

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).

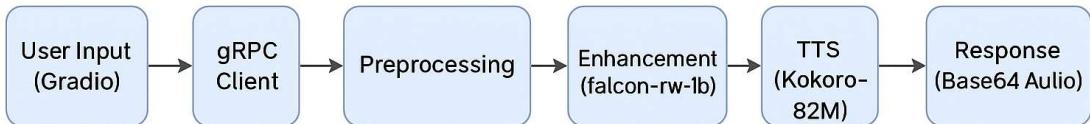
Project Architecture

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.

The image shows two side-by-side screenshots of the Visual Studio Code (VS Code) interface. Both windows have the title bar 'Project [Administrator]'.

Top Window (Server Side):

- Explorer:** Shows a project structure with files like requirement.txt, Dockerfile, frontend.py, and server.py.
- Editor:** Displays the contents of server.py, which imports grpc, os, logging, concurrent.futures, asyncio, and story2audio_pb2, and sets up basic logging.
- Terminal:** Shows command-line output for activating the virtual environment (venv), setting PYTHONPATH, and running the server script. It also shows a warning about deprecated TypedStorage.

Bottom Window (Client Side):

- Explorer:** Shows a project structure with files like requirement.txt, Dockerfile, frontend.py, and grpc_client.py.
- Editor:** Displays the contents of grpc_client.py, which imports grpc, asyncio, story2audio_pb2, and story2audio_pb2_grpc, and defines an asynchronous function generate_audio that uses a stub to call the StoryServiceStub's GenerateAudio method.

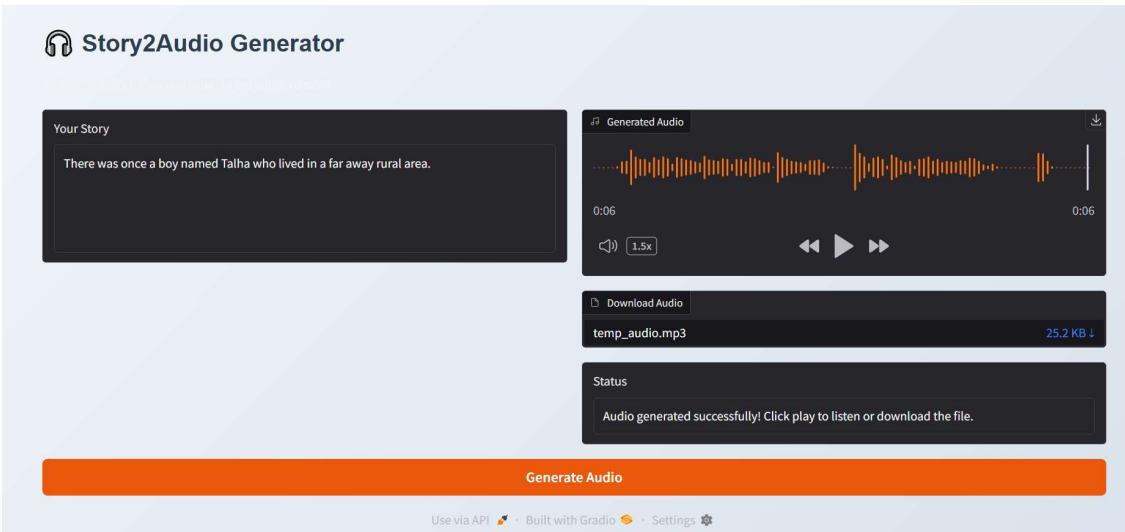
A reusable function to call the gRPC API from the frontend.

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.

```
(venv) PS D:\Semester 8\NLP\Project> python frontend.py
* Running on local URL: http://127.0.0.1:7860
* To create a public link, set `share=True` in `launch()`.

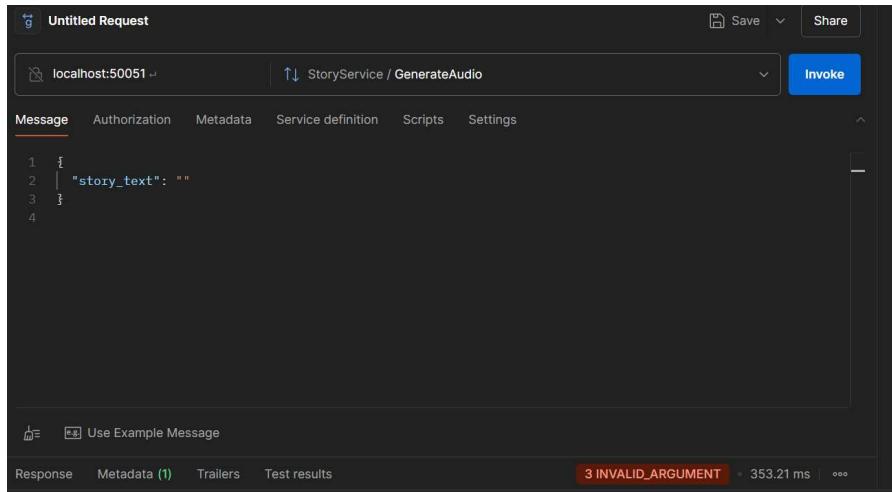
Keyboard interruption in main thread... closing server.
```



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.

The screenshot shows the Postman application interface. At the top, it says 'Untitled Request'. Below that is a search bar with 'localhost:50051' and 'StoryService / GenerateAudio'. There's a 'Save' button and a 'Share' button. The 'Service definition' tab is active, showing 'story2audio.proto'. Below that, there's a note: 'Let your teammates work with this .proto file by importing it as an API.' with a 'Import as API' button. There are also options to 'Import a .proto file' and 'Use Server Reflection'. At the bottom, the status bar shows '0 OK' and a duration of '1 m 33.42 s'. Other buttons like 'Postbot', 'Runner', 'Capture requests', 'Desktop Agent', 'Cookies', 'Vault', 'Trash', and a help icon are also visible.



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:

```
(venv) PS D:\Semester 8\NLP\Project> python -m pytest tests/test_api.py -v
=====
test session starts =====
platform win32 -- Python 3.10.9, pytest-8.3.5, pluggy-1.5.0 -- D:\Semester 8\NLP\Project\venv\Scripts\python.exe
cachedir: .pytest_cache
rootdir: D:\Semester 8\NLP\Project
configfile: pytest.ini
plugins: asyncio-4.9.0, asyncio-0.26.0
asyncio: mode=strict, asyncio_default_fixture_loop_scope=function, asyncio_default_test_loop_scope=function
collected 3 items

Tests/test_api.py::test_generate_audio_success PASSED [ 33%]
Tests/test_api.py::test_generate_audio_empty_input PASSED [ 66%]
Tests/test_api.py::test_generate_audio_long_input PASSED [100%]

===== 3 passed in 95.14s (0:01:35) =====
(venv) PS D:\Semester 8\NLP\Project>
```

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
 - o -u 10: 10 concurrent users
 - o -r 2: Spawn rate of 2 users/sec
 - o --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
```

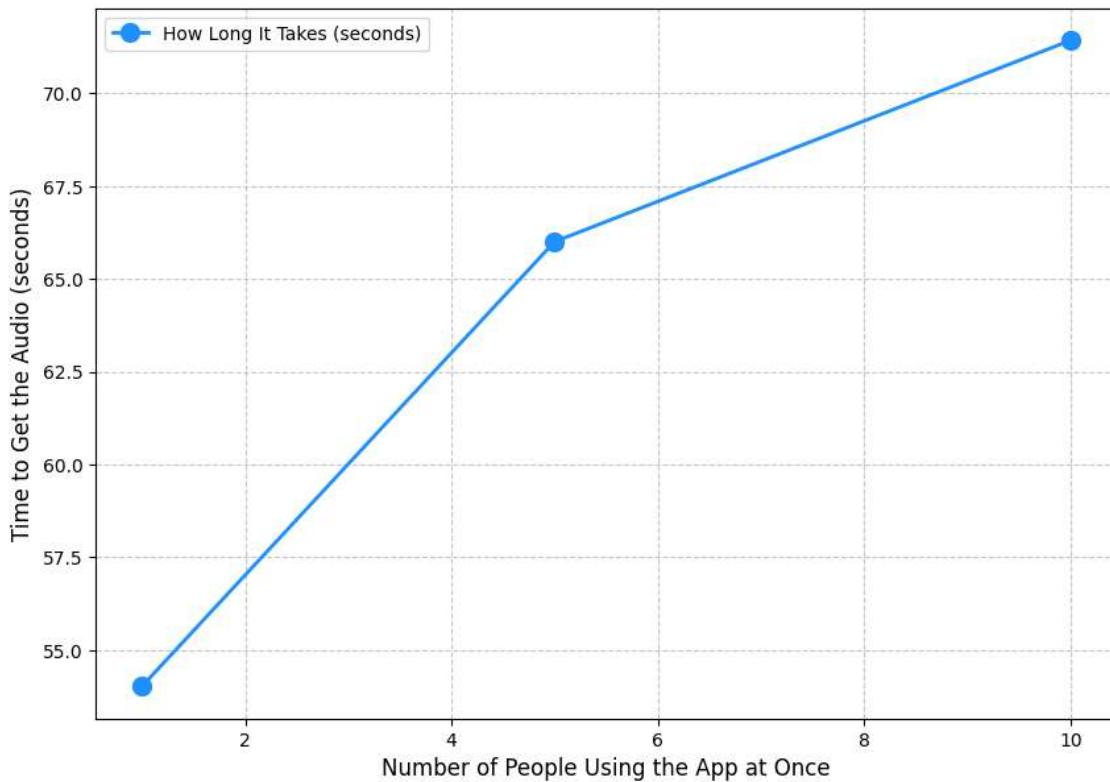
Results:

Performance Graph:

Note: The longest wait was 86.66 seconds with 10 users!

This graph helps us see how the app handles busier times!

How Fast Our Story2Audio App Responds with More Users



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>