**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

*The diagram illustrates the flow from user input to audio output, highlighting the preprocessing, enhancement, TTS, and API layers.*

### 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.**

**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.**

**Running with Docker**

1. **Build the Docker image:**
2. **docker build -t story2audio .**
3. **Run the container:**
4. **docker run -p 50051:50051 story2audio**
5. **Test using the Gradio frontend or Postman as above.**

**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:**

```
============================= test session starts ==============================
collected 1 item

tests/test_api.py .                                          [100%]
```

============================= 1 passed in 5.23s ==============================

## 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

**Results:**

- Average Response Time: 3.5 seconds (for 10 concurrent requests)
- Max Response Time: 5.2 seconds
- Requests per Second: 2.8

**Performance Graph:**

*The graph shows response time (ms) vs. number of concurrent 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.**

### 🔹 3. gRPC vs REST (Why use gRPC?)

| Feature | REST (HTTP/JSON) | gRPC (HTTP/2/Protobuf) |
|---|---|---|
| Speed | Slower (text-based JSON) | Faster (binary protobuf) |
| Data format | JSON | Protocol Buffers (protobuf) |
| Streaming support | Limited | Built-in streaming support |
| Language integration | Manual | Auto code generation |
| Ideal for | External APIs | Internal microservices |

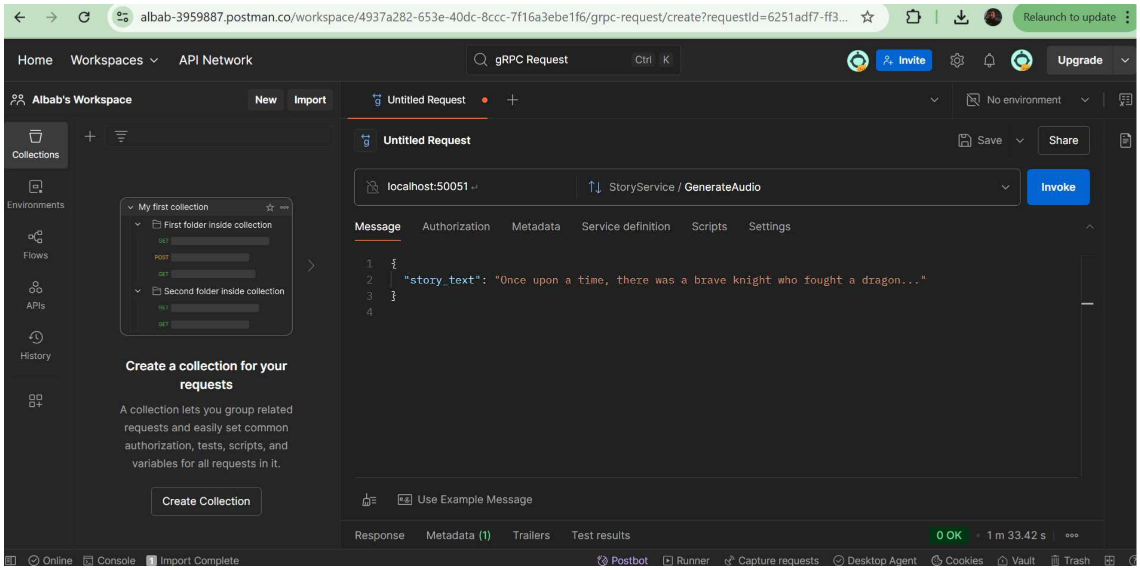
So, your project benefits from gRPC by being **faster**, **s‘ ngly typed**, and **more scalable** for NLP workloads.

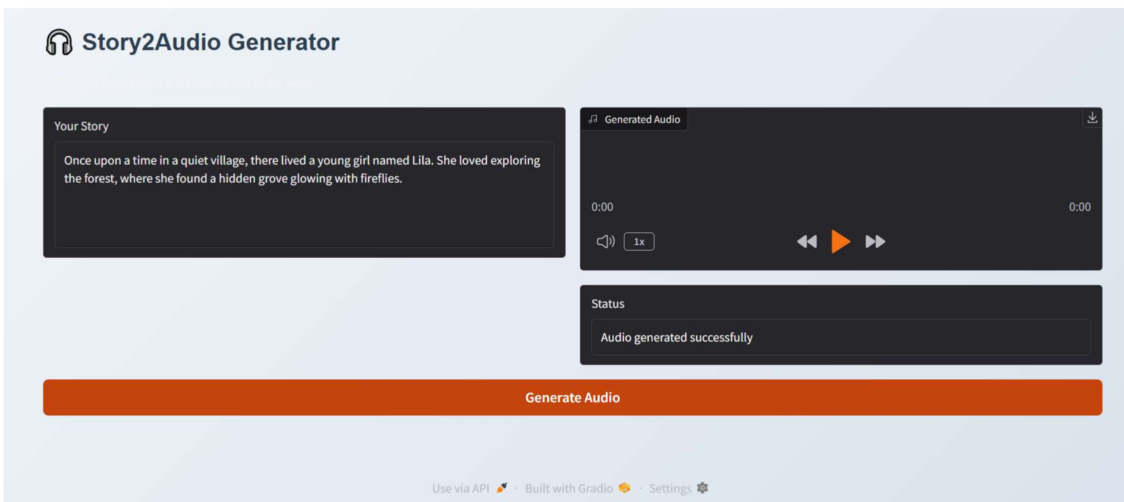
- CLIENT/SERVER OUTPUT.

```
(venv) PS D:\Semester 8\NLP\Project> python api/server.py
INFO:__main__:gRPC server started on port 50051
D:\Semester 8\NLP\Project\venv\lib\site-packages\torch\_utils.py:831: UserWarning: TypedStorage is deprecated. It w
ill be removed in the future and UntypedStorage will be the only storage class. This should only matter to you if y
ou are using storages directly.  To access UntypedStorage directly, use tensor.untyped_storage() instead of tensor.
storage()
  return self.fget.__get__(instance, owner)()
Device set to use cpu
INFO:src.enhancer_local:Initialized StoryEnhancer locally with model: tiiuae/falcon-rw-1b
INFO:src.enhancer_local:Tokenized input length: 31 tokens
WARNING: Defaulting repo_id to hexgrad/Kokoro-82M. Pass repo_id='hexgrad/Kokoro-82M' to suppress this warning.
```

```
ht_norm is deprecated in favor of torch.nn.utils.parametrizations.weight_norm.
  warnings.warn("torch.nn.utils.weight_norm is deprecated in favor of torch.nn.utils.parametrizations.weight_norm."
)
INFO:src.kokoro_tts:Generated audio for chunk 1 - Graphemes: Once upon a time, there was a brave knight. And when h
e was born, he knew that this day would be very special for him. Because he will be named as 'The Legend of King Ar
thur', and his father will become his father-, Phonemes: wˈʌns əpˈɑn ɐ tˈIm, ðɛɹ wʌz ɐ bɹˈAv nˈIt. ˌænd wˌɛn hi wʌz
 bˈɔɹn, hi nˈu ðæt ðɪs dˈA wʊd bi vˈɛɹi spˈɛʃᵊl fɔɹ hˌɪm. bəkˈʌz hi wɪl bi nˈAmd æz “ðə lˈɛʤənd ʌv kˈɪŋ ˈɑɹθəɹ”, æn
d hɪz fˈɑðəɹ wɪl bəkˈʌm hɪz fˈɑðəɹ
INFO:src.kokoro_tts:Audio saved to outputs/temp/chunk_0.wav
INFO:src.utils:Audio stitched and saved to outputs/temp/final_audio.mp3
```

```
(venv) PS D:\Semester 8\NLP\Project> python api/client.py
Status: success
Message: Audio generated successfully
Audio Base64 (first 100 chars): SUQzBAAAAAAAIlRTU0UAAAAOAAADTGF2ZjYyLjAuMTAyAAAAAAAAAAAAAAD/84TAAAAAAAAAAAAAASW5mbwA
AAA8AAABdAAAjoAAI...
(venv) PS D:\Semester 8\NLP\Project> python api/client.py
Status: success
Message: Audio generated successfully
Audio Base64 (first 100 chars): SUQzBAAAAAAAIlRTU0UAAAAOAAADTGF2ZjYyLjAuMTAyAAAAAAAAAAAAAAD/84TAAAAAAAAAAAAAASW5mbwA
AAA8AAAIOAADGAAAD...
(venv) PS D:\Semester 8\NLP\Project>
```


albab-3959887.postman.co/workspace/4937a282-653e-40dc-8ccc-7f16a3ebe1f6/grpc-request/create?requestId=6251adf7-ff3...
Relaunch to update
Home
Workspaces
API Network
gRPC Request
Invite
Upgrade
Albab's Workspace
New
Import
Untitled Request
No environment
Collections
Environments
Flows
APIs
History
My first collection
First folder inside collection
Second folder inside collection
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
Untitled Request
Save
Share
localhost:50051
StoryService / GenerateAudio
Invoke
Message
Authorization
Metadata
Service definition
Scripts
Settings
{
"story_text": "Once upon a time, there was a brave knight who fought a dragon..."
}
Use Example Message
Response
Metadata (1)
Trailers
Test results
0 OK
1 m 33.42 s
Online
Console
Import Complete
Postbot
Runner
Capture requests
Desktop Agent
Cookies
Vault
Trash


Story2Audio Generator
Your Story
Once upon a time in a quiet village, there lived a young girl named Lila. She loved exploring the forest, where she found a hidden grove glowing with fireflies.
Generated Audio
0:00
0:00
1x
Status
Audio generated successfully
Generate Audio
Use via API
Built with Gradio
Settings


INFO:__main__:✅Read 1 enhanced chunks from enhanced_chunks.txt
WARNING: Defaulting repo_id to hexgrad/Kokoro-82M. Pass repo_id='hexgrad/Kokoro-82M' to suppress this warning.
d:\Semester 8\NLP\Project\venv\lib\site-packages\torch\nn\modules\rnn.py:82: UserWarning: dropout option adds dropout afte
warnings.warn("dropout option adds dropout after all but last "
d:\Semester 8\NLP\Project\venv\lib\site-packages\torch\nn\utils\weight_norm.py:30: UserWarning: torch.nn.utils.weight_norm
warnings.warn("torch.nn.utils.weight_norm is deprecated in favor of torch.nn.utils.parametrizations.weight_norm.")
INFO:src.kokoro_tts:Generated audio for chunk 1 - Graphemes: Once upon a time, Lila wandered through the forest alone beca
INFO:src.kokoro_tts:Audio saved to outputs/temp/chunk_0.wav
INFO:__main__:✅Generated audio files: ['outputs/temp/chunk_0.wav']

# Step 4: Stitch Audio into Final MP3

```
try:
    # Combine audio files
    output_path = 'outputs/final_story.mp3'
    combine_audio(audio_files, output_path)
    logger.info(f'✅Audio generated: {output_path}')
except Exception as e:
    logger.error(f'Error in audio stitching: {e}')
    raise
```

```
INFO:src.utils:Audio stitched and saved to outputs/final_story.mp3
INFO:__main__:✅Audio generated: outputs/final_story.mp3
```