AI4001/CS4063 - Fundamentals of NLP/NLP Course Project.

Story2Audio Microservice

Members

21I-0560 Talha Arjumand

21I-2468 Albab Nawaz

21I-0730 Alian Anwar

Story2Audio Microservice

Overview

This project, **Story2Audio**, is developed as part of the **AI4001/CS4063 - Fundamentals of NLP/NLP Course Project**. It converts a given storyline into an engaging audio story using local models for text enhancement and text-to-speech (TTS). The project is implemented as a microservice with a gRPC API, a Gradio frontend for demo, and containerized deployment using Docker. The pipeline includes preprocessing, text enhancement, audio generation, and stitching, all wrapped in a scalable API with testing and documentation.

Project Phases

- **Phase 1:** Initial setup, environment configuration, and dependency installation.
- **Phase 2:** Core pipeline development (preprocessing, enhancement, TTS, audio stitching).
- **Phase 3:** gRPC API development with async support and error handling.
- **Phase 4:** Gradio frontend for user interaction with the API.
- **Phase 5:** Documentation, test cases, and performance evaluation.

Setup and Requirements

Prerequisites

- **Operating System:** Windows/Linux/MacOS
- **Python:** 3.11
- **FFmpeg:** Required for audio processing (pydub)
    - Windows: choco install ffmpeg
    - Linux/MacOS: sudo apt-get install ffmpeg or brew install ffmpeg
- **Docker:** For containerization
- **Postman:** For API testing

Dependencies

Install the required Python packages using the provided requirements.txt:

grpcio==1.71.0

grpcio-tools==1.71.0

transformers==4.51.3

torch==2.4.0

kokoro

pydub

soundfile

gradio

pytest

matplotlib

locust

Installation Steps

1. Clone the repository:
2. git clone <your-repo-url>
3. cd <project-directory>
4. Create and activate a virtual environment:
5. python -m venv venv
6. venv\Scripts\activate # Windows
7. source venv/bin/activate # Linux/MacOS
8. Install dependencies:
9. pip install -r requirements.txt
10. Ensure FFmpeg is installed (see Prerequisites).

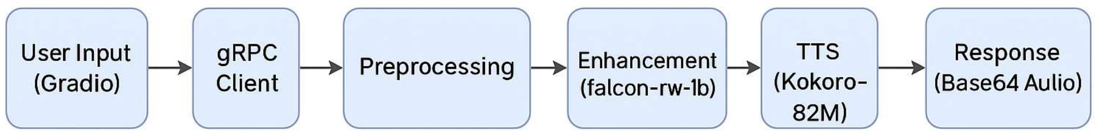
Project Architecture

Pipeline Overview

The Story2Audio pipeline consists of the following stages:

1. **Text Preprocessing:** Splits the input story into chunks (~150 words each) using src/preprocess.py.
2. **Text Enhancement:** Enhances each chunk for emotional storytelling using tiiuae/falcon-rw-1b (src/enhancer_local.py).
3. **Text-to-Speech (TTS):** Converts enhanced text to audio using hexgrad/Kokoro-82M (src/kokoro_tts.py).
4. **Audio Stitching:** Combines audio chunks into a single .mp3 file using pydub (src/utils.py).
5. **gRPC API:** Wraps the pipeline in a /GenerateAudio endpoint (api/server.py).
6. **Frontend:** A Gradio interface for user interaction (frontend.py).

Architecture Diagram



Directory Structure

```
Story2Audio/
├── api/
│   ├── client.py          # gRPC client for testing
│   ├── grpc_client.py     # gRPC client for frontend
│   ├── server.py          # gRPC server implementation
├── src/
│   ├── enhancer_local.py  # Text enhancement logic
│   ├── kokoro_tts.py      # TTS logic
│   ├── preprocess.py      # Story chunking logic
│   ├── utils.py           # Audio stitching logic
├── tests/
│   ├── test_api.py        # Unit tests for gRPC API
│   ├── performance_test.py # Performance test script
├── Dockerfile             # Docker configuration
├── frontend.py            # Gradio frontend
```

```
├── requirements.txt        # Project dependencies
├── story2audio.proto       # gRPC service definition
├── sample_story.txt        # Sample input story
└── README.md               # Project documentation
```

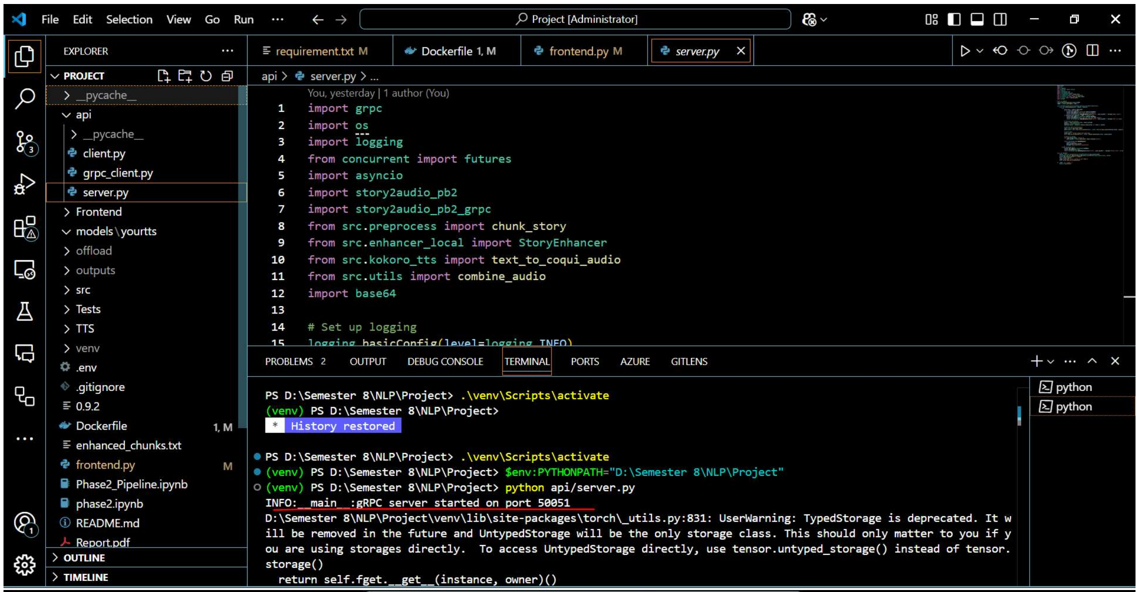
Models Used

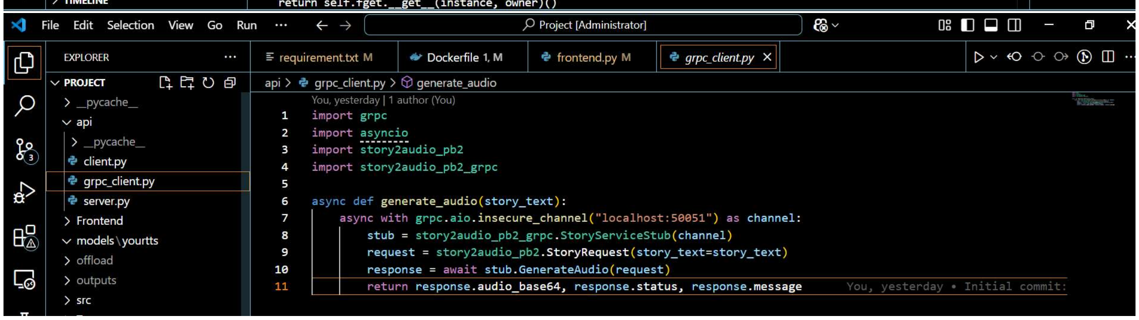
- **Text Enhancement: tiiuae/falcon-rw-1b (Hugging Face)**
  - Used for enhancing storytelling tone.
  - Source: Hugging Face Model Hub
- **Text-to-Speech: hexgrad/Kokoro-82M**
  - Generates expressive audio from text.
  - Source: Local installation (assumed pre-downloaded as per Phase 2).

Usage

Running the gRPC Server

1. Start the server:
2. python api/server.py
3. The server will run on localhost:50051.

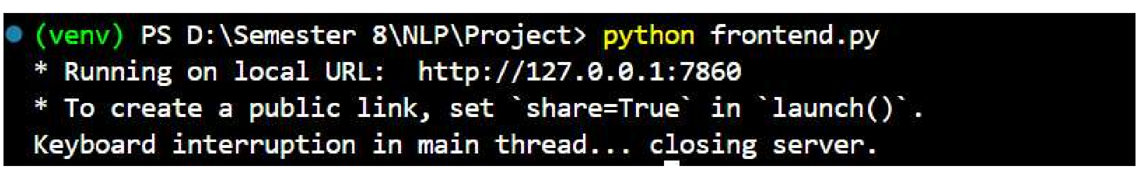


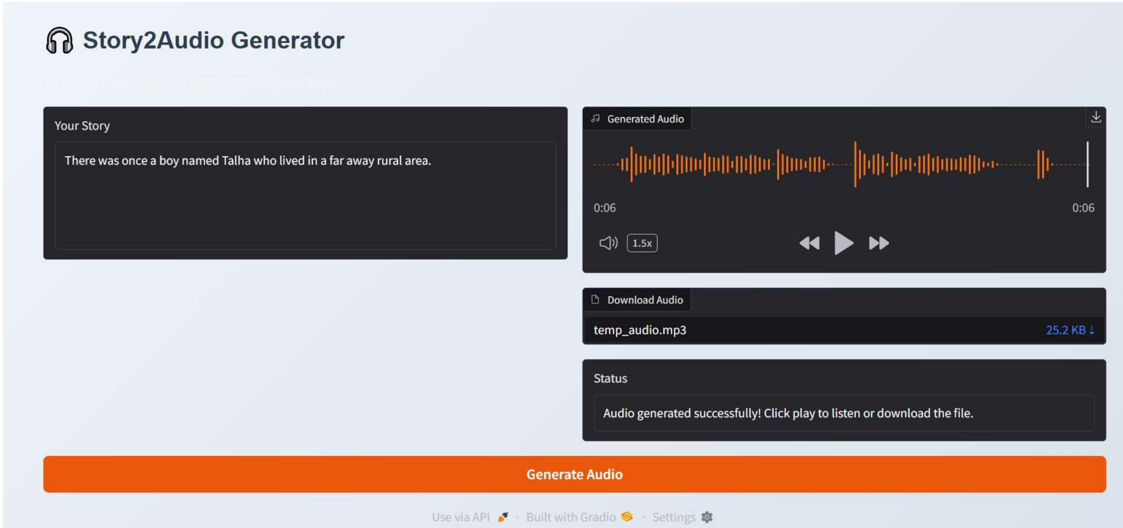


A reusable function to call the gRPC API from the frontend.

Using the Gradio Frontend

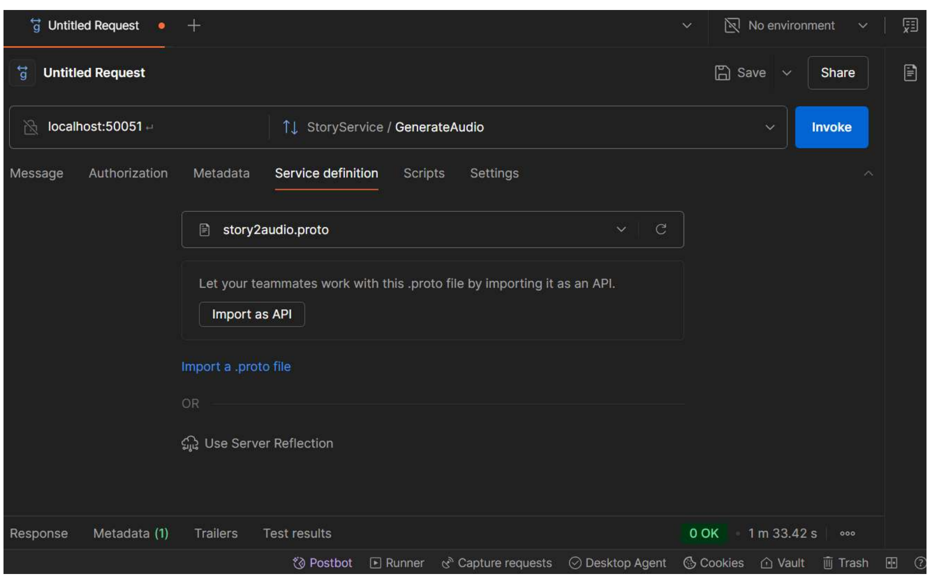
1. Ensure the gRPC server is running.
2. Launch the frontend:
3. python frontend.py
4. Open the provided URL (e.g., http://127.0.0.1:7860) in your browser.
5. Enter a story in the text box and click "Generate Audio" to hear the output.

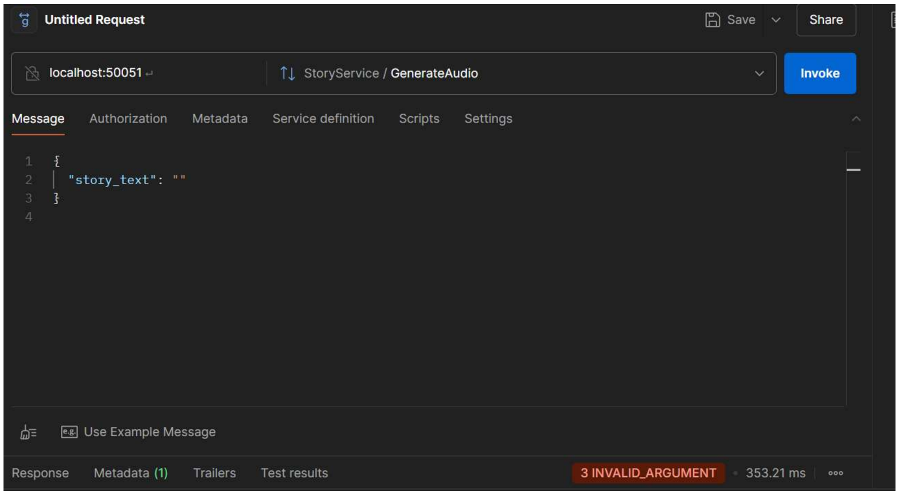




Testing with Postman

1. Import story2audio.proto into Postman.
2. Create a gRPC request to localhost:50051 with the GenerateAudio method.
3. Send a request with a story (e.g., {"story_text": "Once upon a time..."}).
4. Check the response for status, audio_base64, and message.





Test Cases and Results

Unit Tests

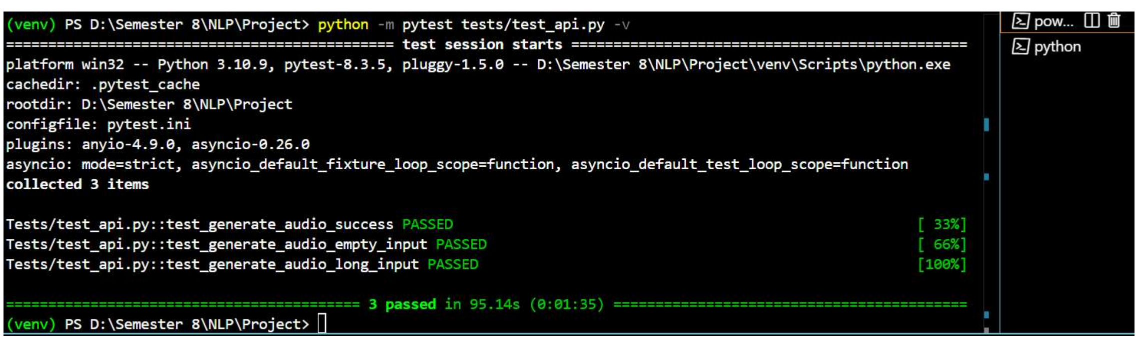
Unit tests for the gRPC API are implemented in tests/test_api.py. They cover:

- Successful audio generation
- Empty input handling
- Server error handling

Run Tests:

python -m pytest tests/test_api.py

Example Results:



Performance Evaluation

Performance tests measure concurrent requests vs. response time using locust.

Run Performance Test:

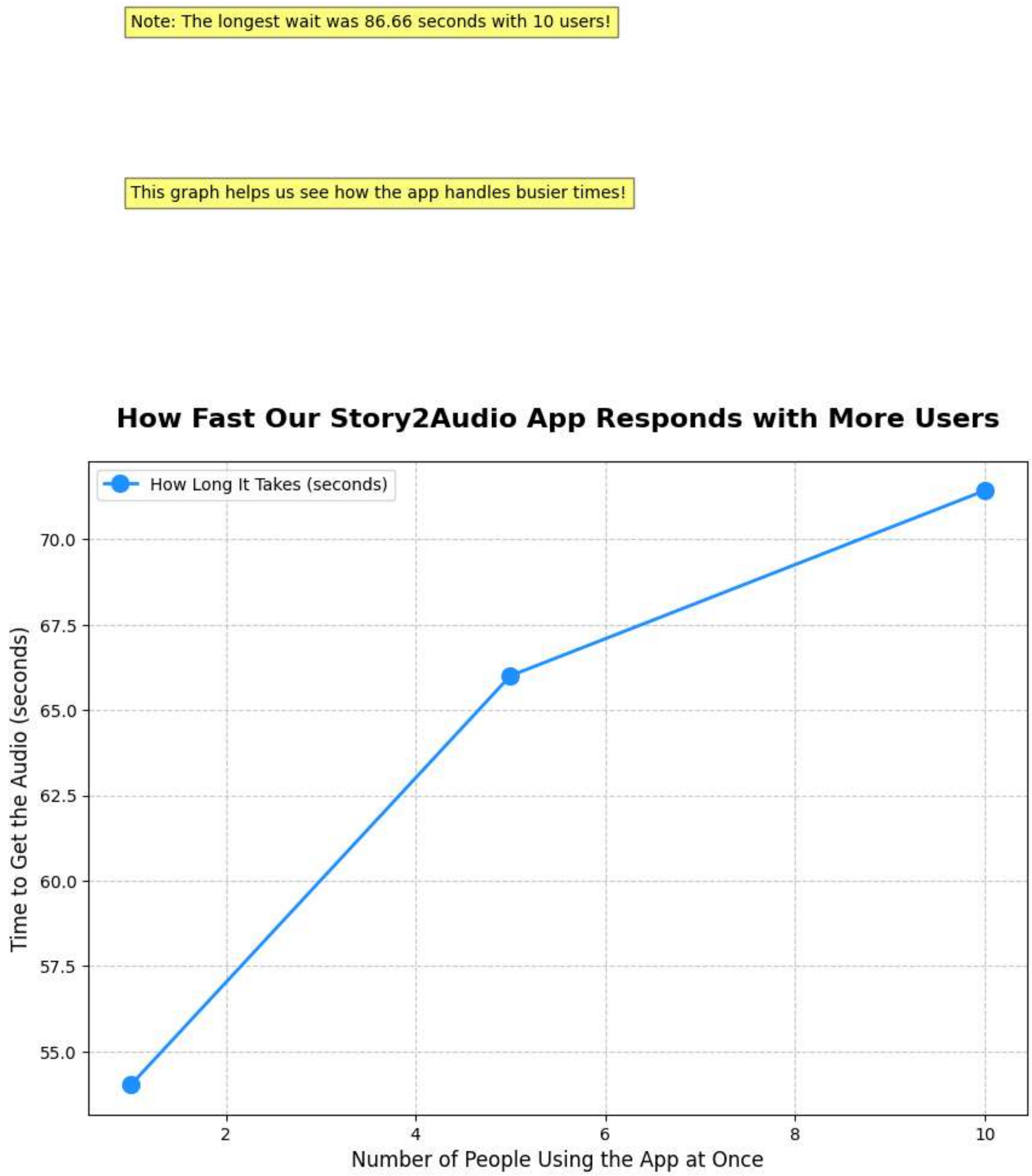
1. Start the gRPC server:
2. python api/server.py
3. Run the performance test:
4. locust -f tests/performance_test.py --headless -u 10 -r 2 --run-time 1m
    - -u 10: 10 concurrent users
    - -r 2: Spawn rate of 2 users/sec
    - --run-time 1m: Run for 1 minute

```
[2025-05-06 18:56:07,606] DESKTOP-GLC14S8/INFO/locust.main: Starting Locust 2.37.0
[2025-05-06 18:56:07,611] DESKTOP-GLC14S8/INFO/locust.main: Run time limit set to 60 seconds
Type     Name                                  # reqs      # fails |    Avg     Min     Max    Med |   req/s  failures/s
--------|-----------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
--------|-----------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
         Aggregated                                 0     0(0.00%) |      0       0       0      0 |    0.00        0.00

[2025-05-06 18:56:07,613] DESKTOP-GLC14S8/INFO/locust.runners: Ramping to 10 users at a rate of 2.00 per second
Type     Name                                  # reqs      # fails |    Avg     Min     Max    Med |   req/s  failures/s
--------|-----------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
gRPC     GenerateAudio                              2     0(0.00%) |  85848   85038   86657  85038 |    0.00        0.00
--------|-----------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
         Aggregated                                 2     0(0.00%) |  85848   85038   86657  85038 |    0.00        0.00

[2025-05-06 18:58:59,398] DESKTOP-GLC14S8/INFO/locust.main: --run-time limit reached, shutting down
```

```
[2025-05-06 19:02:21,895] DESKTOP-GLC14S8/INFO/locust.main: Shutting down (exit code 0)
Type     Name                                  # reqs      # fails |    Avg     Min     Max    Med |   req/s  failures/s
--------|-----------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
gRPC     GenerateAudio                              5     0(0.00%) |  71422   54028   86657  66000 |    0.00        0.00
--------|-----------------------------------|-------|-------------|-------|-------|-------|-------|--------|-----------
         Aggregated                                 5     0(0.00%) |  71422   54028   86657  66000 |    0.00        0.00
```

Results:

Performance Graph:



Limitations

- **Model Constraints:** falcon-rw-1b can be slow on CPU; GPU acceleration is recommended for production.
- **Audio Quality:** Kokoro-82M may struggle with certain accents or emotional tones.

- **Scalability:** The current setup may face bottlenecks with very high concurrency (>50 users) due to local TTS processing.
- **Error Handling:** Limited timeout handling for long audio generation tasks.
- **Frontend:** Gradio is suitable for demos but not production-grade.

Future Improvements

- Add GPU support for faster inference.
- Implement advanced timeout and retry mechanisms.
- Use a production-grade frontend framework (e.g., React).
- Optimize audio generation for higher concurrency.

Acknowledgments

- Models: tiiuae/falcon-rw-1b (Hugging Face), hexgrad/Kokoro-82M.
- Libraries: transformers, kokoro, pydub, gradio, grpcio.

Github Repo link:
https://github.com/Albab001/-Story2Audio-Microservice