LOCAL DEPLOYMENT OF A LIGHTWEIGHT LLM-BASED AI ASSISTANT USING OLLAMA AND DOCKER

Willard Soriano

Contents

Local Deployment of a Lightweight LLM-Based AI Assistant Using Ollama and Docker

Introduction

Artificial Intelligence (AI) has rapidly advanced in recent years, with Large Language Models (LLMs) emerging as one of the most impactful developments in the field. LLMs are designed to understand and generate human-like text by learning patterns from vast amounts of data, enabling applications such as chatbots, virtual assistants, and automated content generation. Traditionally, these models are accessed through cloud-based services; however, recent advancements in lightweight models and local deployment frameworks have made it possible to run LLMs directly on personal computers.

The purpose of this project is to design and deploy a localized AI assistant using a simple and lightweight LLM. The project demonstrates how open-source tools can be configured to host an AI assistant that operates entirely on a local machine. By following a step-by-step installation and configuration process, the project showcases both command-line and web-based interaction with the AI assistant.

Running AI assistants locally offers several important advantages. From a privacy perspective, user data remains on the local system and is not transmitted to external servers. In terms of performance, local execution can reduce latency by eliminating network dependency, while also allowing offline operation. Additionally, local deployment reduces long-term costs associated with subscription-based or usage-based cloud AI services, making it an accessible solution for educational and experimental use.

To achieve these objectives, the project utilizes a combination of open-source technologies. Ollama is used to manage and run the LLaMA 3.2 large language model locally. Docker is employed to containerize the user interface, while Open WebUI provides a browser-based platform for interacting with the AI assistant. Together, these tools form a complete system for deploying and demonstrating a private, local AI assistant.

Project Objectives

The primary objective of this project is to configure a local server environment capable of hosting an AI assistant that operates entirely on a personal computer. This includes setting up the necessary software components to ensure that the system runs reliably and efficiently without reliance on external cloud services. By establishing a local server setup, the project emphasizes privacy, control, and offline accessibility.

Another key objective is to install and execute a lightweight large language model locally. The LLaMA 3.2 model with 1.24 billion parameters was selected due to its balance between performance and resource efficiency, making it suitable for execution on consumer-grade hardware. Running the model locally demonstrates that meaningful AI interactions can be achieved without requiring high-end computing infrastructure.

The project also aims to create a user-friendly, web-based interface for interacting with the AI assistant. This is accomplished using Docker for containerization and Open WebUI for the front-end interface. Containerization ensures consistency and ease of deployment, while the web interface provides an intuitive way for users to communicate with the AI assistant through a browser.

Finally, the project seeks to demonstrate successful interaction with the AI assistant through both command-line and web-based environments. By verifying responses from the model and showcasing real-time conversations, the project confirms that the local AI assistant is fully functional and meets the requirements outlined in the assignment.

System Requirements

This section outlines the hardware and software requirements necessary to successfully implement and operate the local AI assistant. Clearly defining these requirements ensures that the system can run efficiently and that the deployment process can be replicated on similar machines. The requirements are divided into hardware and software components, covering the minimum resources needed to support the large language model, containerized services, and user interface.

Hardware Requirements

To successfully deploy and run a local AI assistant, the system must meet certain hardware requirements to ensure stable and reliable performance. The project is intentionally designed to operate on standard consumer-grade hardware, making it suitable for educational and experimental use without the need for specialized infrastructure.

Processing Requirements

- A modern multi-core CPU is sufficient for running the selected lightweight large language model.
- Example processors include:
    - Intel Core i5 or i7 series (e.g., Intel Core i5-4200U)
    - AMD Ryzen 5 or equivalent
- While a dedicated GPU is not mandatory, systems equipped with a compatible GPU may experience improved inference performance.
- Entry-level GPUs such as NVIDIA GeForce GTX or older mobile GPUs (e.g., NVIDIA GeForce GT 720M) can provide limited acceleration, though CPU-based execution remains the primary mode for this project.

Memory and Storage Requirements

- A minimum of **8 GB of RAM** is recommended to allow smooth execution of the LLM alongside supporting services such as Docker and the web interface.
- Systems with **12 GB or more RAM** can achieve more stable performance and reduced latency during inference.
- Adequate storage is required to accommodate language model files, Docker images, and related dependencies.
- Approximately **10–15 GB of free disk space** is sufficient for the complete setup.
- Using an SSD (e.g., a 256 GB SATA SSD) is recommended for faster model loading and overall system responsiveness, although HDD-based systems can still function with longer load times.

Software Requirements

The software environment for this project consists of several essential tools and platforms required to deploy and operate the local AI assistant. The system was implemented and tested on a desktop operating system with full support for containerization, networking, and command-line utilities.

Operating System

- A supported 64-bit operating system is required to host the local AI assistant.
- Compatible operating systems include:
    - Microsoft Windows 10 or later
    - macOS (Intel or Apple Silicon)
    - Linux distributions such as Ubuntu or Fedora
- The operating system must support Docker Desktop and command-line execution for proper system integration.

Core Software Components

- **Ollama**
    - Used as the primary runtime environment for managing and executing large language models locally.
    - Provides a simple command-line interface for downloading, listing, and running LLMs.
    - Enables efficient local inference without requiring complex configuration.
- **Docker**
    - Required to containerize the web-based user interface.
    - Ensures consistent deployment, isolation of services, and ease of management across different systems.
    - Docker Desktop is used to manage containers and verify runtime status.
- **Open WebUI**
    - Serves as the graphical, browser-based interface for interacting with the AI assistant.
    - Connects to the locally running LLM through Ollama.
    - Allows users to send prompts and receive responses through a user-friendly web interface.

Together, these software components form the foundation of the local AI assistant system. By combining local model execution with containerized web access, the project achieves a flexible, reproducible, and user-accessible AI deployment without reliance on cloud-based services.

Methodology and Implementation

The implementation process focuses on configuring existing open-source tools rather than developing custom application code. Each stage of the setup process is documented and verified through command-line outputs and screenshots to ensure transparency, reproducibility, and alignment with the project requirements.

All configuration commands, documentation, and screenshots referenced in this section are also maintained in a publicly accessible GitHub repository (see Appendix A).

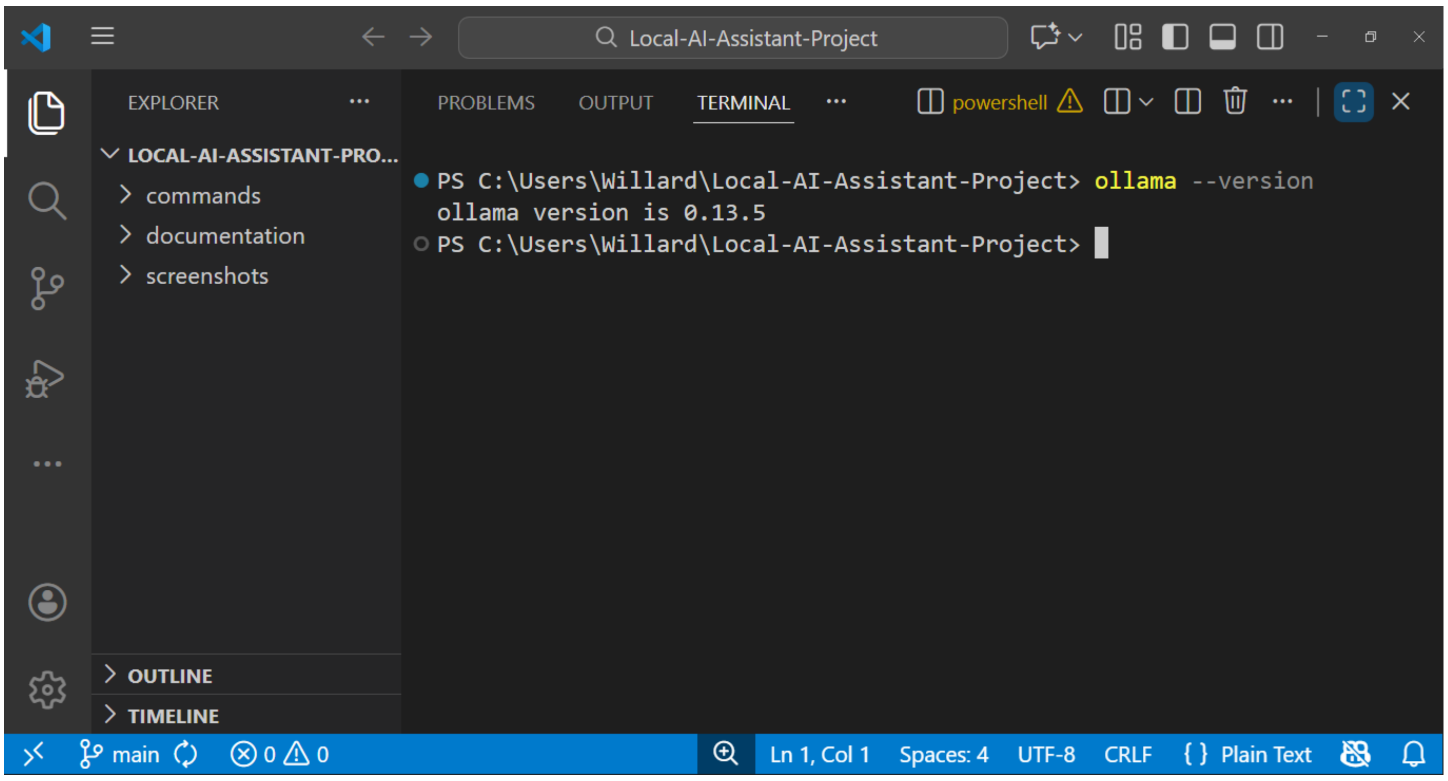
Installing Ollama

Ollama is an open-source tool designed to simplify the process of running large language models locally. It acts as a lightweight runtime environment that manages model downloads, execution, and interaction through a simple command-line interface. Ollama was selected for this project due to its ease of use and compatibility with lightweight LLMs.

To install Ollama, a structured installation process was followed to ensure proper setup and verification of the software. The steps are outlined below:

1. The official Ollama website was accessed using a web browser, and the installer corresponding to the host operating system was selected and downloaded.
2. The downloaded installer was executed, and the installation was completed by following the default setup instructions provided by the platform.
3. After installation, Ollama was launched to ensure that the service was running correctly in the background.
4. The installation was verified by executing a command-line version check, which confirmed that Ollama was successfully installed and properly configured for use.

Figure 1 shows the successful verification of the Ollama installation using the command-line interface.



```
PS C:\Users\Willard\Local-AI-Assistant-Project> ollama --version
ollama version is 0.13.5
PS C:\Users\Willard\Local-AI-Assistant-Project>
```

Figure 1 Ollama version verification confirming successful installation.

Downloading and Running the LLaMA 3.2 Model

The project utilizes the LLaMA 3.2 model with 1.24 billion parameters. This model was chosen because it offers a balance between performance and computational efficiency, making it suitable for execution on consumer-grade hardware without requiring a dedicated GPU. Despite its relatively small size compared to larger models, it is capable of generating coherent and meaningful responses.

The model was downloaded using Ollama's built-in model management functionality through a command-line instruction. To download and execute the large language model, Ollama's built-in model management features were used to ensure a straightforward and reliable setup. The process was completed as follows:

1. The LLaMA 3.2 model was downloaded using the following command, which retrieved the model files from the official Ollama repository:

```
ollama pull llama3.2
```

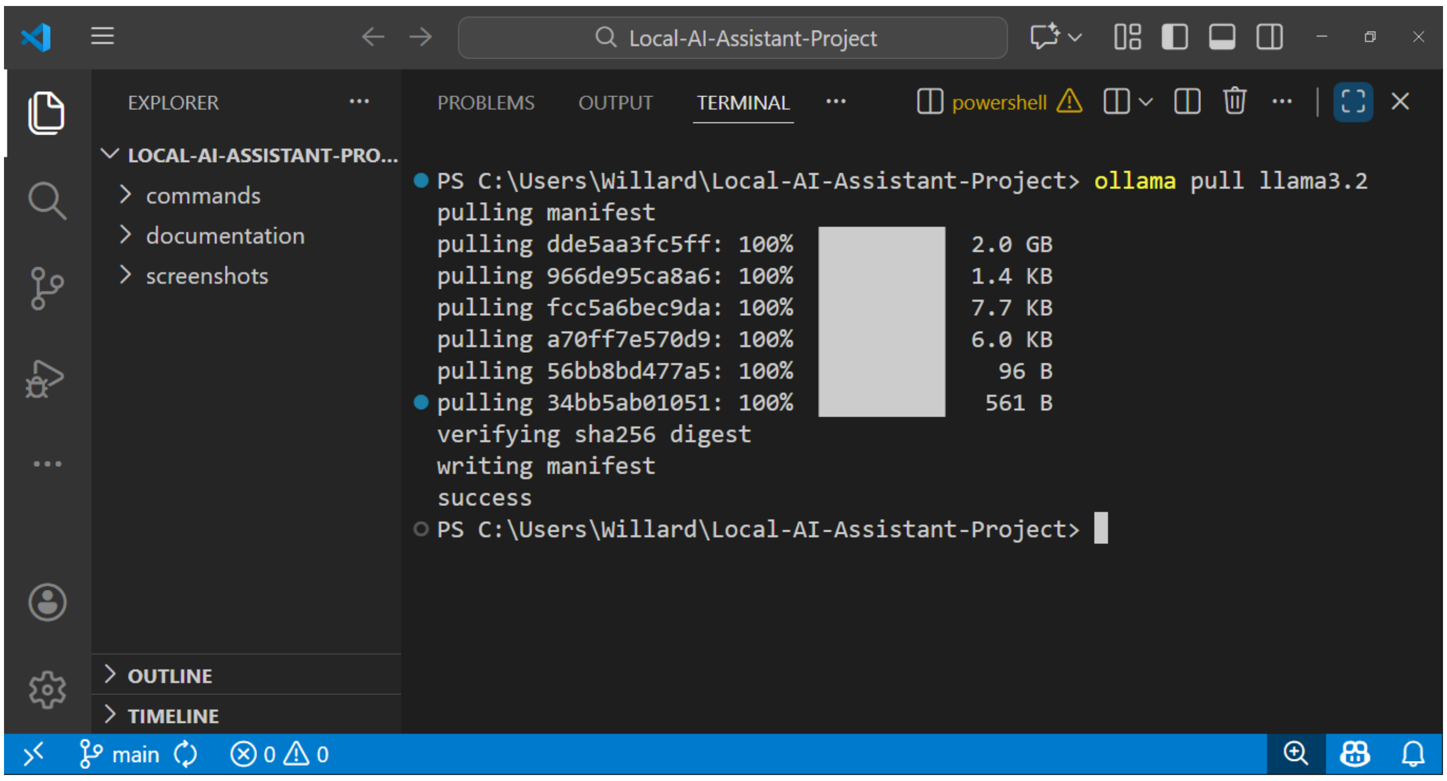
2. After the model download completed successfully, the model was executed locally using the command:

```
ollama run llama3.2
```

3. Once the model was running, test prompts were entered through the terminal to initiate a conversation with the AI assistant.

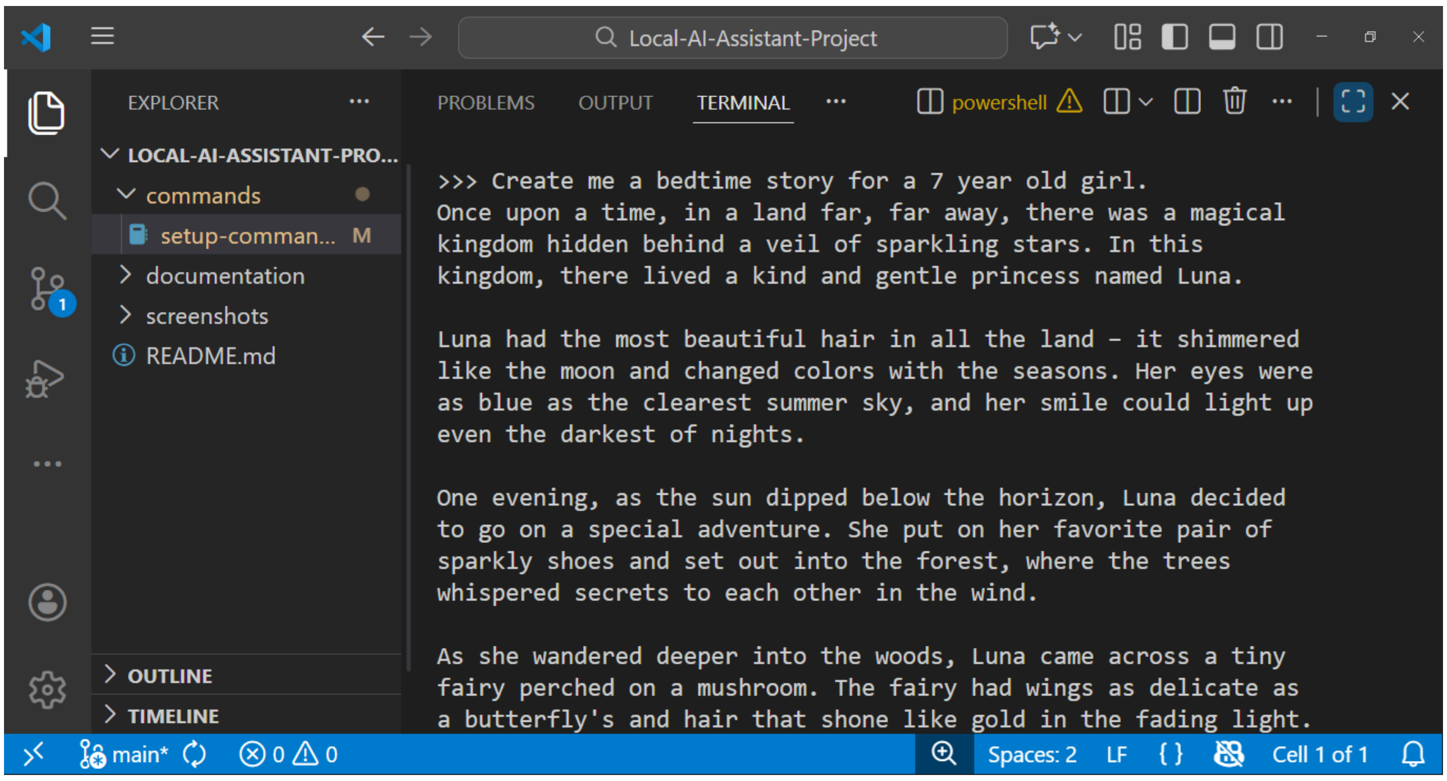
4. The responses generated by the model were observed to verify that the AI assistant was functioning correctly and producing real-time outputs.

Figure 2 illustrates the successful model download process, while **Figure 3** shows a terminal-based interaction with the LLaMA 3.2 model.



```
PS C:\Users\Willard\Local-AI-Assistant-Project> ollama pull llama3.2
pulling manifest
pulling dde5aa3fc5ff: 100%                2.0 GB
pulling 966de95ca8a6: 100%                1.4 KB
pulling fcc5a6bec9da: 100%                7.7 KB
pulling a70ff7e570d9: 100%                6.0 KB
pulling 56bb8bd477a5: 100%                  96 B
pulling 34bb5ab01051: 100%                 561 B
verifying sha256 digest
writing manifest
success
PS C:\Users\Willard\Local-AI-Assistant-Project>
```

Figure 2 Successful download of the LLaMA 3.2 model using Ollama.



```
>>> Create me a bedtime story for a 7 year old girl.
Once upon a time, in a land far, far away, there was a magical
kingdom hidden behind a veil of sparkling stars. In this
kingdom, there lived a kind and gentle princess named Luna.

Luna had the most beautiful hair in all the land – it shimmered
like the moon and changed colors with the seasons. Her eyes were
as blue as the clearest summer sky, and her smile could light up
even the darkest of nights.

One evening, as the sun dipped below the horizon, Luna decided
to go on a special adventure. She put on her favorite pair of
sparkly shoes and set out into the forest, where the trees
whispered secrets to each other in the wind.

As she wandered deeper into the woods, Luna came across a tiny
fairy perched on a mushroom. The fairy had wings as delicate as
a butterfly's and hair that shone like gold in the fading light.
```

Figure 3 Command-line interaction demonstrating a live conversation with the LLaMA 3.2 model.

Installing Docker

Docker is used in this project to containerize the web-based user interface, allowing it to run independently of the host system's environment. Containerization ensures consistent deployment and simplifies the integration between the AI model and the user interface.

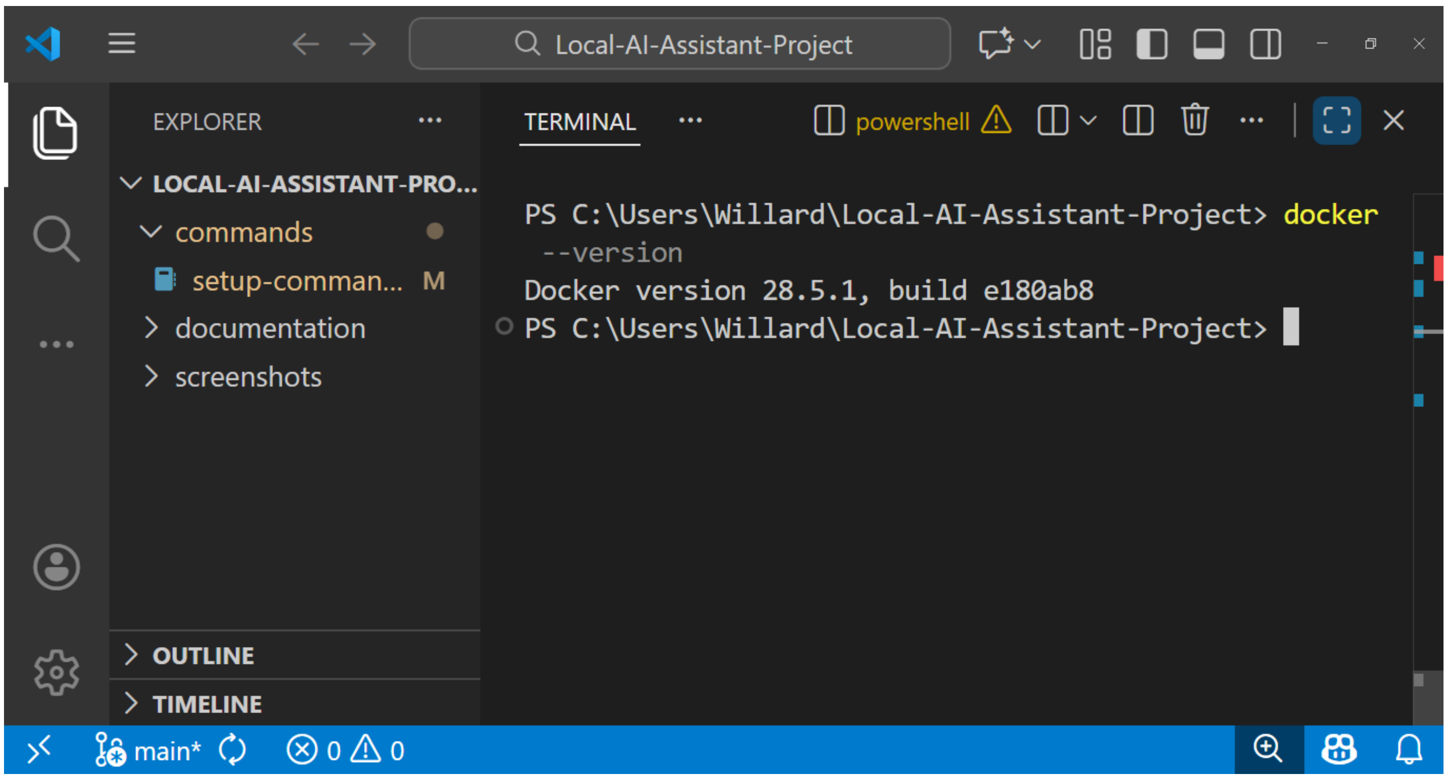
To install Docker, a structured installation and verification process was followed to ensure that the containerization platform was correctly configured. The steps are outlined below:

1. The official Docker website was accessed, and Docker Desktop was downloaded for the appropriate host operating system.
2. The Docker Desktop installer was executed, and the installation was completed by following the default setup instructions.
3. After installation, Docker Desktop was launched to ensure that the Docker engine was running properly in the background.
4. The installation was verified using the following command-line instruction:

```
docker --version
```

5. The displayed version information confirmed that Docker was successfully installed and operational.

Figure 4 presents the Docker version verification, confirming a successful installation.



```
PS C:\Users\Willard\Local-AI-Assistant-Project> docker
 --version
Docker version 28.5.1, build e180ab8
PS C:\Users\Willard\Local-AI-Assistant-Project>
```

Figure 4 Docker version output confirming successful installation.

Setting Up Open WebUI Using Docker

Open WebUI is a browser-based interface that enables users to interact with large language models in a user-friendly manner. In this project, Open WebUI serves as the front-end interface, allowing prompts to be sent to the locally running LLaMA model through Ollama.

To set up the web-based interface, Open WebUI was deployed using Docker and configured to communicate with the locally running Ollama service. The deployment process was carried out as follows:

1. A Docker container for Open WebUI was launched using a command that specifies port mapping and an environment variable to connect the interface to the Ollama runtime:

```
docker run -d -p 3000:8080 -e OLLAMA_BASE_URL=http://host.docker.internal:11434 --
   name open-webui ghcr.io/open-webui/open-webui:main
```

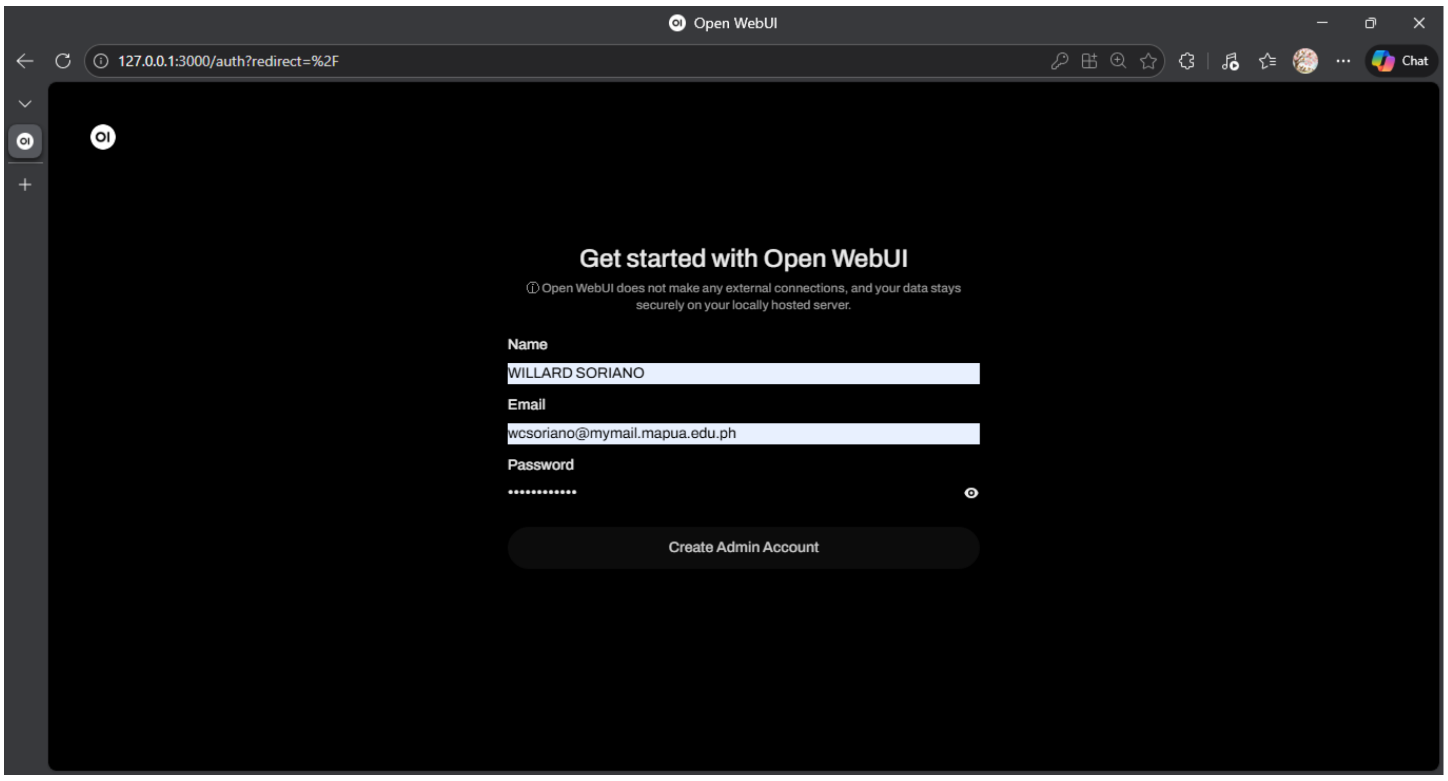
2. This command pulled the Open WebUI container image and started the service in detached mode, allowing it to run in the background.

3. Once the container was running, the Open WebUI interface was accessed through a web browser using the local host address:

```
http://127.0.0.1:3000/
```

4. Successful deployment was confirmed when the Open WebUI dashboard loaded correctly and displayed the locally available language model.

Figure 5 shows the Open WebUI login screen, **Figure 6** displays the model selection interface, and **Figure 7** presents a sample prompt interaction through the web interface.



Get started with Open WebUI

ⓘ Open WebUI does not make any external connections, and your data stays securely on your locally hosted server.

Name: WILLARD SORIANO

Email: wcsoriano@mymail.mapua.edu.ph

Password: ••••••••••••

Figure 5 Open WebUI login and initial setup screen.

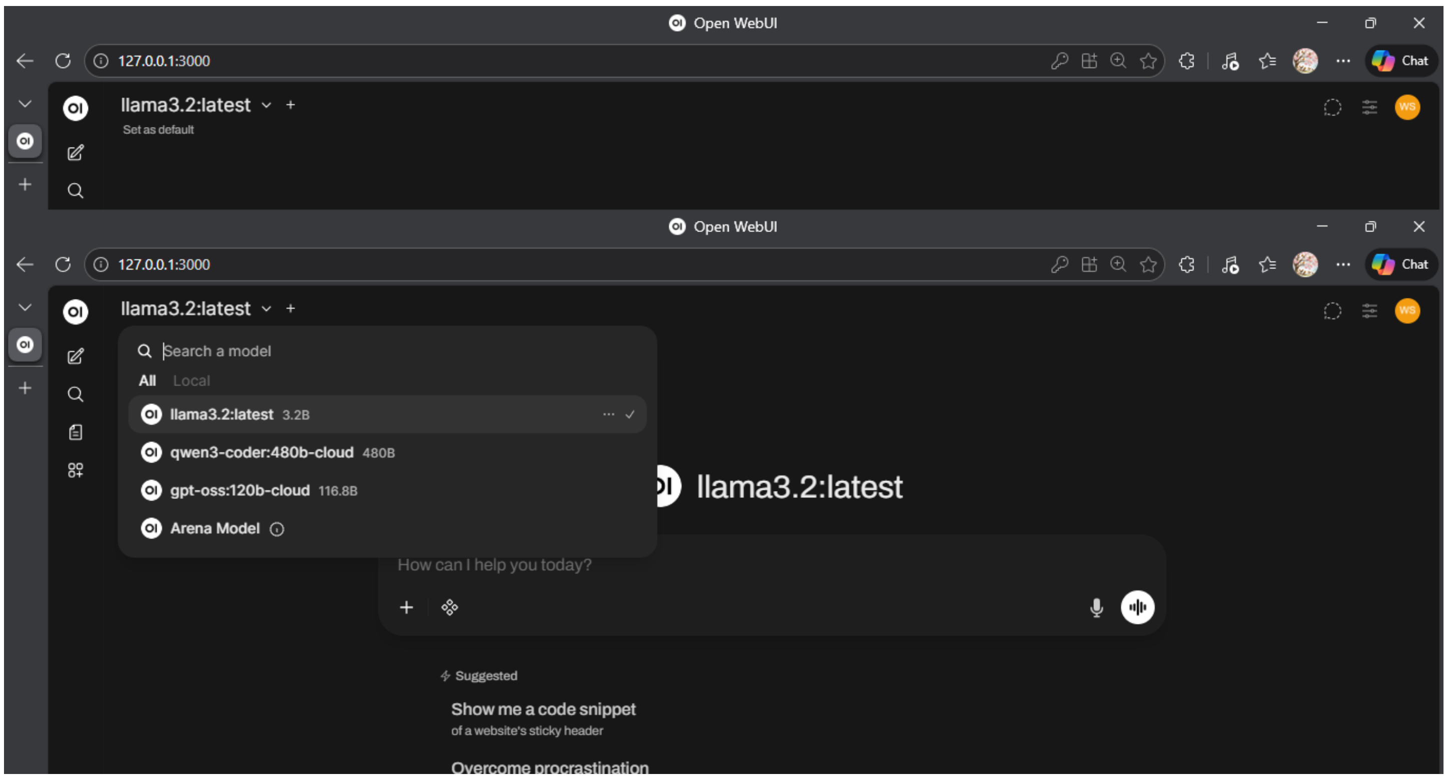
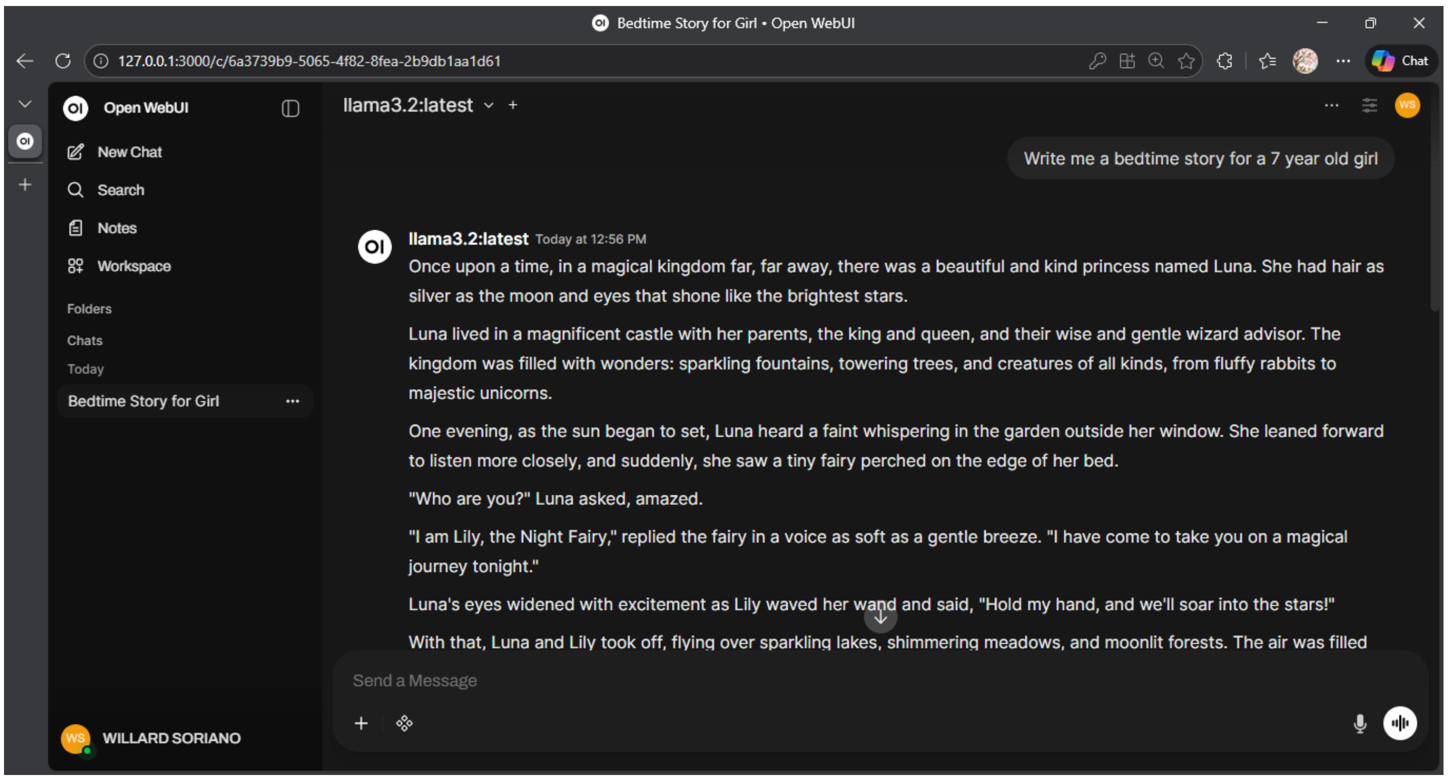


Figure 6 Open WebUI model selection showing the locally available LLaMA 3.2 model.



Write me a bedtime story for a 7 year old girl

llama3.2:latest Today at 12:56 PM

Once upon a time, in a magical kingdom far, far away, there was a beautiful and kind princess named Luna. She had hair as silver as the moon and eyes that shone like the brightest stars.

Luna lived in a magnificent castle with her parents, the king and queen, and their wise and gentle wizard advisor. The kingdom was filled with wonders: sparkling fountains, towering trees, and creatures of all kinds, from fluffy rabbits to majestic unicorns.

One evening, as the sun began to set, Luna heard a faint whispering in the garden outside her window. She leaned forward to listen more closely, and suddenly, she saw a tiny fairy perched on the edge of her bed.

"Who are you?" Luna asked, amazed.

"I am Lily, the Night Fairy," replied the fairy in a voice as soft as a gentle breeze. "I have come to take you on a magical journey tonight."

Luna's eyes widened with excitement as Lily waved her wand and said, "Hold my hand, and we'll soar into the stars!"

With that, Luna and Lily took off, flying over sparkling lakes, shimmering meadows, and moonlit forests. The air was filled

Figure 7 Sample prompt and response displayed in the Open WebUI interface.

Testing the Local AI Assistant

After completing the setup, the local AI assistant was tested to verify its functionality across both the command-line interface and the web-based Open WebUI. Prompts were sent through the Open WebUI interface, and the responses generated by the LLaMA 3.2 model were observed to confirm that the interface was properly connected to Ollama and that the model was processing user input as expected.

During testing, it was observed that interactions through the command-line interface were noticeably faster than those performed through Open WebUI. The web-based interface exhibited slower response times, likely due to the additional overhead introduced by containerization and the browser-based communication layer. Despite this difference in speed, the quality and content of the responses generated by the model remained largely consistent across both interfaces.

To further evaluate response consistency, identical prompts were issued through both the CLI and the web interface. The AI assistant produced similar narrative structure, tone, and level of detail in its outputs, indicating that the underlying model behavior remained unchanged regardless of the interaction method. This demonstrated that Open WebUI serves purely as an interface layer and does not alter the model's response generation.

The successful exchange of prompts and responses across both interfaces confirmed that the local AI assistant was fully operational. This testing phase validated the correct integration of all system components and demonstrated that the project met its intended objectives, while also highlighting performance differences between direct CLI interaction and web-based access.

Results and Observations

The performance of the local AI assistant was evaluated on a consumer-grade laptop equipped with an Intel Core i5-4200U processor, 12 GB of RAM, and a combination of SSD and HDD storage. Despite the relatively older hardware and lack of a high-performance GPU, the local AI assistant operated reliably using the lightweight LLaMA 3.2 model. The system was able to start the model successfully, accept user prompts, and generate coherent responses through both the command-line interface and the web-based Open WebUI.

In terms of response quality and speed, the AI assistant produced relevant and contextually appropriate answers for general-purpose queries. While response times were not instantaneous compared to cloud-based AI services, the delays were reasonable and acceptable for educational and demonstration purposes. The response speed was noticeably influenced by the CPU-based inference process, but once the model was loaded into memory, subsequent interactions became more consistent.

Resource usage was moderate and aligned with expectations for local LLM execution on older hardware. CPU utilization increased significantly during model inference, while memory usage remained within the available 12 GB of RAM. The use of an SSD contributed positively to model loading times, whereas the dedicated GPU was not actively utilized for inference. The presence of additional cloud-based models in the Ollama environment did not impact local system performance, as these models are accessed remotely and do not rely on local computational resources.

Challenges and Solutions

During the implementation of the local AI assistant, a few challenges were encountered that required troubleshooting and adjustment. One of the initial challenges involved the use of incorrect or incomplete command-line prompts when interacting with Ollama. In some instances, commands such as version checks or model execution failed due to the Ollama executable not being immediately recognized by the system or due to incorrect model naming conventions. These issues were resolved by ensuring that Ollama was properly installed, available in the system path, and by using the correct model identifiers as listed through Ollama's model management commands.

Another challenge arose during the process of downloading and running the LLaMA 3.2 model. Early attempts to execute the model resulted in manifest-related errors, which were caused by incomplete or incorrect model pull commands. This issue was addressed by re-running the correct model pull command and verifying the successful download using the model listing functionality. Once the model was fully downloaded and verified, it was able to run successfully and respond to user prompts as expected.

The deployment of Open WebUI using Docker also required careful verification. While the container initially appeared to start correctly, additional checks were necessary to confirm that the service was fully operational and healthy. By inspecting running containers and reviewing container logs, it was possible to verify that Open WebUI had initialized correctly and was successfully connected to the locally running Ollama service. This step emphasized the importance of log inspection when working with containerized applications.

A notable limitation encountered during the project was the inability to run larger-scale models, such as those with 32 billion parameters, on the available hardware. Due to constraints related to CPU performance, memory capacity, and lack of high-end GPU support, such models were not feasible for local execution on the test system. This limitation highlights a broader challenge in local AI deployment, where model size and hardware accessibility must be carefully balanced. As hardware becomes more powerful and optimization techniques continue to improve, it is anticipated that running larger models locally will become more accessible in the future.

These challenges provided valuable insight into practical considerations when deploying local AI systems, reinforcing the importance of correct configuration, resource awareness, and systematic troubleshooting.

Security and Privacy Considerations

Security and privacy are critical considerations in the deployment of AI systems, particularly when handling user-generated data. One of the primary advantages of deploying an AI assistant locally is the increased level of control over data and system behavior. Since the model runs entirely on the host machine, users retain full ownership of their interactions and are not dependent on third-party servers for processing or storage.

A significant data privacy benefit of local AI deployment is that user prompts and responses remain on the local system. Unlike cloud-based AI services, where data may be transmitted, logged, or stored externally, local execution minimizes the risk of data exposure or unauthorized access. This is especially important when handling sensitive or personal information, as it reduces reliance on external data-handling policies and network security measures.

When compared to cloud-based AI solutions, local AI assistants offer stronger privacy guarantees but may involve trade-offs in performance and scalability. Cloud-based models typically benefit from powerful hardware, optimized infrastructure, and the ability to run very large models with minimal latency. However, they often require continuous internet access and may involve subscription costs or usage limitations. In contrast, local AI solutions prioritize privacy, offline availability, and user control, making them well-suited for educational, experimental, and privacy-conscious applications. This project demonstrates that, despite hardware limitations, local AI deployment can provide a secure and practical alternative to cloud-based AI systems.

Conclusion

This project successfully demonstrated the design and deployment of a local AI assistant using a lightweight large language model and open-source tools. By configuring a local server environment, installing and running the LLaMA 3.2 model through Ollama, and integrating a web-based interface using Docker and Open WebUI, the project achieved its objective of creating a functional, privacy-focused AI assistant that operates entirely on a personal computer.

Several key learning outcomes emerged from this project. The implementation process provided practical experience with local AI deployment, command-line operations, containerization, and system configuration. It also highlighted the importance of accurate command usage, model selection based on hardware capabilities, and effective troubleshooting through log inspection and verification commands. These skills are directly applicable to real-world system administration and AI deployment scenarios.

Local AI assistants have a wide range of practical applications, particularly in environments where privacy, offline access, or cost efficiency is a priority. Potential use cases include personal productivity tools, educational assistants, research support systems, and private data analysis environments. By running AI locally, users can maintain greater control over their data while still benefiting from intelligent automation and natural language interaction.

While the project met its intended goals, there are opportunities for future improvement and scalability. Upgrading hardware, particularly CPU and GPU resources, would allow for the execution of larger and more capable models. Additional enhancements could include fine-tuning models for domain-specific tasks, integrating persistent memory or knowledge bases, and improving user interface features. As AI hardware and optimization techniques continue to evolve, local AI assistants are expected to become increasingly powerful and accessible, further expanding their potential applications.

References

Appendix A: Project Repository

The complete project repository for this study, including setup commands, documentation files, and screenshots used throughout the implementation and testing process, is publicly available on GitHub.

Repository Link: https://github.com/willardcsoriano/Local-AI-Assistant-Project/tree/main