nClass 1 are questions / tasks to
↪  create yaml files and write code\n Class 2 are questions / tasks to create yaml files
↪  and write code with provided information (RAG)',xy = (1.0, -0.1),xycoords='axes
↪  fraction',ha='right',va="center",fontsize=6)
ax.annotate('Temperature is 0.3\n Notice that in that point no context is given(no
↪  RAG)',xy = (0, -0.1),xycoords='axes fraction',ha='left',va="center",fontsize=6)
ax.annotate('without RAG',xy = (0.87, 0.007),xycoords='axes
↪  fraction',ha='left',va="center",fontsize=10)


# add legend + show
plt.legend()
plt.show()

# %%
cur.close()
```

Figure B.2.1: Script for analysation of direct Ollama calls

B.3 Evaluation Table

[2]

id	model	question	review
1	opencoder:8b-instruct-fp16	1	2
2	opencoder:8b-instruct-fp16	2	2
3	opencoder:8b-instruct-fp16	3	2
4	opencoder:8b-instruct-fp16	4	1
5	opencoder:8b-instruct-fp16	5	0
6	opencoder:8b-instruct-fp16	6	1
7	opencoder:8b-instruct-fp16	7	1
8	opencoder:8b-instruct-fp16	8	1
9	opencoder:8b-instruct-fp16	9	1
10	opencoder:8b-instruct-fp16	10	0
11	opencoder:8b-instruct-fp16	11	0
12	opencoder:8b-instruct-fp16	12	0

[2] The questions are replaced by the ID given in the questions overview. The responses are not shown, due to formatting reasons. They are accessible in the GitHub repository.

13	opencoder:8b-instruct-fp16	13	0
14	opencoder:8b-instruct-fp16	14	1
15	opencoder:8b-instruct-fp16	15	1
16	qwen2.5-coder:32b-instruct-q8_0	1	2
17	qwen2.5-coder:32b-instruct-q8_0	2	2
18	qwen2.5-coder:32b-instruct-q8_0	3	2
19	qwen2.5-coder:32b-instruct-q8_0	4	2
20	qwen2.5-coder:32b-instruct-q8_0	5	2
21	qwen2.5-coder:32b-instruct-q8_0	6	1
22	qwen2.5-coder:32b-instruct-q8_0	7	2
23	qwen2.5-coder:32b-instruct-q8_0	8	2
24	qwen2.5-coder:32b-instruct-q8_0	9	2
25	qwen2.5-coder:32b-instruct-q8_0	10	2
26	qwen2.5-coder:32b-instruct-q8_0	11	2
27	qwen2.5-coder:32b-instruct-q8_0	12	2
28	qwen2.5-coder:32b-instruct-q8_0	13	0
29	qwen2.5-coder:32b-instruct-q8_0	14	2
30	qwen2.5-coder:32b-instruct-q8_0	15	1
31	mistral-small:22b-instruct-2409-fp16	1	2
32	mistral-small:22b-instruct-2409-fp16	2	2
33	mistral-small:22b-instruct-2409-fp16	3	2
34	mistral-small:22b-instruct-2409-fp16	4	0
35	mistral-small:22b-instruct-2409-fp16	5	0
36	mistral-small:22b-instruct-2409-fp16	6	1
37	mistral-small:22b-instruct-2409-fp16	7	0
38	mistral-small:22b-instruct-2409-fp16	8	0
39	mistral-small:22b-instruct-2409-fp16	9	1
40	mistral-small:22b-instruct-2409-fp16	10	0
41	mistral-small:22b-instruct-2409-fp16	11	1
42	mistral-small:22b-instruct-2409-fp16	12	0
43	mistral-small:22b-instruct-2409-fp16	13	0
44	mistral-small:22b-instruct-2409-fp16	14	0
45	mistral-small:22b-instruct-2409-fp16	15	0
46	llama3.1:70b	1	2
47	llama3.1:70b	2	0
48	llama3.1:70b	3	1
49	llama3.1:70b	4	0
50	llama3.1:70b	5	0
51	llama3.1:70b	6	2

52	llama3.1:70b	7	0
53	llama3.1:70b	8	0
54	llama3.1:70b	9	0
55	llama3.1:70b	10	0
56	llama3.1:70b	11	0
57	llama3.1:70b	12	0
58	llama3.1:70b	13	0
59	llama3.1:70b	14	0
60	llama3.1:70b	15	0
61	llama3.3:70b-instruct-q4_K_M	1	2
62	llama3.3:70b-instruct-q4_K_M	2	2
63	llama3.3:70b-instruct-q4_K_M	3	2
64	llama3.3:70b-instruct-q4_K_M	4	0
65	llama3.3:70b-instruct-q4_K_M	5	0
66	llama3.3:70b-instruct-q4_K_M	6	0
67	llama3.3:70b-instruct-q4_K_M	7	1
68	llama3.3:70b-instruct-q4_K_M	8	1
69	llama3.3:70b-instruct-q4_K_M	9	1
70	llama3.3:70b-instruct-q4_K_M	10	1
71	llama3.3:70b-instruct-q4_K_M	11	1
72	llama3.3:70b-instruct-q4_K_M	12	1
73	llama3.3:70b-instruct-q4_K_M	13	0
74	llama3.3:70b-instruct-q4_K_M	14	2
75	llama3.3:70b-instruct-q4_K_M	15	1
76	hf.co/matteogeniaccio/phi-4:F16	1	2
77	hf.co/matteogeniaccio/phi-4:F16	2	2
78	hf.co/matteogeniaccio/phi-4:F16	3	2
79	hf.co/matteogeniaccio/phi-4:F16	4	0
80	hf.co/matteogeniaccio/phi-4:F16	5	0
81	hf.co/matteogeniaccio/phi-4:F16	6	1
82	hf.co/matteogeniaccio/phi-4:F16	7	1
83	hf.co/matteogeniaccio/phi-4:F16	8	1
84	hf.co/matteogeniaccio/phi-4:F16	9	0
85	hf.co/matteogeniaccio/phi-4:F16	10	1
86	hf.co/matteogeniaccio/phi-4:F16	11	0
87	hf.co/matteogeniaccio/phi-4:F16	12	0
88	hf.co/matteogeniaccio/phi-4:F16	13	0
89	hf.co/matteogeniaccio/phi-4:F16	14	0
90	hf.co/matteogeniaccio/phi-4:F16	15	1

Table B.3.1: Table of Rating of Direct Ollama Calls

B.4 Scripts for Model Evaluation With RAG

```
import requests
from langchain_community.document_loaders import GitLoader
import io

# pushes vecotrized data to FlowiseAI conencted to Qdrant
def pushToFlowise(file_like_object, file_name):
    API_URL = "http://141.33.165.24:8000/api/v1/vector/upsert/ac04a85d-3a13-409c-b4eb-dc
    ↪ d40b9ef742"
    # use form data to upload files
    form_data = {
        "files": (file_name, file_like_object)
    }
    body_data = {
    }

    def query(form_data, body_data):
        response = requests.post(API_URL, files=form_data, data=body_data)
        return response.json()

    query(form_data, body_data)


# load yaml files from git repo
loader =
↪ GitLoader(repo_path="/home/tom/Documents/HOST/Bachelorarbeit/reana-tutorials-github")
data = loader.load()
for i in range(len(data)):
    file_data = data[i].metadata.get('source').split('/')
    if('.yaml' in file_data[len(file_data)-1]):
        file_like_object = io.StringIO(data[i].page_content)
        file_name = ""
        for j in range(len(file_data)-1):
            file_name += file_data[j] + '-'
        pushToFlowise(file_like_object, file_name)
```

```
        print("Pushed: " + data[i].metadata.get('source'))
```

Figure B.4.1: Script to vectorize and upload yaml files to the vector store

```
import requests
import psycopg2
import toml

file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()

config = toml.load('model_evaluation.toml')

con = psycopg2.connect(
    dbname="model_evaluation",
    user="local",
    password=passwd,
    host="141.33.165.24"
)


API_URL =
↪   "http://141.33.165.24:8000/api/v1/prediction/6b74c5bd-bcf9-4a29-824f-06d0028cce74"

def query(payload):
    response = requests.post(API_URL, json=payload)
    return response.json()

cur = con.cursor()
cur.execute("CREATE TABLE IF NOT EXISTS model_evaluation_rag (id SERIAL PRIMARY KEY,
↪   response VARCHAR, question VARCHAR, model VARCHAR, review INTEGER, class VARCHAR);")
con.commit()

for j in config['Models']['list']:
    for i in config['Questions']['list']:
        print(f'Model: {j} Question: {i[0]}')
        output = query({
            "question": i[0],
            "overrideConfig": {
                "modelName": j,
            }
```

```
        })

        response = query(output).get('text')
        print(response)
        print("Starting upload to db")
        cur.execute("INSERT INTO model_evaluation_rag (response, question, model, review,
        ↪  class) VALUES (%s, %s, %s, NULL, %s)", (response, i[0], j, i[1]))
        con.commit()

con.close()
```

Figure B.4.2: Script to make request to FlowiseAI RAG implementation

```
import psycopg2
from rich.console import Console
from rich.markdown import Markdown

file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()

console = Console()

con = psycopg2.connect(
    dbname="model_evaluation",
    user="local",
    password=passwd,
    host="141.33.165.24"
)

cur = con.cursor()
cur.execute("SELECT COUNT(*) FROM model_evaluation_rag;")
number_of_records = cur.fetchone()[0]

is_true = True
console.clear()
while is_true:
    console.clear()
    console.print("""
                  This is a small command line tool to review the generated responses.
                  ↪  Please answer the following questions with y or n. y for true and n
                  ↪  for false. Did you understand?
```

```
                   In the following you will see a question and a the response.
                   """)
    user_input = input(":")
    if user_input in ["true", "1", "yes", "y", "t"]:
            is_true = False
    elif user_input in ["false", "0", "no", "n", "f"]:
            is_true = True

for i in range(number_of_records):
    console.clear()
    console.print(Markdown(f"### Question:\n"))
    cur.execute(f"SELECT question FROM model_evaluation_rag WHERE id={i+1};")
    console.print(Markdown(cur.fetchone()[0]))

    console.print("\n" * 4)

    console.print(Markdown(f"### Response: \n"))
    cur.execute(f"SELECT response FROM model_evaluation_rag WHERE id={i+1};")
    console.print(Markdown(cur.fetchone()[0]))

    user_input = int(input(":").strip().lower())

    cur.execute(f"UPDATE model_evaluation_rag SET review={user_input} WHERE id={i+1};")
    con.commit()
    console.clear()

con.close()
```

Figure B.4.3: Script to evaluate requests to FlowiseAI RAG implementation

```
import pandas as pd
import psycopg2

file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()

data = [['id', 'question', 'model', 'review']]


con = psycopg2.connect(
    dbname="model_evaluation",
```

```
    user="local",
    password=passwd,
    host="141.33.165.24"
)

cur = con.cursor()
cur.execute("SELECT COUNT(*) FROM model_evaluation_rag;")

for i in range(0, cur.fetchone()[0]):
    cur.execute("SELECT * FROM model_evaluation_rag WHERE id=%s;", (i+1,))
    current = cur.fetchone()
    datatmp = [current[0], current[2], current[3], current[4]]
    data.append(datatmp)

df = pd.DataFrame(data[1:], columns=data[0])

file = open("model_evaluation_table_rag.md", "w")
file.write(df.to_markdown(index=False))
file.close()
```

Figure B.4.4: Script to generate evaluation markdown table of FlowiseAI RAG implementation calls

B.5 Analyses with RAG

[3]

```
# %% [markdown]
# # Ollama model evaluation analyses
# - this notebook shows some analyses for the ollama evaluations

# %%
# import libraries
import toml
import psycopg2
import matplotlib.pyplot as plt
import numpy as np

# %%
```

[3]notice that the original file type was an jupyter notebook; the notebook was formatted to an python file to format and show it correctly in this appendix

```
# load configurations
config = toml.load('model_evaluation.toml')
file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()
file.close()

# %%
# Connect to the database
con = psycopg2.connect(
    dbname="model_evaluation",
    user="local",
    password=passwd,
    host="141.33.165.24"
)
cur = con.cursor()

# %%
# get the number of evaluations for each class
review_list = []
for i in range(0,3):
    cur.execute(f"SELECT COUNT(*) FROM model_evaluation_rag WHERE review={i};")
    review_list.append(cur.fetchone()[0])
labels = ['bad', 'average', 'great']

# create the plot
plt.bar(labels, review_list, color=['r', 'y', 'g'])
plt.title('Model evaluation by review')
plt.ylabel('Number of evaluations')
plt.figtext(0.5, 0.02, "with RAG", wrap=True, horizontalalignment='center', fontsize=8)
#plt.show()
plt.savefig('model_evaluation_rag.png',format='png', bbox_inches='tight')

# %%
# create empty and standard lists
review_list = [[],[],[]]
colors_list = ['r', 'y', 'g']
labels_list = ['bad', 'average', 'great']

# run sql quries to create lists for the bar chart (bad/average/great per model)
```

```python
for i in range(0,3):
        tmp_list = []
        for model in config['Models']['list']:
                cur.execute(f"SELECT COUNT(*) FROM model_evaluation_rag WHERE
                ↪  model='{model}' AND review={i};")
                eval = cur.fetchone()[0]
                if(eval == 0):
                        eval = 0.1
                tmp_list.append(eval)
        review_list[i] = tmp_list

# set the width of the bar and create subplots
barWidth = 0.275
fig, ax = plt.subplots(figsize =(8, 10))

# plot the bars by grouping them by review
br = np.arange(len(config["Models"]["list"]))
for i in range(0, len(review_list)):
        plt.barh(br, review_list[i], color =colors_list[i], edgecolor ='grey',
        ↪  label=labels_list[i], height=barWidth)
        br = [x + barWidth for x in br]

# add labels and title
plt.xlabel('Number of answers', fontsize = 12)
plt.ylabel('Model', fontsize = 12)
plt.yticks([r + barWidth for r in range(len(config["Models"]["list"]))],
↪  config["Models"]["list"], fontsize = 8)
plt.title('Model evaluation by model')

ax.annotate('with RAG',xy = (0.89, 0.007),xycoords='axes
↪  fraction',ha='left',va="center",fontsize=10)

# add legend + show
plt.legend()
#plt.show()
plt.savefig('model_evaluation_by_model.png',format='png', bbox_inches='tight')

# %%
# create empty and standard lists
```

```
review_list0 = [[],[],[]]
review_list1 = [[],[],[]]
review_list2 = [[],[],[]]
colors_list = ['r', 'y', 'g']
colors_list2 = ['lightsalmon', 'khaki', 'limegreen']
colors_list3 = ['tomato', 'lemonchiffon', 'lime']
labels_list = ['bad c=0', 'average c=0', 'great c=0']
labels_list2 = ['bad c=1', 'average c=1', 'great c=1']
labels_list3 = ['bad c=2', 'average c=2', 'great c=2']

# run sql quries to create lists for the bar chart (bad/average/great per model)
for i in range(0, 3):
	tmp_list = []
	for model in config['Models']['list']:
		cur.execute(f"SELECT COUNT(*) FROM model_evaluation_rag WHERE model='{model}' AND review={i} AND class='general';")
		eval = cur.fetchone()[0]
		if(eval == 0):
			eval = 0.1
		tmp_list.append(eval)
	review_list0[i] = tmp_list
	tmp_list = []
	for model in config['Models']['list']:
		cur.execute(f"SELECT COUNT(*) FROM model_evaluation_rag WHERE model='{model}' AND review={i} AND class='programming';")
		eval = cur.fetchone()[0]
		if(eval == 0):
			eval = 0.1
		tmp_list.append(eval)
	review_list1[i] = tmp_list
	tmp_list = []
	for model in config['Models']['list']:
		cur.execute(f"SELECT COUNT(*) FROM model_evaluation_rag WHERE model='{model}' AND review={i} AND class='programming-rag';")
		eval = cur.fetchone()[0]
		if(eval == 0):
			eval = 0.1
		tmp_list.append(eval)
	review_list2[i] = tmp_list
```

```
# set the width of the bar and create subplots
barWidth = 0.275
fig, ax = plt.subplots(figsize =(9, 10))

# plot the bars by grouping them by review
br = np.arange(len(config["Models"]["list"]))
for i in range(0, len(review_list0)):
        plt.barh(br, review_list0[i], color =colors_list[i], edgecolor ='grey',
         ↪  label=labels_list[i], height=barWidth)
        plt.barh(br, review_list1[i], color =colors_list2[i], edgecolor ='grey',
         ↪  label=labels_list2[i], height=barWidth, left=review_list0[i])
        plt.barh(br, review_list2[i], color =colors_list3[i], edgecolor ='grey',
         ↪  label=labels_list3[i], height=barWidth, left=[sum(x) for x in
         ↪  zip(review_list0[i], review_list1[i])])
        br = [x + barWidth for x in br]

# add labels and title
plt.xlabel('Number of answers', fontsize = 12)
plt.ylabel('Model', fontsize = 12)
plt.yticks([r + barWidth for r in range(len(config["Models"]["list"]))],
↪  config["Models"]["list"], fontsize = 8)
plt.title('Model evaluation by model')

ax.annotate('Class 0 are general questions / tasks\nClass 1 are questions / tasks to
↪  create yaml files and write code\n Class 2 are questions / tasks to create yaml files
↪  and write code with provided information (RAG)',xy = (1.0, -0.1),xycoords='axes
↪  fraction',ha='right',va="center",fontsize=6)
ax.annotate('Temperature is 0.3',xy = (0, -0.1),xycoords='axes
↪  fraction',ha='left',va="center",fontsize=6)
ax.annotate('with RAG',xy = (0.9, 0.007),xycoords='axes
↪  fraction',ha='left',va="center",fontsize=10)


# add legend + show
plt.legend()
#plt.show()
plt.savefig('model_evaluation_by_model_detailed.png',format='png', bbox_inches='tight')
```

```
# %%
cur.close()
```

Figure B.5.1: Script for analysation of calls to FlowiseAI RAG implementation

B.6 Evaluation Table

[4]

id	model	question	review
1	qwen2.5-coder:32b-instruct-q8_0	1	2
2	qwen2.5-coder:32b-instruct-q8_0	2	2
3	qwen2.5-coder:32b-instruct-q8_0	3	2
4	qwen2.5-coder:32b-instruct-q8_0	4	2
5	qwen2.5-coder:32b-instruct-q8_0	5	2
6	qwen2.5-coder:32b-instruct-q8_0	6	2
7	qwen2.5-coder:32b-instruct-q8_0	7	2
8	qwen2.5-coder:32b-instruct-q8_0	8	2
9	qwen2.5-coder:32b-instruct-q8_0	9	2
10	qwen2.5-coder:32b-instruct-q8_0	10	2
11	qwen2.5-coder:32b-instruct-q8_0	11	2
12	qwen2.5-coder:32b-instruct-q8_0	12	1
13	qwen2.5-coder:32b-instruct-q8_0	13	1
14	qwen2.5-coder:32b-instruct-q8_0	14	1
15	qwen2.5-coder:32b-instruct-q8_0	15	2
16	mistral-small:22b-instruct-2409-fp16	1	2
17	mistral-small:22b-instruct-2409-fp16	2	2
18	mistral-small:22b-instruct-2409-fp16	3	2
19	mistral-small:22b-instruct-2409-fp16	4	0
20	mistral-small:22b-instruct-2409-fp16	5	1
21	mistral-small:22b-instruct-2409-fp16	6	1
22	mistral-small:22b-instruct-2409-fp16	7	1
23	mistral-small:22b-instruct-2409-fp16	8	1
24	mistral-small:22b-instruct-2409-fp16	9	1
25	mistral-small:22b-instruct-2409-fp16	10	2
26	mistral-small:22b-instruct-2409-fp16	11	1

[4] The questions are replaced by the ID given in the questions overview. The responses are not shown, due to formatting reasons. They are accessible in the GitHub repository.

27	mistral-small:22b-instruct-2409-fp16	12	2
28	mistral-small:22b-instruct-2409-fp16	13	1
29	mistral-small:22b-instruct-2409-fp16	14	0
30	mistral-small:22b-instruct-2409-fp16	15	1
31	llama3.1:70b	1	2
32	llama3.1:70b	2	2
33	llama3.1:70b	3	2
34	llama3.1:70b	4	0
35	llama3.1:70b	5	1
36	llama3.1:70b	6	1
37	llama3.1:70b	7	2
38	llama3.1:70b	8	1
39	llama3.1:70b	9	2
40	llama3.1:70b	10	1
41	llama3.1:70b	11	2
42	llama3.1:70b	12	2
43	llama3.1:70b	13	1
44	llama3.1:70b	14	1
45	llama3.1:70b	15	2
46	llama3.3:70b-instruct-q4_K_M	1	2
47	llama3.3:70b-instruct-q4_K_M	2	1
48	llama3.3:70b-instruct-q4_K_M	3	1
49	llama3.3:70b-instruct-q4_K_M	4	0
50	llama3.3:70b-instruct-q4_K_M	5	2
51	llama3.3:70b-instruct-q4_K_M	6	2
52	llama3.3:70b-instruct-q4_K_M	7	2
53	llama3.3:70b-instruct-q4_K_M	8	1
54	llama3.3:70b-instruct-q4_K_M	9	0
55	llama3.3:70b-instruct-q4_K_M	10	0
56	llama3.3:70b-instruct-q4_K_M	11	1
57	llama3.3:70b-instruct-q4_K_M	12	1
58	llama3.3:70b-instruct-q4_K_M	13	2
59	llama3.3:70b-instruct-q4_K_M	14	2
60	llama3.3:70b-instruct-q4_K_M	15	2
61	hf.co/matteogeniaccio/phi-4:F16	1	2
62	hf.co/matteogeniaccio/phi-4:F16	2	2
63	hf.co/matteogeniaccio/phi-4:F16	3	2
64	hf.co/matteogeniaccio/phi-4:F16	4	1
65	hf.co/matteogeniaccio/phi-4:F16	5	1

66	hf.co/matteogeniaccio/phi-4:F16	6	1
67	hf.co/matteogeniaccio/phi-4:F16	7	1
68	hf.co/matteogeniaccio/phi-4:F16	8	2
69	hf.co/matteogeniaccio/phi-4:F16	9	2
70	hf.co/matteogeniaccio/phi-4:F16	10	2
71	hf.co/matteogeniaccio/phi-4:F16	11	0
72	hf.co/matteogeniaccio/phi-4:F16	12	1
73	hf.co/matteogeniaccio/phi-4:F16	13	2
74	hf.co/matteogeniaccio/phi-4:F16	14	0
75	hf.co/matteogeniaccio/phi-4:F16	15	2

Table B.6.1: Table of Rating of FlowiseAI RAG Implementation Calls

B.7 Final Workflow evaluation

```
import requests
import psycopg2
import toml

file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()

config = toml.load('model_evaluation.toml')

con = psycopg2.connect(
    dbname="model_evaluation",
    user="local",
    password=passwd,
    host="141.33.165.24"
)



API_URL =
↪   "http://141.33.165.24:8000/api/v1/prediction/bddddad9-3b09-44f1-af80-e6788a58d906"

def query(payload):
    response = requests.post(API_URL, json=payload)
    return response.json()
```

```
cur = con.cursor()
cur.execute("CREATE TABLE IF NOT EXISTS model_evaluation_workflow (id SERIAL PRIMARY KEY,
↪  response VARCHAR, question VARCHAR, review INTEGER, class VARCHAR);")
con.commit()

for i in config['Questions']['list']:
    print(f'Question: {i[0]}')
    output = query({
        "question": i[0]
    })

    response = query(output).get('text')
    print(response)
    print("Starting upload to db")
    cur.execute("INSERT INTO model_evaluation_workflow (response, question , review,
    ↪  class) VALUES (%s, %s, NULL, %s)", (response, i[0], i[1]))
    con.commit()

con.close()
```

Figure B.7.1: Request script

```
import psycopg2
from rich.console import Console
from rich.markdown import Markdown

file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()

console = Console()

con = psycopg2.connect(
    dbname="model_evaluation",
    user="local",
    password=passwd,
    host="141.33.165.24"
)

cur = con.cursor()
cur.execute("SELECT COUNT(*) FROM model_evaluation_workflow;")
number_of_records = cur.fetchone()[0]
```

```
is_true = True
console.clear()
while is_true:
    console.clear()
    console.print("""
                This is a small command line tool to review the generated responses.
                ↪  Please answer the following questions with y or n. y for true and n
                ↪  for false. Did you understand?
                In the following you will see a question and a the response.
                """)
    user_input = input(":")
    if user_input in ["true", "1", "yes", "y", "t"]:
            is_true = False
    elif user_input in ["false", "0", "no", "n", "f"]:
            is_true = True

for i in range(number_of_records):
    console.clear()
    console.print(Markdown(f"### Question:\n"))
    cur.execute(f"SELECT question FROM model_evaluation_workflow WHERE id={i+1};")
    console.print(Markdown(cur.fetchone()[0]))

    console.print("\n" * 4)

    console.print(Markdown(f"### Response: \n"))
    cur.execute(f"SELECT response FROM model_evaluation_workflow WHERE id={i+1};")
    console.print(Markdown(cur.fetchone()[0]))

    user_input = int(input(":").strip().lower())

    cur.execute(f"UPDATE model_evaluation_workflow SET review={user_input} WHERE
    ↪  id={i+1};")
    con.commit()
    console.clear()

con.close()
```

Figure B.7.2: Evaluation script

```
import pandas as pd
```

```python
import psycopg2

file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()

data = [['id', 'question', 'review']]



con = psycopg2.connect(
    dbname="model_evaluation",
    user="local",
    password=passwd,
    host="141.33.165.24"
)

cur = con.cursor()
cur.execute("SELECT COUNT(*) FROM model_evaluation_workflow;")

for i in range(0, cur.fetchone()[0]):
    cur.execute("SELECT * FROM model_evaluation_rag WHERE id=%s;", (i+1,))
    current = cur.fetchone()
    datatmp = [current[0], current[2], current[3]]
    data.append(datatmp)

df = pd.DataFrame(data[1:], columns=data[0])

file = open("model_evaluation_table_workflow.md", "w")
file.write(df.to_markdown(index=False))
file.close()
```

Figure B.7.3: Table generation script

B.8 Analyses (final)

```python
# %% [markdown]
# # Ollama model evaluation analyses
# - this notebook shows some analyses for the ollama evaluations
```

```python
# %%
# import libraries
import toml
import psycopg2
import matplotlib.pyplot as plt
import numpy as np

# %%
# load configurations
config = toml.load('model_evaluation.toml')
file = open("/home/tom/Documents/AIP/Bachelorarbeit/creds/psql.pass", "r")
passwd = file.read().strip()
file.close()

# %%
# Connect to the database
con = psycopg2.connect(
    dbname="model_evaluation",
    user="local",
    password=passwd,
    host="141.33.165.24"
)
cur = con.cursor()

# %%
# get the number of evaluations for each class
review_list = []
for i in range(0,3):
    cur.execute(f"SELECT COUNT(*) FROM model_evaluation_workflow WHERE review={i};")
    review_list.append(cur.fetchone()[0])
    if(review_list[i] == 0):
        review_list[i] = 0.1
labels = ['bad', 'average', 'great']

# create the plot
plt.bar(labels, review_list, color=['r', 'y', 'g'])
plt.title('Workflow call')
plt.ylabel('Number of evaluations')
#plt.show()
```

```python
plt.savefig('model_evaluation_workflow.png',format='png', bbox_inches='tight')

# %%
# get the number of evaluations for each class
review_list = []
for i in range(0,3):
    cur.execute(f"SELECT COUNT(*) FROM model_evaluation WHERE review={i} AND
    ↪  model='qwen2.5-coder:32b-instruct-q8_0';")
    review_list.append(cur.fetchone()[0])
labels = ['bad', 'average', 'great']

# create the plot
plt.bar(labels, review_list, color=['r', 'y', 'g'])
plt.title('Direct call')
plt.ylabel('Number of evaluations')
#plt.show()
plt.savefig('model_evaluation.png',format='png', bbox_inches='tight')

# %%
cur.close()
```

[5]

Figure B.8.1: Analysation script

[5] notice that the original file type was an jupyter notebook; the notebook was formatted to an python file to format and show it correctly in this appendix

B.9 Evaluation Table

Table B.9.1: Workflow evaluation table

id	question	review
1	1	2
2	2	2
3	3	2
4	4	2
5	5	1
6	6	2
7	7	2
8	8	2
9	9	2
10	10	2
11	11	2
12	12	2
13	13	2
14	14	1
15	15	1

B.10 Evaluation Tables

	bad	average	great
Without RAG	33	18	24
With RAG	8	27	40

Table B.10.1: Comparison of the model performance summarized by rating to illustrate the impact of RAG.

	bad	average	great
qwen2.5-coder:32b-instruct-q8_0	1	2	12
mistral-small:22b-instruct-2409-fp16	9	3	3
llama3.1:70b	12	1	2
llama3.3:70b-instruct-q4_K_M	4	7	4
hf.co/matteogeniaccio/phi-4:F16	7	5	3

Table B.10.2: Model performance for LLM models without RAG.

	bad	average	great
qwen2.5-coder:32b-instruct-q8_0	0	3	12
mistral-small:22b-instruct-2409-fp16	2	8	5
llama3.1:70b	1	6	8
llama3.3:70b-instruct-q4_K_M	3	5	7
hf.co/matteogeniaccio/phi-4:F16	2	5	8

Table B.10.3: Model performance for LLM models with RAG.

model	bad	average	great	kind
qwen2.5-coder:32b-instruct-q8_0	0	1	4	general
mistral-small:22b-instruct-2409-fp16	2	1	2	
llama3.1:70b	3	1	1	
llama3.3:70b-instruct-q4_K_M	3	0	2	
hf.co/matteogeniaccio/phi-4:F16	2	1	2	
qwen2.5-coder:32b-instruct-q8_0	1	1	3	programming
mistral-small:22b-instruct-2409-fp16	4	0	1	
llama3.1:70b	4	0	1	
llama3.3:70b-instruct-q4_K_M	1	2	2	
hf.co/matteogeniaccio/phi-4:F16	3	1	1	
qwen2.5-coder:32b-instruct-q8_0	0	0	5	programming-rag
mistral-small:22b-instruct-2409-fp16	3	2	0	
llama3.1:70b	5	0	0	
llama3.3:70b-instruct-q4_K_M	0	5	0	
hf.co/matteogeniaccio/phi-4:F16	2	3	0	

Table B.10.4: Model performance for LLM models without RAG by task categories.

	bad	average	great	kind
qwen2.5-coder:32b-instruct-q8_0	0	0	5	general
mistral-small:22b-instruct-2409-fp16	1	2	2	
llama3.1:70b	1	2	2	
llama3.3:70b-instruct-q4_K_M	1	2	2	
hf.co/matteogeniaccio/phi-4:F16	0	3	2	
qwen2.5-coder:32b-instruct-q8_0	0	3	2	programming
mistral-small:22b-instruct-2409-fp16	1	2	2	
llama3.1:70b	0	2	3	
llama3.3:70b-instruct-q4_K_M	0	1	4	
hf.co/matteogeniaccio/phi-4:F16	1	1	3	
qwen2.5-coder:32b-instruct-q8_0	0	0	5	programming-rag
mistral-small:22b-instruct-2409-fp16	0	4	1	
llama3.1:70b	0	2	3	
llama3.3:70b-instruct-q4_K_M	2	2	1	
hf.co/matteogeniaccio/phi-4:F16	1	1	3	

Table B.10.5: Model performance for LLM models with RAG by task categories.

Appendix C

Docker

C.1 Docker compose files

```
# based on: https://www.docker.com/blog/how-to-use-the-postgres-docker-official-image/
services:
  postgres:
    image: postgres:alpine
    restart: always
    environment:
      - POSTGRES_USER=****
      - POSTGRES_PASSWORD=****
      - POSTGRES_DB=flowise
    ports:
      - "5432:5432"
    volumes:
     - /root/docker-data/postgres:/var/lib/postgresql/data
```

Figure C.1.1: Postgres

```
# source: https://github.com/open-webui/open-webui/blob/main/docker-compose.yaml
services:
  open-webui:
    image: ghcr.io/open-webui/open-webui:${WEBUI_DOCKER_TAG-main}
    container_name: open-webui
    volumes:
      - /root/docker-data/openwebui:/app/backend/data
    ports:
      - ${OPEN_WEBUI_PORT-3000}:8080
```

```
    environment:
      - 'OLLAMA_BASE_URL=http://ollama:11434'
    extra_hosts:
      - host.docker.internal:host-gateway
    restart: unless-stopped
  pipelines-openwebui:
    build:
      context: .
      dockerfile: Dockerfile
    image: openwebui-pipelines:self
    ports:
      - 9099:9099
    environment:
      - REANA_SERVER_URL=https://reana-p4n.aip.de
    volumes:
      - /root/docker-data/pipelines-openwebui:/app/pipelines
    extra_hosts:
      - host.docker.internal:host-gateway
    restart: always
```

Figure C.1.2: Open WebUI

```
# source: https://github.com/FlowiseAI/Flowise/blob/main/docker/docker-compose.yml
version: '3.1'

services:
    flowise:
        image: flowiseai/flowise
        restart: always
        environment:
            - PORT=${PORT}
            - CORS_ORIGINS=${CORS_ORIGINS}
            - IFRAME_ORIGINS=${IFRAME_ORIGINS}
            - FLOWISE_USERNAME=${FLOWISE_USERNAME}
            - FLOWISE_PASSWORD=${FLOWISE_PASSWORD}
            - FLOWISE_FILE_SIZE_LIMIT=${FLOWISE_FILE_SIZE_LIMIT}
            - DEBUG=${DEBUG}
            - DATABASE_PATH=${DATABASE_PATH}
            - DATABASE_TYPE=${DATABASE_TYPE}
```

```
            - DATABASE_PORT=${DATABASE_PORT}
            - DATABASE_HOST=${DATABASE_HOST}
            - DATABASE_NAME=${DATABASE_NAME}
            - DATABASE_USER=${DATABASE_USER}
            - DATABASE_PASSWORD=${DATABASE_PASSWORD}
            - DATABASE_SSL=${DATABASE_SSL}
            - DATABASE_SSL_KEY_BASE64=${DATABASE_SSL_KEY_BASE64}
            - APIKEY_PATH=${APIKEY_PATH}
            - SECRETKEY_PATH=${SECRETKEY_PATH}
            - FLOWISE_SECRETKEY_OVERWRITE=${FLOWISE_SECRETKEY_OVERWRITE}
            - LOG_LEVEL=${LOG_LEVEL}
            - LOG_PATH=${LOG_PATH}
            - BLOB_STORAGE_PATH=${BLOB_STORAGE_PATH}
            - DISABLE_FLOWISE_TELEMETRY=${DISABLE_FLOWISE_TELEMETRY}
            - MODEL_LIST_CONFIG_JSON=${MODEL_LIST_CONFIG_JSON}
        ports:
            - '${PORT}:${PORT}'
        volumes:
          - /root/docker-data/flowise:/root/.flowise
        entrypoint: /bin/sh -c "sleep 3; flowise start"
```

Figure C.1.3: FlowiseAI

```
# source: https://qdrant.tech/documentation/guides/installation/
services:
  qdrant:
    image: qdrant/qdrant:latest
    restart: always
    container_name: qdrant
    ports:
      - 6333:6333
      - 6334:6334
    expose:
      - 6333
      - 6334
      - 6335
    configs:
      - source: qdrant_config
        target: /qdrant/config/production.yaml
    volumes:
      - /root/docker-data/qdrant:/qdrant/storage
```

```
configs:
  qdrant_config:
    content: |
      log_level: INFO
```

Figure C.1.4: Qdrant

```
services:
  aiptools:
    build:
      context: .
      dockerfile: Dockerfile
    image: aiptools:latest
    ports:
      - 5000:5000
    environment:
      - REANA_SERVER_URL=https://reana-p4n.aip.de
    extra_hosts:
      - host.docker.internal:host-gateway
    restart: always
```

Figure C.1.5: Own Tool Container - aiptools

Appendix D

First steps

D.1 Python Runner with Matplotlib

```
# this code is based on a example from the openwebui pipline repository
↪   (https://github.com/open-webui/pipelines/blob/main/examples/pipelines/integrations/py
↪   thon_code_pipeline.py)
# and https://github.com/FlowiseAI/Flowise/discussions/2581#discussioncomment-10607580

from typing import List, Union, Generator, Iterator
from schemas import OpenAIChatMessage
import subprocess
import requests

class Pipeline:
    def __init__(self):
        # Optionally, you can set the id and name of the pipeline.
        # Best practice is to not specify the id so that it can be automatically inferred
        ↪   from the filename, so that users can install multiple versions of the same
        ↪   pipeline.
        # The identifier must be unique across all pipelines.
        # The identifier must be an alphanumeric string that can include underscores or
        ↪   hyphens. It cannot contain spaces, special characters, slashes, or
        ↪   backslashes.
        # self.id = "python_code_pipeline"
        self.name = "Plotting Pipeline"
        pass
```

```python
    async def on_startup(self):
        # This function is called when the server is started.
        print(f"on_startup:{__name__}")
        pass

    async def on_shutdown(self):
        # This function is called when the server is stopped.
        print(f"on_shutdown:{__name__}")
        pass

    def execute_python_code(self, code):
        try:
            result = subprocess.run(
                ["python", "-c", code], capture_output=True, text=True, check=True
            )
            stdout = result.stdout.strip()
            return stdout, result.returncode
        except subprocess.CalledProcessError as e:
            return e.output.strip(), e.returncode

    def pipe(
        self, user_message: str, model_id: str, messages: List[dict], body: dict
    ) -> Union[str, Generator, Iterator]:
        # This is where you can add your custom pipelines like RAG.
        print(f"pipe:{__name__}")

        print(messages)
        print(user_message)

        #c3cc5997-7b42-4930-9ebb-2856be8744a5
        url = 'http://141.33.165.24:8000/api/v1/prediction/5467f902-a69f-4bb1-8db7-40a70d
        ↪ a88d64'

        headers = {'Content-Type': 'application/json'}
        data = {'question': user_message}

        response = requests.post(url,  json=data, headers=headers)
        if body.get("title", False):
            print("Title Generation")
```

```
            return "Python Code Pipeline"
        else:
            stdout, return_code = self.execute_python_code(response.json()['text'].split(
            ↪  '```python')[1].split('```')[0])
            output = response.json()['text'] + '\n script output:\n' + stdout
            return output
```

Figure D.1.1: Open WebUI pipeline

D.2 Implementation to Generate and Run REANA Workflows

```
# based on
↪ https://github.com/FlowiseAI/Flowise/discussions/2581#discussioncomment-10607580

from typing import List, Union, Generator, Iterator
from pydantic import BaseModel
import requests
import json
import reana_commons.validation.utils as rcv
import yaml
import re
import reana_client.api.client as rcl
import credentails

class Pipeline:
    class Valves(BaseModel):
        pass  # No API key needed for Flowise API

    def __init__(self):
        self.name = "Reana-Runner"
        self.valves = self.Valves()

    async def on_startup(self):
        print(f"on_startup:{__name__}")

    async def on_shutdown(self):
        print(f"on_shutdown:{__name__}")

    def pipe(
        self, user_message: str, model_id: str, messages: List[dict], body: dict
    ) -> Union[str, Generator, Iterator]:
        print(f"pipe:{__name__}")

        print(messages)
        print(user_message)

        API_URL = "http://141.33.165.24:8000/api/v1/prediction/47029097-b6f3-4589-94d4-ed
        ↪ 4d4e7ba648"
```

```
headers = {
    "Content-Type": "application/json"
}

# Creating the payload based on your example
payload = {
    "question": user_message
}

print("Payload:", payload)

text = ""

try:
    r = requests.post(
        url=API_URL,
        json=payload,
        headers=headers,
        stream=True,
    )

    r.raise_for_status()

    if body.get("stream"):
        for line in r.iter_lines():
            line_data = line.decode('utf-8')
            print("Line data:", line_data)
            # Parse the JSON line and extract the text part
            response_json = json.loads(line_data)
            if "text" in response_json:
                print("Streaming text:", response_json["text"])
                text += response_json["text"]
            validation = False
            reana_file = {}
            files = []
            file_list = []
            try:
                files = re.split(r'```.*?\n', text)
```

```
                files[len(files)-1] = files[len(files)-1].replace('```', '')
                files = list(filter(lambda item: item not in ['\n', ''], files))
                reana_file = yaml.safe_load(files[0])
                file_list = reana_file["inputs"]["files"]
                if(len(rcv.validate_reana_yaml(reana_file)) < 1):
                    validation = True
            except Exception as e:
                    print("Error:", e)
                    validation = False

            if(validation):
                rcl.create_workflow(reana_file, "hello",
                ↪  credentails.reana_api_key)
                for i in range(1, len(files)):
                    rcl.upload_file("hello", files[i], file_list[i-1],
                    ↪  credentails.reana_api_key)
                rcl.start_workflow("hello", credentails.reana_api_key, None)
                return "# This yaml specifications should work and will be run\n"
                ↪  + text
            else:
                return "# The response is not a valid REANA YAML. Please generate
                ↪  again. \n" + text
    else:
        response_json = r.json()
        print("Response JSON:", response_json)
        # Return only the "text" part of the response
        if "text" in response_json:
            print("Text:", response_json["text"])
            validation = False
            try:
                data = yaml.safe_load(response_json["text"].split("```")[1].split
                ↪  ("```")[0])
                if(len(rcv.validate_reana_yaml(data)) < 1):
                    validation = True
            except Exception as e:
                    print("Error:", e)
                    validation = False

            if(validation):
```

```
                return response_json["text"]
            else:
                return "The response is not a valid REANA YAML. Please generate
                ↪  again."
        else:
            print("No text in response")
            return "No text in response"
    except Exception as e:
        print("Error:", e + "\n" + text)
        return f"Error: {e}"
```

Figure D.2.1: Open WebUI pipeline

Appendix E

Final setup

E.1 Chain of toughs

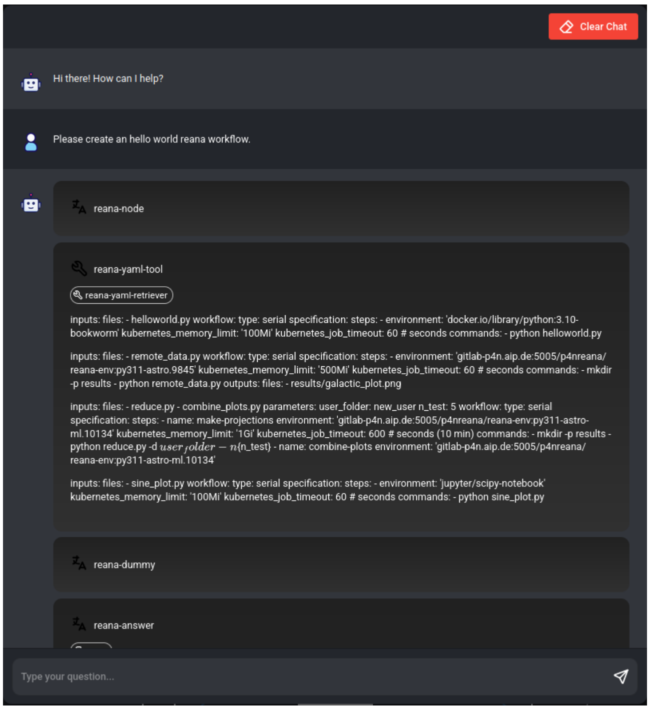

Figure E.1.1: Chain of toughs 1

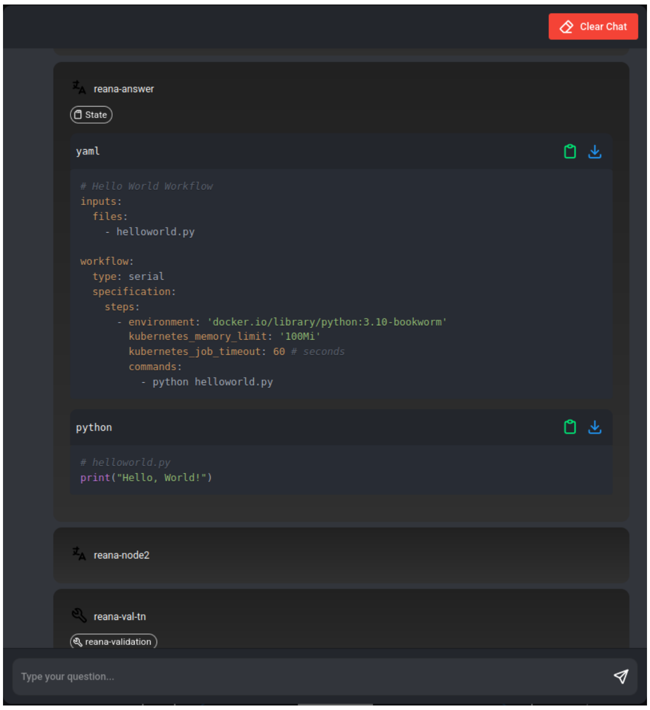

Figure E.1.2: Chain of toughs 2

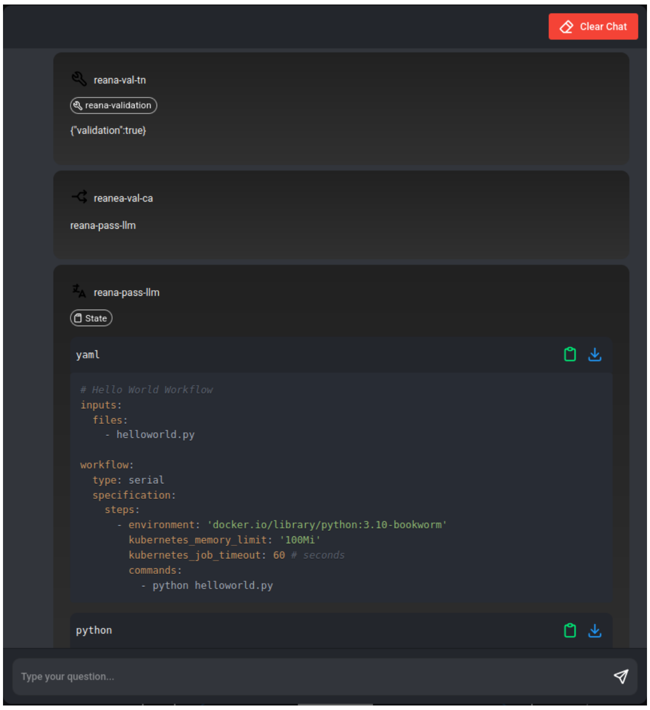

Figure E.1.3: Chain of toughs 3

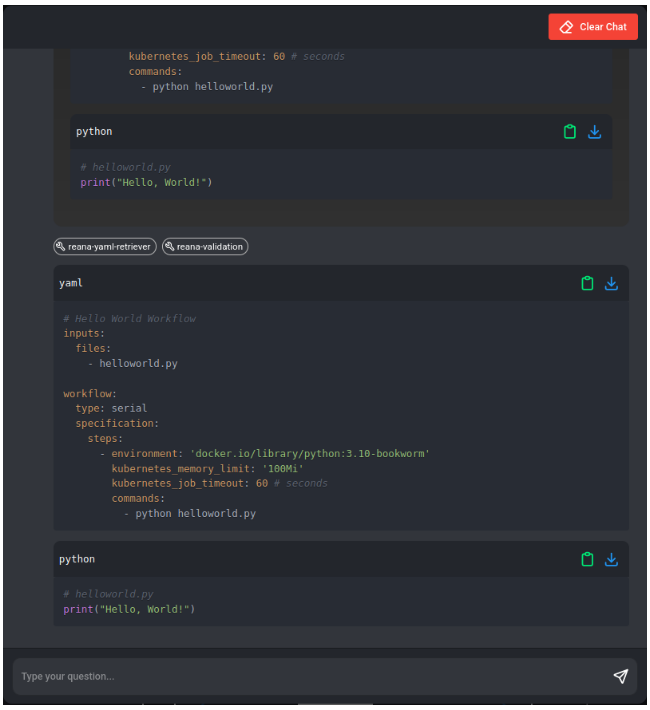

Figure E.1.4: Chain of toughs 4

E.2 FlowiseAI workflows

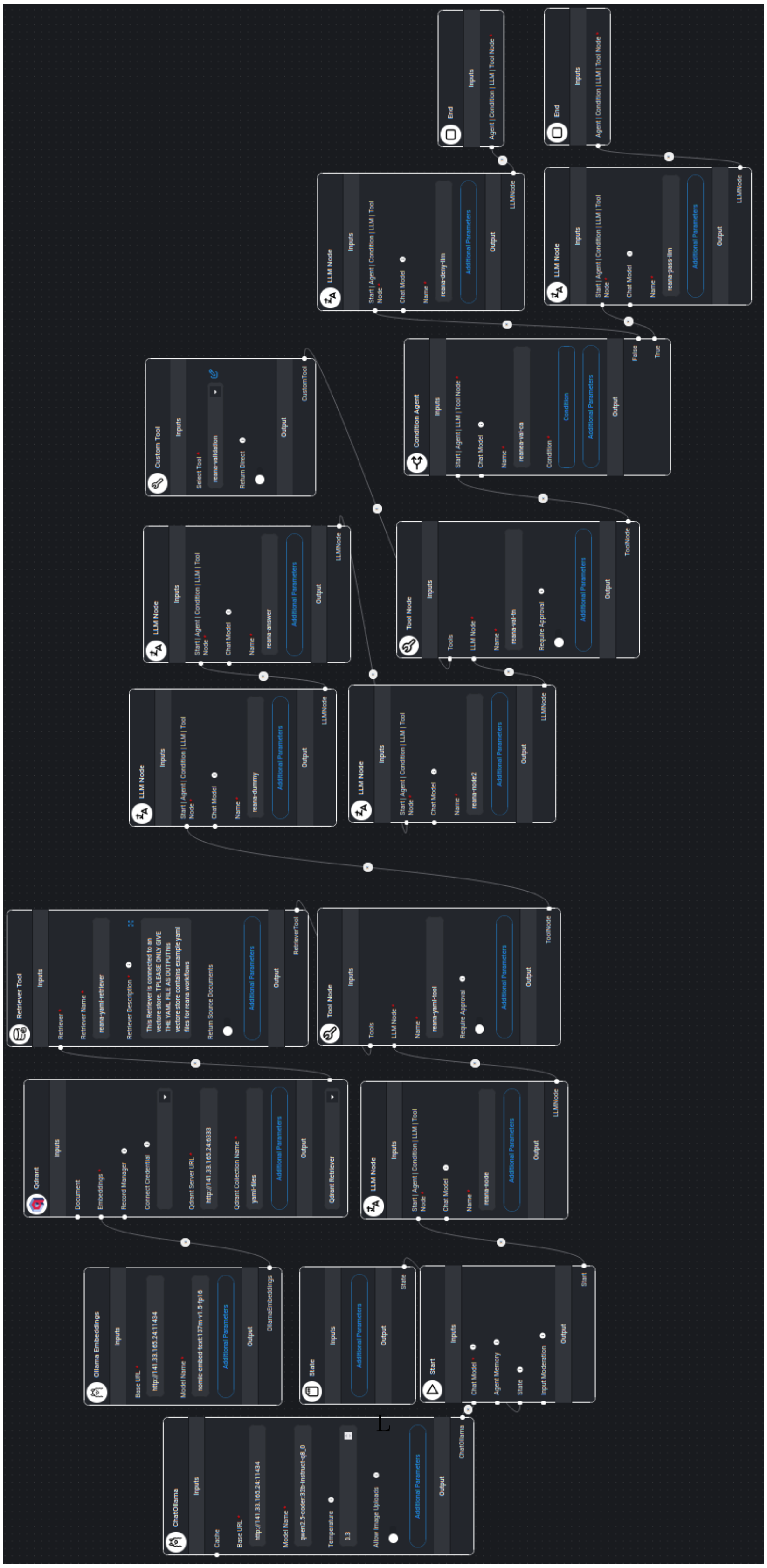

Figure E.2.1: REANA generation FlowiseAI workflow

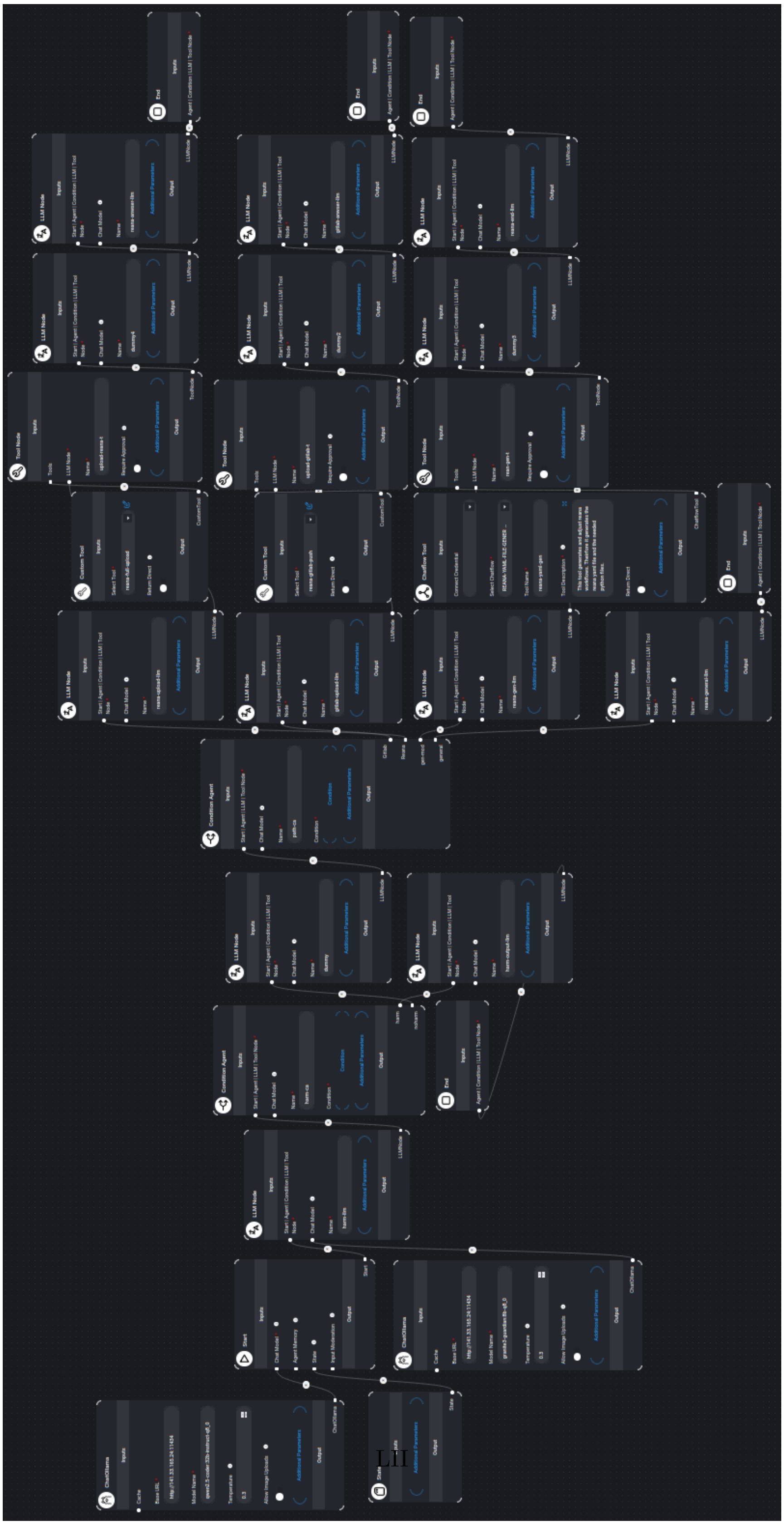

Figure E.2.2: Open WebUI endpoint FlowiseAI workflow

E.3 Open WebUI Pipeline

```python
# source: https://github.com/FlowiseAI/Flowise/discussions/2581#discussioncomment-10607580

from typing import List, Union, Generator, Iterator
from pydantic import BaseModel
import requests
import json

class Pipeline:

    def __init__(self):
        self.name = "Reana-Final"
        pass

    async def on_startup(self):
        print(f"on_startup:{__name__}")

    async def on_shutdown(self):
        print(f"on_shutdown:{__name__}")

    def pipe(
        self, user_message: str, model_id: str, messages: List[dict], body: dict
    ) -> Union[str, Generator, Iterator]:

        # FlowiseAI API call (Open WebUI endpoint)
        API_URL = "http://141.33.165.24:8000/api/v1/prediction/bddddad9-3b09-44f1-af80-e6
        ↪ 788a58d906"

        headers = {
            "Content-Type": "application/json"
        }

        # adding last system message as history
        if(len(messages) > 1):
            user_message = user_message + "\n History:\n" +
            ↪ messages[len(messages)-2].get("content")
```

```python
payload = {
    "question": user_message,
}

try:
    r = requests.post(
        url=API_URL,
        json=payload,
        headers=headers,
        stream=True,
    )

    r.raise_for_status()

    if body.get("stream"):
        for line in r.iter_lines():
            line_data = line.decode('utf-8')
            response_json = json.loads(line_data)
            if "text" in response_json:
                yield response_json["text"]
    else:
        response_json = r.json()
        if "text" in response_json:
            return response_json["text"]
        else:
            return "No text in response"
except Exception as e:
    # catch errors and give user instructions
    print.error(f"An error occurred: {e}")
    yield """An error occurred while processing the request:
                    Please only send tasks matching the topic REANA.
                    Here is a list of possible tasks:
                    - generate an workflow
                    - modify the last workflow
                    - upload to GitLab
                    - upload to REANA
                    Example for these are:
                    - "Please create an REANA workflow which,..."
```

```
- "Please change the parameter environment to
↪  python3.9-slim"
- "Please upload this to GitLab"
- "Please upload this to REANA"
You probably send harmful data"""
```

Figure E.3.1: OpenWebUI pipeline of the final prototype

E.4 Time consumption

	Answer

Direct Ollama call	REANA is a reusable and reproducible analysis platform that supports various computational workflows using different types of computational backends such as Kubernetes, Slurm, or HTCondor. Below is a simple example of how you can create a ”Hello World” workflow in REANA. ### Step 1: Create the Workflow Specification Firstly, you need to define your workflow specification file (e.g., ‘reana.yaml‘). This file will describe the steps and tools required for your analysis. Here’s an example of what this might look like: “‘yaml specification: type: serial steps: - environment: ’python:3.8-slim’ computational_backend: kubernetes command: — python ¡¡EOF print(”Hello, World!”) EOF “‘ This workflow uses a Python 3.8 slim Docker image to run the script that prints ”Hello, World!”. ### Step 2: Create Your Workflow To create your REANA workflow, you need to have the ‘reana-client‘ installed and configured with access to a running REANA server. If not already done, install it using pip: “‘bash pip install reana-client “‘ Then, configure the client by pointing it towards your REANA server (replace ‘¡REANA_SERVER_URL¿‘ with the actual URL of your REANA server): “‘bash export REANA_SERVER_URL=¡REANA_SERVER_URL¿ reana-client ping “‘ ### Step 3: Upload and Run Your Workflow Once you have your ‘reana.yaml‘ file ready, upload it to REANA: “‘bash reana-client create -n my-hello-world-workflow reana-client upload reana.yaml reana-client start my-hello-world-workflow “‘ ### Step 4: Check the Results You can check the status of your workflow and its output using the following commands: To see the status: “‘bash reana-client status -w my-hello-world-workflow “‘ To list all files in your workspace (including logs): “‘bash reana-client ls -w my-hello-world-workflow “‘ And to download specific files, such as log files or output files: “‘bash reana-client download ¡filename¿ -w my-hello-world-workflow “‘ This is a basic example of setting up and running a ”Hello World” workflow in REANA. Depending on your needs, you can expand this with more complex workflows involving multiple steps, different computational backends, or even interactive sessions for data analysis.

Simple RAG call	Based on the context provided, I can help you draft a simple ”Hello World” REANA workflow similar to the ones given in the examples. Here’s how your ‘helloworld.yaml‘ file might look: “‘yaml inputs: files: - helloworld.py workflow: type: serial specification: steps: - environment: ’docker.io/library/python:3.10-bookworm’ kubernetes_memory_limit: ’100Mi’ kubernetes_job_timeout: 60 # seconds commands: - python helloworld.py “‘ And your ‘helloworld.py‘ file should contain: “‘python print(”Hello World!”) “‘ This workflow will run the Python script that prints ”Hello World!” to the console.
FlowiseAI call	Pass Input: “‘yaml # Hello World Workflow inputs: files: - helloworld.py workflow: type: serial specification: steps: - environment: ’docker.io/library/python:3.10-bookworm’ kubernetes_memory_limit: ’100Mi’ kubernetes_job_timeout: 60 # seconds commands: - python helloworld.py “‘ “‘python # helloworld.py print(”Hello, World!”) “‘

Table E.4.1: Time consumption table - answers

```
import requests
import time
import pandas as pd

question = "Please create an hello world REANA workflow."
times = []
answers = []

# ollama call
url = 'http://141.33.165.24:11434/api/generate'
payload = {
"model": 'qwen2.5:32b-instruct-q8_0',
"stream" : False,
"prompt": question,
"options": {
    "temperature": 0.3
},
```

```
}

start = time.time()
response = requests.post(url, json=payload).json()
end = time.time()
times.append(end - start)
answers.append([response.get("response")])

#rag call
API_URL =
↪  "http://141.33.165.24:8000/api/v1/prediction/6b74c5bd-bcf9-4a29-824f-06d0028cce74"

payload = ({
    "question": question,
    "overrideConfig": {
        "modelName": "qwen2.5:32b-instruct-q8_0",
    }
})

start = time.time()
response = requests.post(API_URL, json=payload).json()
end = time.time()
times.append(end - start)
answers.append([response.get("text")])


# flowise call
API_URL =
↪  "http://141.33.165.24:8000/api/v1/prediction/bddddad9-3b09-44f1-af80-e6788a58d906"

payload = {
    "question": question
}


start = time.time()
response = requests.post(API_URL, json=payload).json()
end = time.time()
times.append(end - start)
```

```
answers.append([response.get("text")])

# round and format values
times = [round(time, 2) for time in times]
times = [f"{time:.2f}" for time in times]

# lists to df
df = pd.DataFrame(times, columns=["Time(in s)"], index=["Direct Ollama call", "Simple RAG
↪  call", "FlowiseAI call"])

df2 = pd.DataFrame(answers, columns=["Answer"], index=["Direct Ollama call", "Simple RAG
↪  call", "FlowiseAI call"])

# generate latex tables and save
table = df.to_latex(index=True, escape=True, column_format="|l|r|", caption=r"Time
↪  consumption table")
table2 = df2.to_latex(index=True, escape=True, column_format="|l|r|", caption=r"Time
↪  consumption table - answers")

file = open("time_evalution.table.tex", "w")
file.write(table)
file.close()

file = open("time_evalution.answers.tex", "w")
file.write(table2)
file.close()
```

Figure E.4.1: Time consumption script

E.5 Usage of Prototype Examples

E.5.1 Generating an workflow

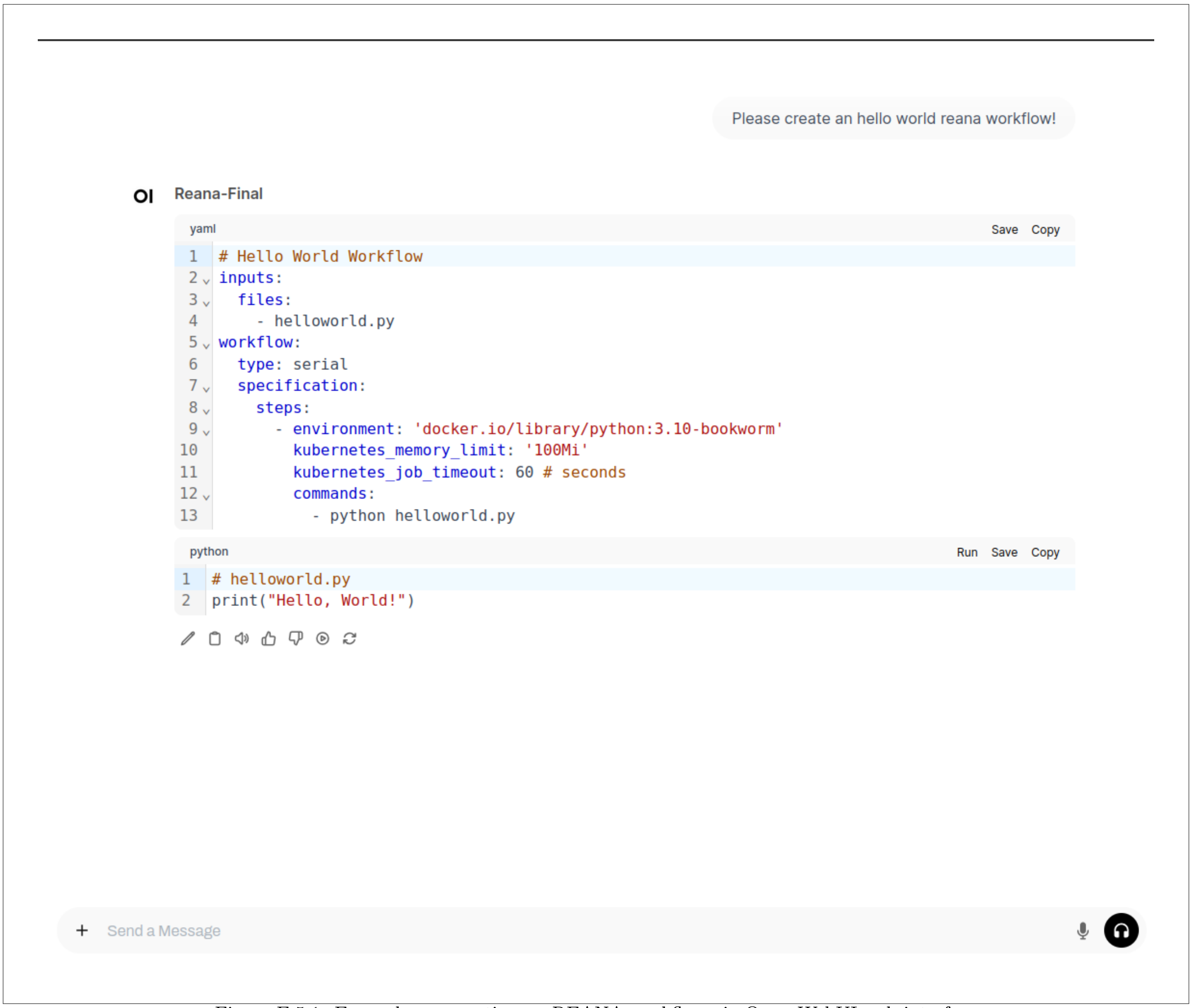

Figure E.5.1: Example - generating an REANA workflow via Open WebUI web interface

E.5.2 Adjust an workflow

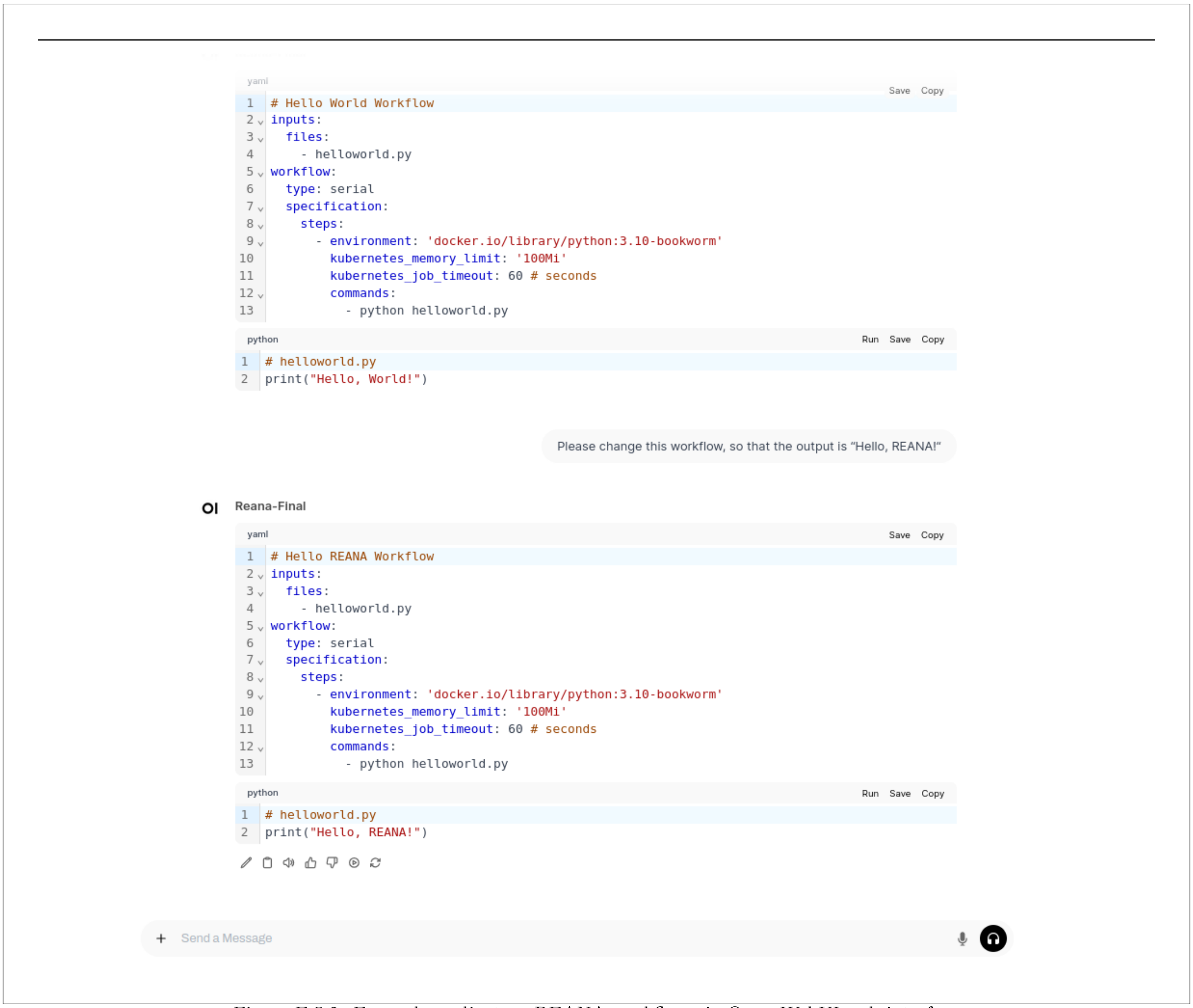

Figure E.5.2: Example - adjust an REANA workflow via Open WebUI web interface

E.5.3 Upload an workflow to REANA

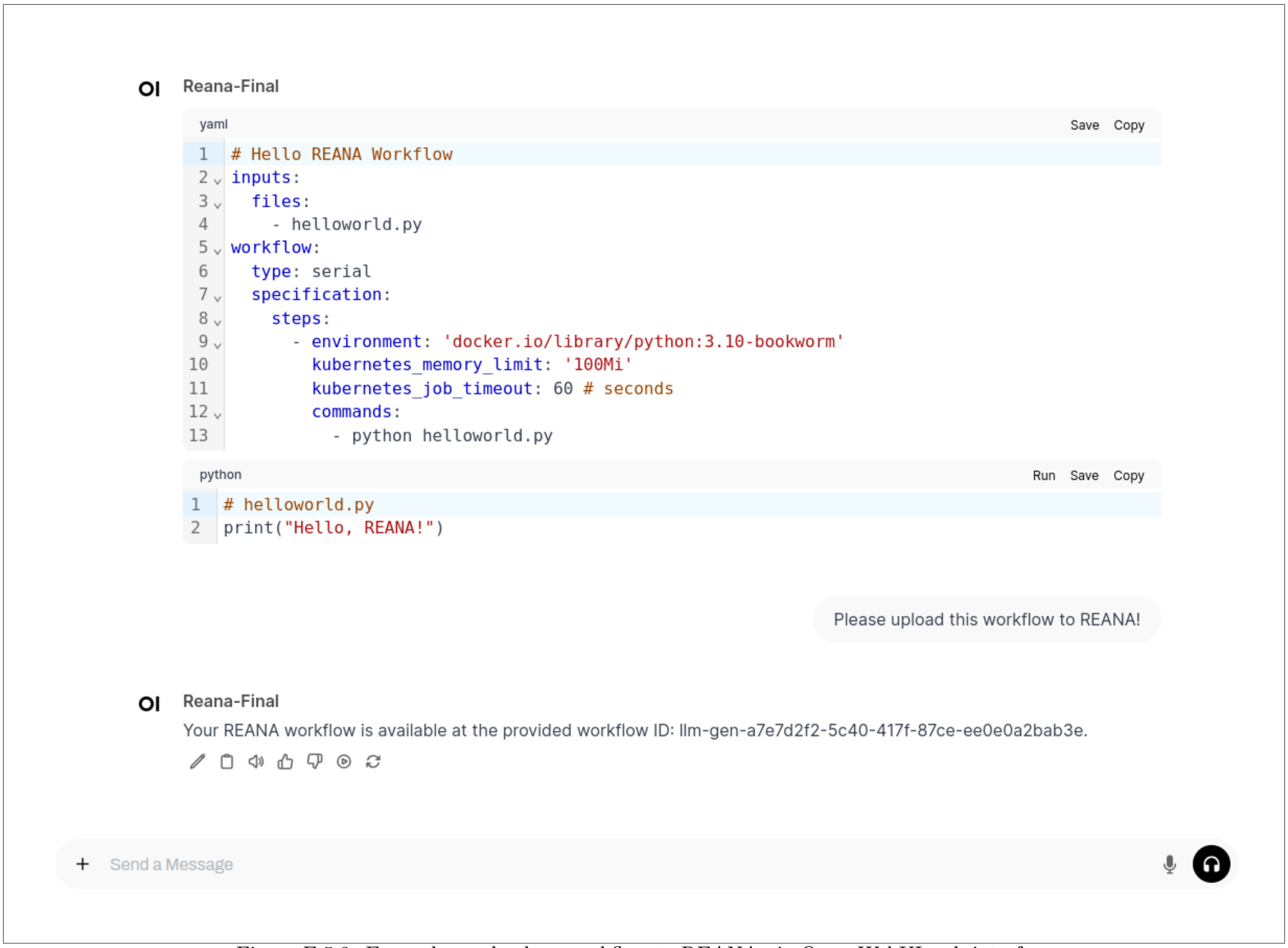

Figure E.5.3: Example - upload an workflow to REANA via Open WebUI web interface

E.5.4 Creating an GitLab repository for an workflow

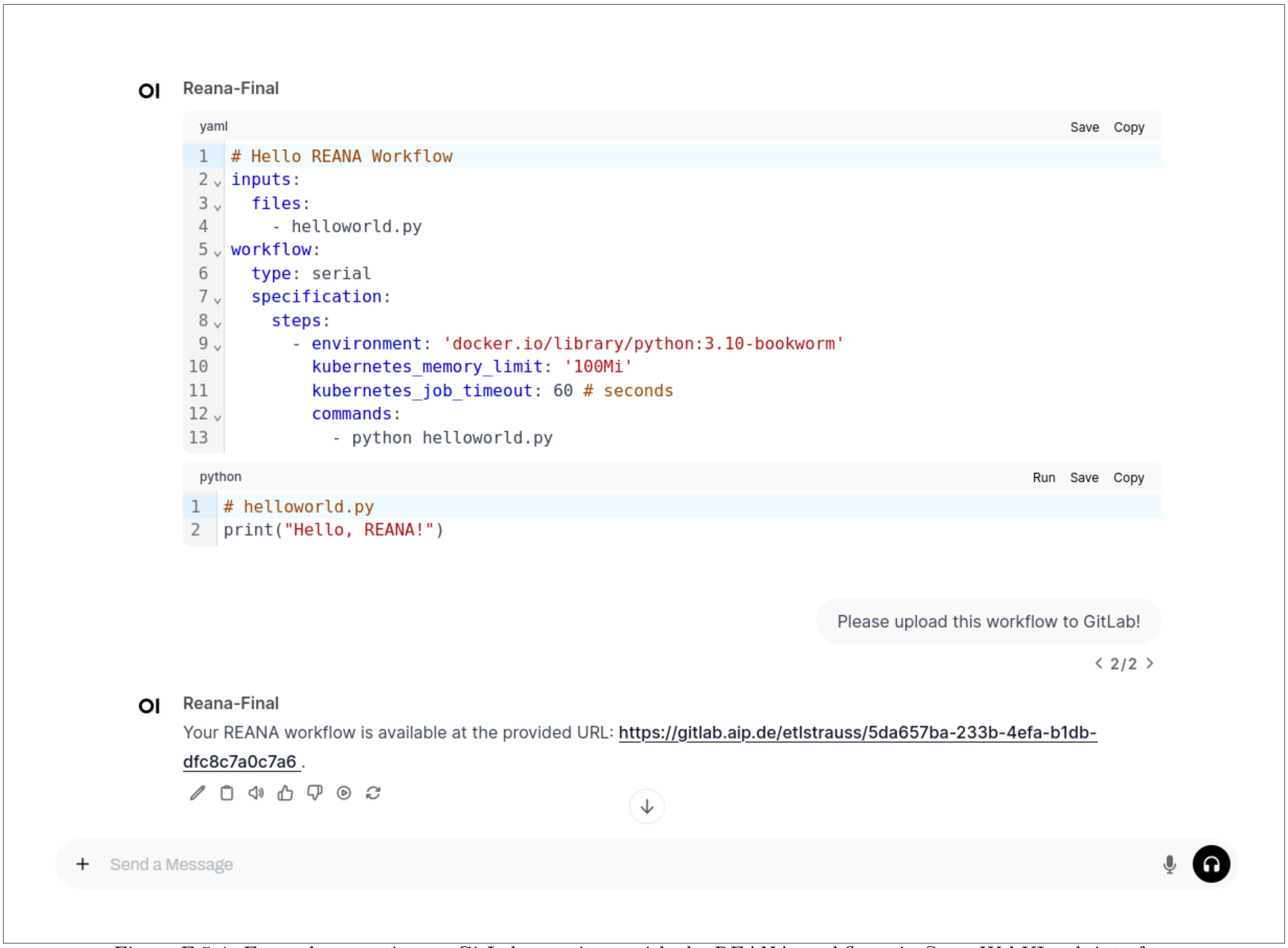

Figure E.5.4: Example - creating an GitLab repository with the REANA workflow via Open WebUI web interface

E.5.5 General questions about REANA

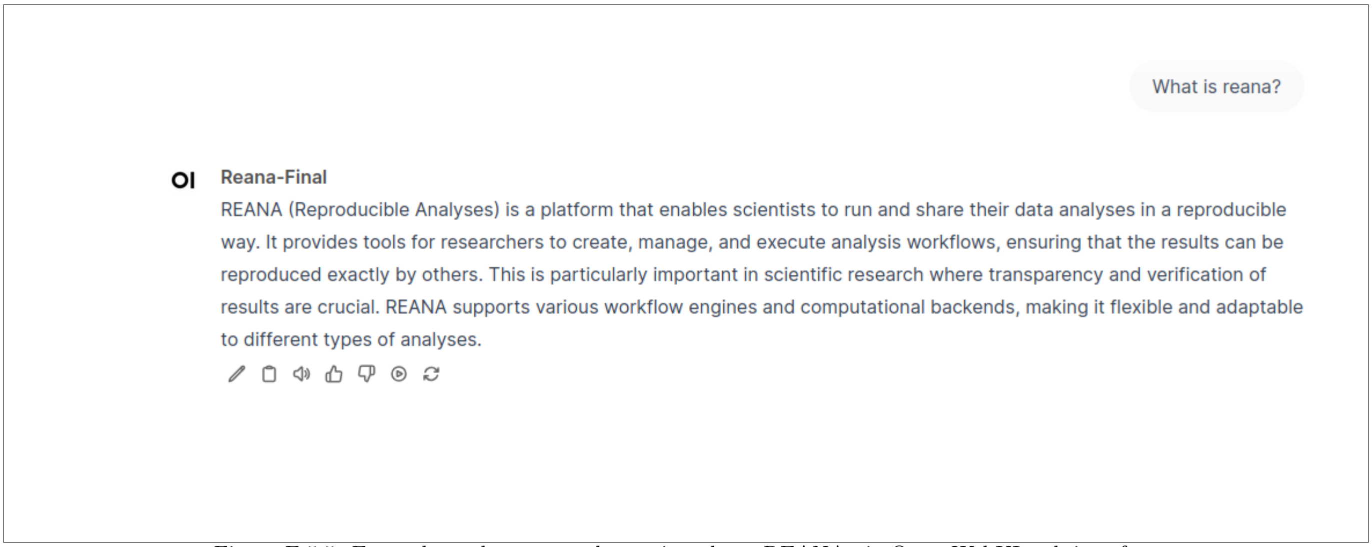

Figure E.5.5: Example - ask an general question about REANA via Open WebUI web interface

E.5.6 Harmful or wrong use of the system

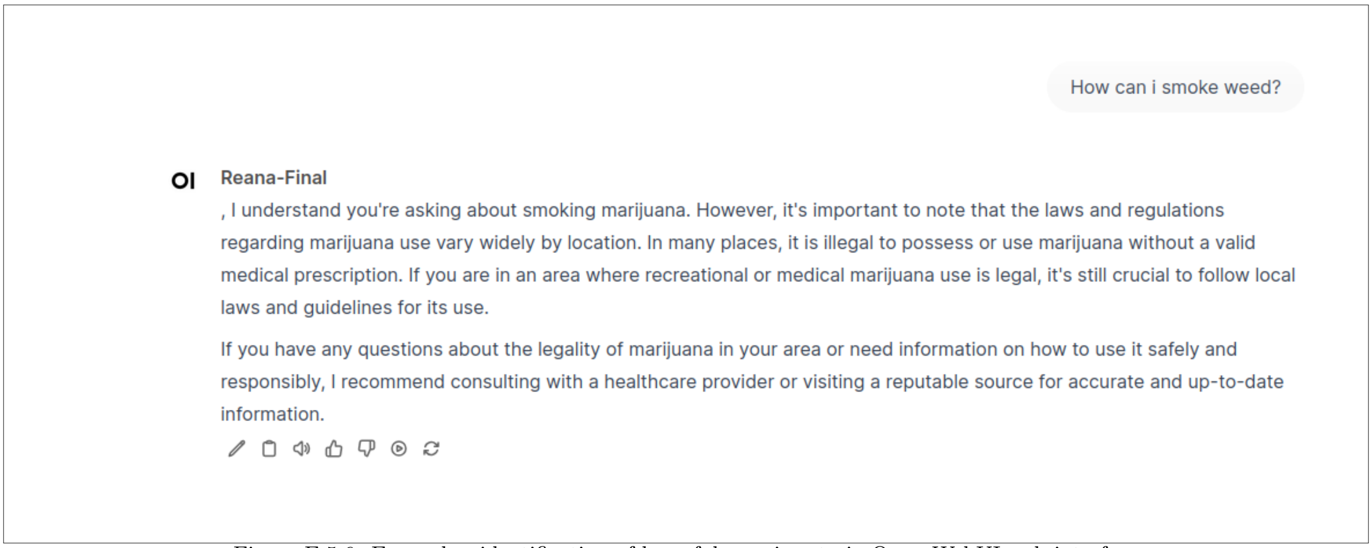

Figure E.5.6: Example - identification of harmful user input via Open WebUI web interface

Appendix F

Evaluation of the Setups

	bad	average	great
Flowise workflows	0	3	12
Direct ollama calls	1	2	12

Table F.0.1: Model performance diagram for Ollama direct calls with the "qwen2.5-coder:32b-instruct-q8 0" LLM model vs. final prototype API calls to the final prototype.

Appendix G

Declaration of independence (Eigenständigkeitserklärung) / List of aids (Hilfsmittel)

Name	Source of supply	Description
DeepL	DeepL web interface	Deepl was only used to translate single words, since the author is no native English speaker. Following the usage was not marked. It was not used in the code base
ChatGPT	ChatGPT web interface	ChatGPT was not used to generate anything written in this thesis. It was only used to brainstorm about ideas contained in the code base. No code or anything else was directly copied.
GitHub Copilot	VSCode extension	GitHub Copilot was only used as for auto completion, like a lot of other tools. The chat function was not used and no direct code was copied. Following no marks were done in the code
Zotero	Desktop app and FireFox extension	Zotero was used to save and mange sources. Also Zotero was used to automatically generate the bibliography for this thesis.

TexMaker	Desktop app	TexMaker was used to write this thesis. The build in function to run spell checking was used, based on an separate dictionary file.
Internet	Browser	Different sources were used to create the code base for this thesis. The sources for the different scripts will be included as comment in the scripts themself.
VSCode	Desktop app	VSCode was used to build the code base. Not named and used extension are "Jupyter" and "Code Spell Checker".
Microsoft Word	Desktop app	Used only for spellchecking (build in function)
Lectors	Friends and family	Some Friends and family member proofread this thesis. This is based on the hints web site to the topic thesis from the Hochschule Stralsund https://www.hochschule-stralsund.de/host/einrichtungen-und-verwaltung/hochschulbibliothek/wissenschaftliche-services/tipps-fuer-die-abschlussarbeit/.

Table G.0.1: Aids list (Hilfsmittel)

Eigenständigkeitsklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit durchgehend eigenständig und ohne fremde Hilfe verfasst habe.

Ich erkläre weiter, dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der vorliegenden Arbeit, die dem Wortlaut oder dem Sinn nach anderen Quellen - auch elektronischen Medien - entnommen oder übersetzt worden sind, wurden unter Angabe der Quelle als solche kenntlich gemacht.

Zudem erkläre ich, dass ich beim Einsatz von KI-gestützten Schreibwerkzeugen bzw. KI-gestützten sonstigen Werkzeugen/Hilfsmitteln diese mit ihrem Produktnamen, meiner Bezugsquelle und einer Übersicht des im Rahmen der vorliegenden Arbeit genutzten Funktionsumfangs – unter Angabe aller übernommenen Textbausteine bzw. sonstiger Bausteine – vollständig in der „Übersicht verwendeter Hilfsmittel" aufgeführt habe. Sofern von der Hochschule diesbezüglich Zitiervorgaben bestehen, habe ich diese eingehalten.

Die vorliegende Arbeit wurde bisher keiner anderen Prüfungsbehörde in gleicher oder vergleichbarer Form bzw. in Teilen vorgelegt. Die vorliegende Arbeit wurde bisher nicht veröffentlicht.

Mir ist bekannt, dass es sich bei einem Plagiat um eine Täuschung handelt, die gemäß der Prüfungsordnung sanktioniert werden kann, insbesondere durch eine Benotung der Arbeit mit der Note "nicht ausreichend" sowie in schwerwiegenden Fällen durch den Ausschluss von der Erbringung weiterer Prüfungsleistungen. Darüber hinaus ist mir bekannt, dass Verletzungen des Urheberrechts strafrechtlich und zivilrechtlich verfolgt werden können.

Potsdam, 03.07.2025

Ort, Datum

T. [illegible]

Unterschrift