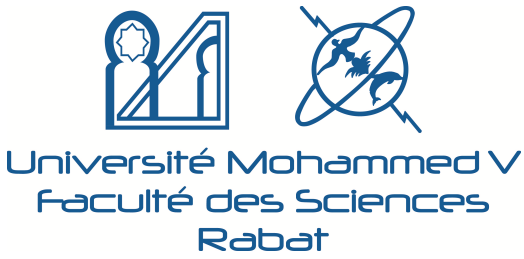
UNIVERSITÉ MOHAMMED V DE RABAT
Faculté des Sciences



Laboratoire (LRIT)
Master Informatique et Télécommunications

Project in Machine Learning and Deep Learning

Text-to-Image Application

Presented by :
Marwa Haoudi & Hanane Saidi

Defended on May 20, 2025 before the Jury

Prof. Abdelhak Mahmoudi Professor at the Faculty of Sciences - Rabat

Academic Year 2024–2025

Table des matières

Introduction

Generative artificial intelligence has made remarkable progress in recent years, enabling computers to create realistic images, text, and other media from simple user inputs. One exciting application of this technology is text-to-image generation, where an AI model creates images based on descriptive textual prompts. This project aims to develop a simple and user-friendly web application that allows users to input a text description and receive a generated image in response.

The main objective of this application is to demonstrate the capabilities of state-of-the-art AI models available through Hugging Face's API. By leveraging this external service, the application avoids the need for expensive local computation and complex model management. The interface is built with Streamlit, a Python framework that enables rapid development of interactive web apps without requiring frontend programming skills.

This project explores the integration of these technologies to provide an accessible tool for text-to-image generation, highlighting both the opportunities and limitations of using cloud-based AI APIs.

Chapitre 1

System Architecture and Implementation

1.1 System Architecture

The overall workflow of the application relies on a straightforward exchange between the user, the web app, and an external image generation API. The diagram below illustrates this flow :

User → Streamlit Application → Hugging Face API → Generated Image → Displayed in App

Step-by-step process :

1. **Prompt Input :** The user enters a textual description in the Streamlit interface.
2. **Request Sending :** The app sends this text to the Hugging Face API via a secure HTTP POST request.
3. **API Processing :** The image generation model interprets the text and creates a corresponding image.
4. **Image Reception :** The image is returned as a response, usually encoded in base64 or binary form.
5. **Display to User :** Streamlit displays the image directly in the interface.

1.2 Technologies Used

Below are the main technologies involved in this project, with their specific roles :

Technology	Role in the Project
Python	Main programming language for backend logic and Streamlit
Streamlit	Framework for building an interactive, easy-to-deploy web interface
Hugging Face API	Cloud service hosting image generation models (e.g., Stable Diffusion)
requests	Python library to send HTTP requests to the API
dotenv	Secure management of environment variables (e.g., API keys)
Pillow (PIL)	Image processing and display within the app

1.3 Main Code Logic

Key parts of the code with explanations :

1. Capturing user prompt :

```
import streamlit as st
# User input
prompt = st.text_input("Enter a description for the image:")
```

The app waits for the user to enter text. This prompt is the basis for image generation.

2. Sending the API request :

```
import os
import requests
from dotenv import load_dotenv



load_dotenv()
api_key = os.getenv("HUGGINGFACE_API_KEY")
api_url = "https://api-inference.huggingface.co/models/stabilityai/stable-diffusion-3
5-large"
headers = {"Authorization": f"Bearer {api_key}"}
response = requests.post(api_url, headers=headers, json=data)
```

The API key is securely loaded via dotenv. A POST request is sent to the API with the prompt in the request body. The API responds with the generated image or an error message.

3. Processing the response :

```
from io
from PIL import Image
```

```
 if response.status_code == 200:
            image = Image.open(io.BytesIO(response.content))
            # Use the container width instead of column width
            st.image(image, caption=f" Image generated successfully ", use_container_
else:
            st.error(f" Error {response.status_code}: {response.text}")
```

If successful, the image is converted into a manipulable and displayable object. Otherwise, an error message is shown to the user.

4. Displaying the image in Streamlit :

```
if st.button(" Generate Image") and prompt:
    with st.spinner(" The image is being generated..."):
        # Initialize progress bar and timer
        progress_bar = st.progress(0)
        start_time = time.time()  # Start timer
```

The image is displayed on the web page with a caption showing the prompt.

5. Error handling and input validation :

— If the prompt is empty, a warning is displayed and the request is blocked.

— If the API is unreachable or returns an error, an appropriate message is shown.

— Optionally, a loading spinner (`st.spinner`) can be shown while the image is being generated.

This backend-frontend integration offers a smooth, simple, and interactive user experience, while leveraging powerful AI models via API.

Chapitre 2

Demonstration, Results and Evaluation

2.1 Application Screenshots

Below are screenshots illustrating the application interface before and after image generation. The interface clearly displays the user prompt alongside the generated image for easy comparison.

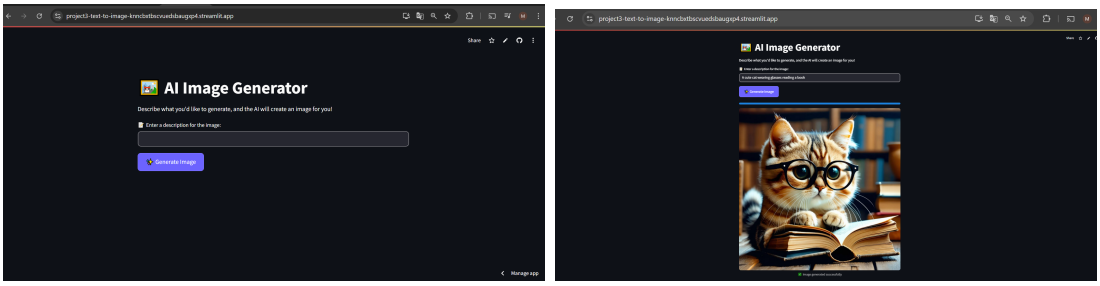
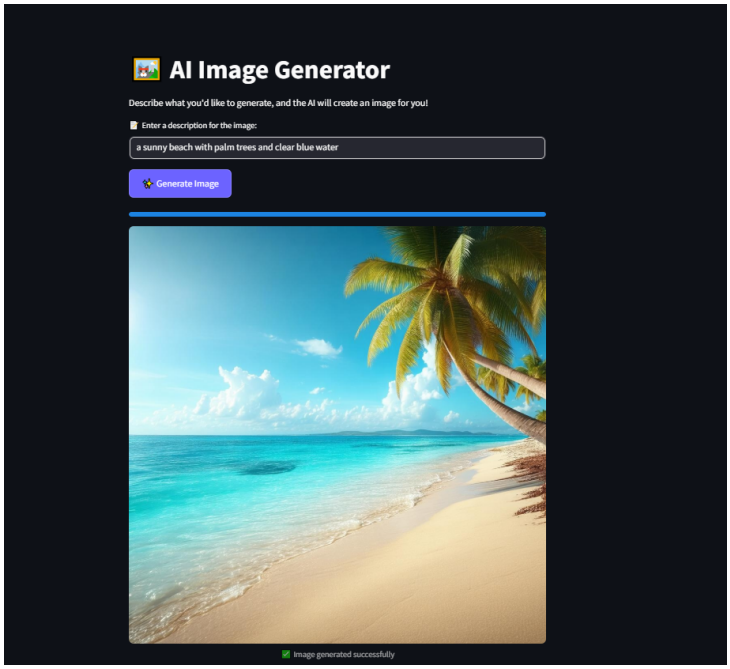


FIGURE 2.1.1 – Application interface before (left) and after (right) image generation

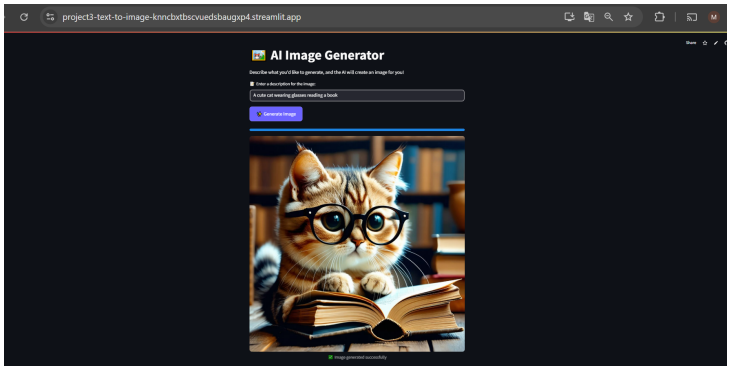
2.2 Prompt Examples & Outputs

Here is a selection of text prompts tested with the application, alongside the resulting generated images :

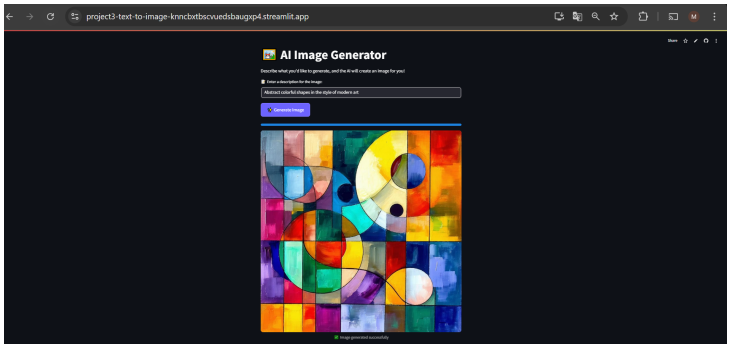
— **Prompt :** "A futuristic cityscape at sunset"



— **Prompt :** "A cute cat wearing glasses reading a book"



— **Prompt :** "Abstract colorful shapes in the style of modern art"



2.3 Evaluation of Results

The generated images generally demonstrate high quality, with good adherence to the textual prompts. The images exhibit coherence and originality, capturing the key elements described in the inputs.

However, several limitations were observed :

— **Response Time :** Generation typically takes several seconds, which may affect user experience for longer prompts.

— **Variable Quality :** Some outputs are less precise or contain artifacts depending on prompt complexity.

— **Errors :** Occasional API errors or timeouts may occur, necessitating error handling in the app.

User feedback indicates that the application is intuitive and engaging, though improvements such as faster generation and richer prompt support would enhance usability.

Conclusion

This project successfully demonstrates the integration of a text-to-image generation system using the Hugging Face API and Streamlit. By allowing users to input a simple text prompt and obtain an AI-generated image, the application makes advanced machine learning models accessible through a clean and interactive interface.

The overall performance was satisfying, with accurate image generation for most descriptive prompts. The use of Python and lightweight libraries (`requests`, `dotenv`, and Streamlit) ensured simplicity and ease of deployment.

Future Improvements

— Support for multiple generation models or style options.

— Local image generation using pre-trained models to reduce API dependency.

— Advanced user controls (e.g., aspect ratio, resolution, artistic style).

— Upload feature to fine-tune results based on user input.

In summary, this project lays the foundation for more robust AI-powered creative tools and illustrates how APIs can simplify complex machine learning tasks for everyday users.