

# **MMI712 Project Phase 1 Report**

Okan Berhoglu 2739050

January 11, 2026

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Methodology</b>	<b>4</b>
2.1	Application Architecture . . . . .	4
2.2	Model Integration . . . . .	4
2.3	Containerization . . . . .	4
<b>3</b>	<b>Usage</b>	<b>5</b>
<b>4</b>	<b>Results</b>	<b>5</b>
<b>5</b>	<b>Conclusion</b>	<b>7</b>

## Abstract

*This report describes the implementation of a Visual Question Answering (VQA) system using the Moondream model. The application is containerized with Docker to ensure easy deployment and reproducibility. A command-line interface (CLI) allows users to ask questions about images. The project demonstrates how to run efficient vision-language models on local hardware.*

# 1 Introduction

Recent advancements in Artificial Intelligence have led to the development of powerful Vision Language Models (VLMs) capable of understanding and generating text based on visual inputs. Models such as GPT-4V and Gemini have demonstrated remarkable capabilities in tasks ranging from image captioning to complex visual reasoning. Despite their performance, these state-of-the-art models often present significant challenges for practical deployment. Many are proprietary, limiting accessibility and research transparency. Furthermore, their substantial computational requirements make them unsuitable for resource-constrained environments, such as edge devices or local applications where latency and privacy are critical concerns. This necessitates the exploration of efficient, open-source alternatives that balance performance with resource utilization.

Moondream emerges as a compelling solution in this landscape. It is a small, open-source vision language model designed to run efficiently on consumer-grade hardware and even mobile devices. By leveraging a compact architecture, Moondream significantly reduces the computational overhead associated with traditional VLMs without sacrificing essential capabilities. Consequently, Moondream offers a practical pathway for deploying visual understanding capabilities in real-world scenarios where resource efficiency and accessibility are paramount.

To ensure reproducibility and streamline the deployment process, Docker is utilized to containerize the application. Docker provides a lightweight and portable environment that encapsulates the application and its dependencies, eliminating the "it works on my machine" problem. By defining the runtime environment in a Dockerfile, we ensure that the model behaves consistently across different computing platforms. This approach simplifies the setup process for other researchers and developers, allowing them to easily deploy and experiment with software applications or models without worrying about conflicting libraries or system configurations.

In this work, the Moondream model is implemented for Visual Question Answering (VQA) tasks. The model processes natural language questions about input images and generates corresponding natural language answers. To ensure consistency and ease of deployment, the application is containerized using Docker. The implementation is validated using a set of test images and questions. The application is a web application that developed by using Streamlit.

This report is structured as follows; after the introduction, the methodology section describes the technical details of the implementation, an example usage of the application is given in the results section and the usage of the application is given in the usage section. Lastly the conclusion section provides a summary of the work done.

## 2 Methodology

The application is transformed into a web-based Visual Question Answering (VQA) system using the Streamlit framework. This shift from a command-line interface to a web application significantly enhances user interaction and accessibility. The core of the system remains the Moondream model, but the surrounding architecture is redesigned to support stateful user sessions, image management, and a responsive user interface.

### 2.1 Application Architecture

The application follows a modular structure, separating the user interface, business logic, and model interaction. The entry point is the `main.py` script, which serves as the main entry point for the Streamlit application. It initializes the session state, manages the navigation between different pages, and ensures the Moondream model is loaded. It directs the user to either the Main Page or the Chat Page based on the application state.

The User Interface, located in the `ui/` directory, is built using Streamlit's component-based design. It consists of the `MainPage` and `ChatPage` components. The `MainPage` handles the initial user landing, providing functionality for users to upload new images to start a chat session and displaying a history of recent chat sessions. The `ChatPage` manages the active chat interface, displaying the uploaded image, rendering the chat history (questions and answers), and providing an input field for new queries to interact with the model effectively.

To maintain clean code and separation of concerns, service modules are implemented in the `services/` directory. The `ChatService` manages the storage and retrieval of chat history, creating new sessions, and generating chat names. The `ImageService` handles image file operations, including saving uploaded images and loading them for display or processing.

### 2.2 Model Integration

The `Model` class in `model/model.py` encapsulates the interaction with the Moondream model. It is responsible for conditionally loading the model on GPU or CPU based on hardware availability, processing the input image into embeddings that the model can understand, and taking a natural language question and the encoded image to generate a text response during inference.

A dedicated script, `model_installer.py`, is used to download the specific revision of the Moondream model from the Hugging Face hub. This ensures that the correct model files are present before the application starts, which is particularly important for the Dockerized environment.

### 2.3 Containerization

The application is containerized using Docker to ensure a consistent execution environment. The `Dockerfile` is updated to support the web application. It uses `python:3.10-slim` as a base image for a lightweight footprint and installs system dependencies and Python packages listed in `requirements.txt`, which now includes `streamlit`.

A crucial step in the methodology is the execution of `model_installer.py` during the Docker build phase. This bakes the large model weights into the Docker image, preventing the need to download them every time the container starts.

Finally, the container exposes port 8501, the default for Streamlit, and uses `streamlit run main.py` as the entry point command.

## 3 Usage

The resulting image is published in the docker hub. The application can be found in this link: [https://hub.docker.com/r/okanberhoglu/chat\\_moon\\_vlm](https://hub.docker.com/r/okanberhoglu/chat_moon_vlm)

After downloading the image, the application can be run using the following command:

```
docker run -p 8501:8501 okanberhoglu/chat_moon_vlm:latest
```

The given command runs the Docker image in a container and maps the host's port 8501 to the container's port 8501. This allows the user to access the web application by navigating to `http://localhost:8501` in their web browser. The Docker image "okanberhoglu/chat\_moon\_vlm:latest" encapsulates all required dependencies, the Streamlit server, and the inference environment. Once the container is running, the application interface will be available for uploading images and asking questions.

## 4 Results

An example usage of the application and the results are given in this section. Firstly the test input image is given in the figure 1.



Figure 1: Example input image.

After running the application with the command given in the section 3, the application can be used as shown in the figures 2 and 3.

```
okanberhoglu@romer-OMEN-by-HP-40L-Gaming-Desktop-GT21-2xxx:~/Desktop/model_densenet$ docker run -it --rm -v ./images:/data:ro okanberhoglu/mm1712_project_mondream:latest --image_path /data/test_image.png
tokenizer.json: 3.69MB [00:00, 43.9MB/s]
Ask questions about your image.
Question: What is in the image?
Answer: The image features a man sitting at a desk in a modern office, working on a tablet while surrounded by various technological equipment. He is positioned in front of a computer monitor displaying code or programming information.

On the desk, there are multiple keyboards, a mouse, and a remote control. Additionally, there are two robots, one closer to the man and the other further away, positioned near the desk.

Question: How many people are in the image?
Answer: There is one person in the image, a man with a beard.
```

Figure 2: Example usage.

```
Question: What color is the t-shirt the man is wearing?  
Answer: The man is wearing a gray t-shirt.  
  
Question: What color is the hoodie the man is wearing?  
Answer: The man is wearing a brown hoodie.  
  
Question: Is there any monitor in the image?  
Answer: Yes, there is a large computer monitor in the image.  
  
Question: What does the monitor show?  
Answer: The monitor displays code or programming information.  
  
Question: What components the robot in the image has?  
Answer: The robot in the image has wheels, a camera, and a sensor.  
  
Question: What is the name of the robot?  
Answer: The robot's name is AgileX Scout Mini.
```

Figure 3: Example usage with questions.

The application is executed with another test image which is given in the figure 4 as well. The results are given in the figure 5.



Figure 4: Example input image.

```
okan@okan-Predator-PT315-52:~/Desktop$ sudo docker run -it --rm -v ./images:/data
:ro okanberhoglu/mni712_project_moondream:latest --image_path /data/test_image_tw
o.png
tokenizer.json: 3.69MB [00:00, 14.8MB/s]
Ask questions about your image.
Question: What is in the image?
Answer: The image features a wolf and a bison standing in a snowy field, facing each other. The wolf is positioned on the left side of the image, while the bison is on the right side.

Question: What is the wolf doing?
Answer: The wolf is jumping or leaping towards the bison in the snowy field.

Question: What season does it appear to be?
Answer: It appears to be winter, as the scene takes place in a snowy forest.

Question: What might happen in the next few seconds?
Answer: In the image, a wolf is chasing a bison in the snow, possibly attempting to catch it or engage in a confrontation. The wolf's aggressive posture and focused gaze suggest that it might be attempting to catch the bison or engage in a display of dominance. The outcome of this interaction could depend on the speed and agility of both animals, as well as the strength of their fangs or teeth.
```

Figure 5: Example usage with other test image.

## 5 Conclusion

In this project, the Moondream vision language model was successfully containerized using Docker. A Dockerfile was created to automate the installation of dependencies and the model itself, ensuring a consistent and reproducible environment. A Python script was developed to interface with the model, allowing users to perform Visual Question Answering tasks via a command-line interface. This implementation simplifies the deployment process and demonstrates the feasibility of running efficient VLMs on local hardware.

## References

- [1] Moondream.ai. *Moondream: A tiny vision language model*. Available at: <https://moondream.ai/> (Accessed: January 11, 2026).
- [2] Hugging Face. *Hugging Face: The AI community building the future*. Available at: <https://huggingface.co/> (Accessed: January 11, 2026).
- [3] Docker. *Docker Documentation*. Available at: <https://docs.docker.com/> (Accessed: January 11, 2026).
- [4] Streamlit. *Streamlit: A faster way to build and share data apps*. Available at: <https://streamlit.io/> (Accessed: January 11, 2026).