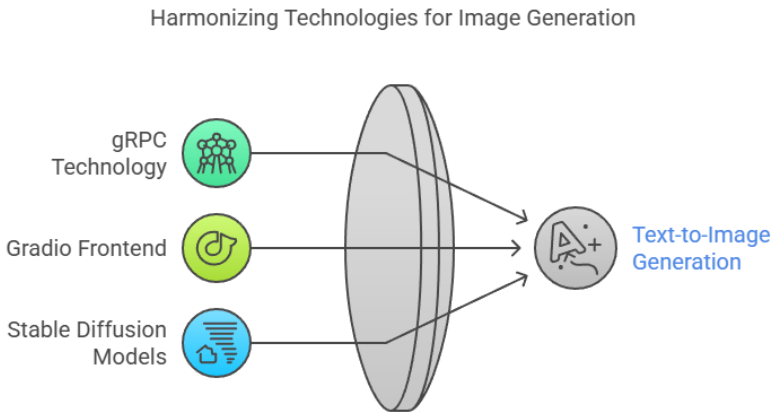
NLP Project

I. **Moeed Asif (21i-0483)**
II. **Ahmad Ali (I210655)**
III. **M Hanzella Khan (P20-0616)**

Text-to-Image Generator Documentation Report

1. Project Overview

The Text-to-Image Generator is a distributed application that combines gRPC technology with a Gradio frontend to provide users with an intuitive interface for generating images from text prompts. The system leverages Stable Diffusion models hosted on Hugging Face Spaces to perform the actual image generation, while maintaining a clean separation between the backend service logic and the user interface.



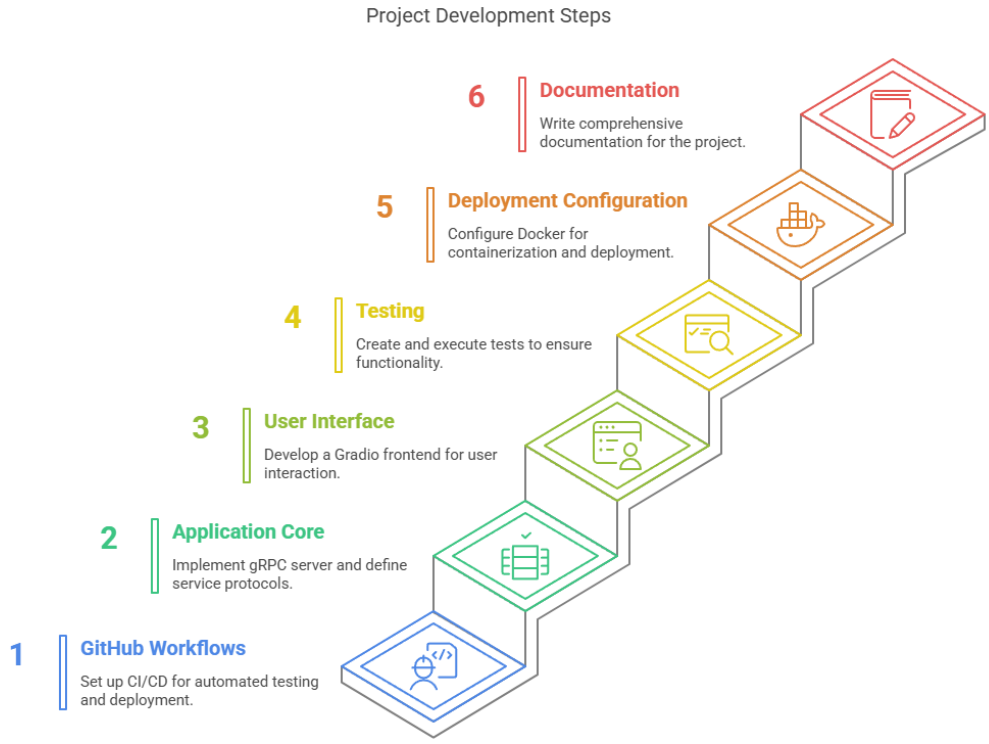
Project Structure

```
NLP-Project/
├── .github/
│   └── workflows/
│       └── main.yml        # GitHub Actions CI/CD workflow
├── app/
│   ├── __init__.py
│   ├── server.py           # gRPC server implementation
│   ├── service.proto       # Protocol Buffers definition for gRPC
│   ├── client.py           # (Empty) Potential gRPC client test script
│   ├── service_pb2.py      # Generated gRPC code (ignored by
git) │   └── service_pb2_grpc.py # Generated gRPC code (ignored
by git)
├── frontend/
│   └── ui.py               # Gradio frontend application
├── tests/
```

```
│   └── postman/
│       └── collection.json # (Empty) Postman collection for API testing
├── .dockerignore           # Specifies files to ignore in Docker build
├── .gitignore              # Specifies intentionally untracked files
├── Dockerfile              # Docker configuration for the application
├── README.md               # Project documentation
└── requirements.txt        # Python dependencies
```

The project is structured as follows:

- **GitHub Workflows (.github/workflows/)**: Contains CI/CD configuration that automates testing, linting, and code generation when changes are pushed to the main branch.
- **Application Core (app/)**:
  - `server.py` : Contains the gRPC server implementation that handles incoming requests and communicates with the Hugging Face API.
  - `service.proto` : Defines the Protocol Buffers schema for the gRPC service interface.
  - `client.py` : A placeholder file for potential client-side testing.
  - `service_pb2.py` and `service_pb2_grpc.py` : Auto-generated gRPC code from the proto definition (not tracked in git).
- **User Interface (frontend/)**:
  - `ui.py` : Implements the Gradio web interface that users interact with to generate images.
- **Testing (tests/)**:
  - Contains a Postman collection file (currently empty) for API testing.
- **Deployment Configuration**:
  - `Dockerfile` : Defines how to build the application container.
  - `.dockerignore` : Specifies which files should be excluded from the Docker build.
  - `.gitignore` : Lists files that should not be tracked by version control.
- **Documentation and Dependencies**:
  - `README.md` : Comprehensive documentation about the project.
  - `requirements.txt` : Lists all Python package dependencies.



2. Setup Instructions

Prerequisites

- Python 3.10 or later
- pip package manager
- Git (for repository management)
- Internet connection (to access Hugging Face model APIs)

Installation Steps

1. **Clone the repository**

```
bash git clone
<repository-url> cd NLP-
Project
```

2. **Set up a virtual environment (recommended)**

```bash
python -m venv
```

```
venv
```

```
# Activate on Windows
.\venv\Scripts\activate

# Activate on macOS/Linux
source venv/bin/activate
```

3. **Install dependencies**

```bash
pip install -r
```

```
requirements.txt
```

4. **Generate gRPC code from Protocol Buffers definition**

```bash
python -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=.
```

```
app/service.proto
```

5. **Configure Hugging Face API endpoint**
   - Open `app/server.py`
   - Replace `"https://your-space-url.hf.space/generate"` with your actual Hugging Face Space API endpoint URL

3. Usage Guide

Running the Application

1. **Start the gRPC server**

```bash
```

```
# In terminal 1 (with virtual environment activated)
python app/server.py
```

2. **Launch the Gradio frontend**

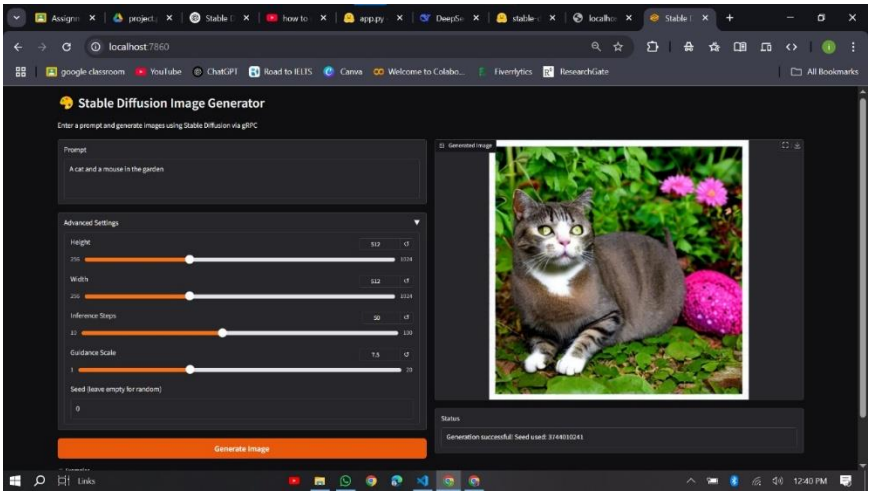
```bash
```

```
# In terminal 2 (with virtual environment activated)
python frontend/ui.py
```

3. **Access the web interface**
    - Open your browser and navigate to the URL displayed in the terminal (typically `http://127.0.0.1:7860`)
    - Enter a text prompt in the input field
    - Click "Submit" to generate an image
    - The generated image will be displayed in the interface

Example Generations

The model can generate a wide variety of images based on text prompts. Here is an example of a creative generation:



Docker Deployment

For containerized deployment:

1. **Build the Docker image**

   ```bash
   docker build -t nlp-project .
   ```

2. **Run the container**

   ```bash
   docker run -p 7860:7860 nlp-project
   ```

3. **Access the application**
   - Open your browser and navigate to `http://localhost:7860`

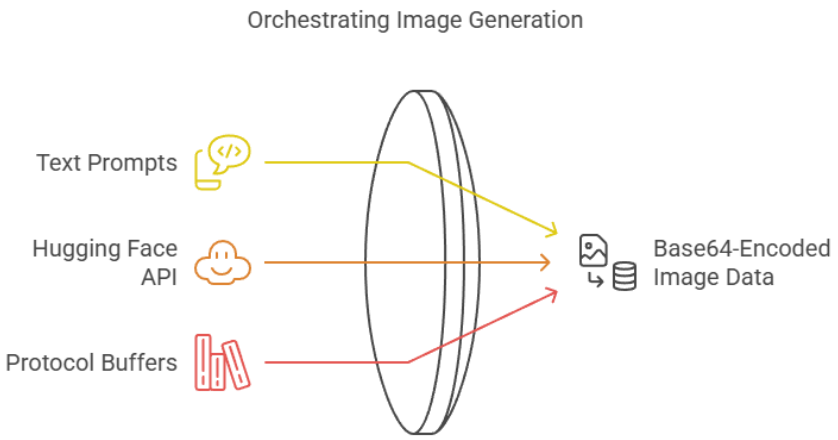
4. Architecture Description

The system follows a client-server architecture with the following components:

Backend Service (gRPC Server)

- Implemented in `app/server.py`
- Listens on port 50051 for incoming gRPC requests
- Provides the `GenerateImage` RPC method that:
  - Receives text prompts from clients
  - Forwards requests to the Hugging Face Space API
  - Returns base64-encoded image data to clients
-

Uses Protocol Buffers for efficient, language-agnostic data serialization



Frontend Application (Gradio UI)

- Implemented in `frontend/ui.py`
- Provides a web-based user interface using Gradio library
- Connects to the gRPC server as a client
- Handles user interactions, prompt submission, and image display
- Decodes base64 image data for rendering in the browser

Communication Protocol

- Defined in `app/service.proto`
- Specifies the service interface and message formats:
  - `PromptRequest` : Contains the text prompt from the user
  - `ImageReply` : Contains the base64-encoded image generated by the model

CI/CD Pipeline

- GitHub Actions workflow in `.github/workflows/main.yml`
- Automates testing, linting, and gRPC code generation
- Ensures code quality and consistent builds

5. Model Information

Model Source

The application uses Stable Diffusion models hosted on Hugging Face Spaces

-
- The specific model is accessed via a REST API endpoint specified in the `HUGGINGFACE_API` variable in `app/server.py`
- Image generation processing happens on Hugging Face's infrastructure, not locally

Integration Method

- The backend server forwards text prompts to the Hugging Face API via HTTP POST requests
- The Hugging Face Space processes the prompt and returns a base64-encoded image
- This approach leverages cloud resources for computationally intensive image generation

6. Limitations and Considerations

Technical Limitations

- Requires constant connection to Hugging Face API services
- Performance depends on external API response times
- No offline mode or local model inference capabilities
- Limited error handling for API downtime or service disruptions

Scalability Considerations

- The current implementation doesn't include load balancing for multiple users
- No caching mechanism for repeated prompts
- API rate limits may apply from the Hugging Face service

Deployment Notes

- Docker container needs network access to both the Hugging Face API and the local network for UI exposure
- The README doesn't specify authentication mechanisms for the service
- No configuration for TLS/SSL security on the gRPC connection

Future Improvements

- Implement authentication for the API and UI
- Add support for additional model parameters (image size, sampling steps, etc.)
- Create a more robust error handling system
- Enable local caching of generated images to reduce redundant API calls
- Implement monitoring and logging for system performance and user analytics.