Natural Language Processing

Course Project

Department of Artificial Intelligence

National University of Computer and Emerging Sciences
Islamabad, Pakistan

Submitted by:

21I-0255 Qasim Jalil

21I-0345 Amna Khan

21I-2706 Ansab Alam

Project Report: Text-to-Audio Microservice

1 Introduction

This project presents a microservice designed to convert textual stories into audio using the Bark text-to-speech model. The service is implemented using Python and leverages gRPC for efficient communication. It supports concurrent requests, provides error handling, and includes a user-friendly frontend via Gradio.

2 Architecture Overview

2.1 Microservice Components

i. GRPC Server (`main.py`): Handles incoming gRPC requests and delegates processing to the TTS module.
ii. Text-to-Speech Module (`tts.py`): Processes text input, detects emotions, and generates corresponding audio using the Bark model.
iii. Emotion Detection (`emotion.py`): Analyzes text to determine the underlying emotion.
iv. Voice Profiles (`voices.py`): Defines various voice profiles.
v. Model Preloading (`loadmodels.py`): Preloads necessary models.
vi. Frontend Interface (`gradioapp.py`): Web interface for text input and audio output.

2.2 Communication Flow

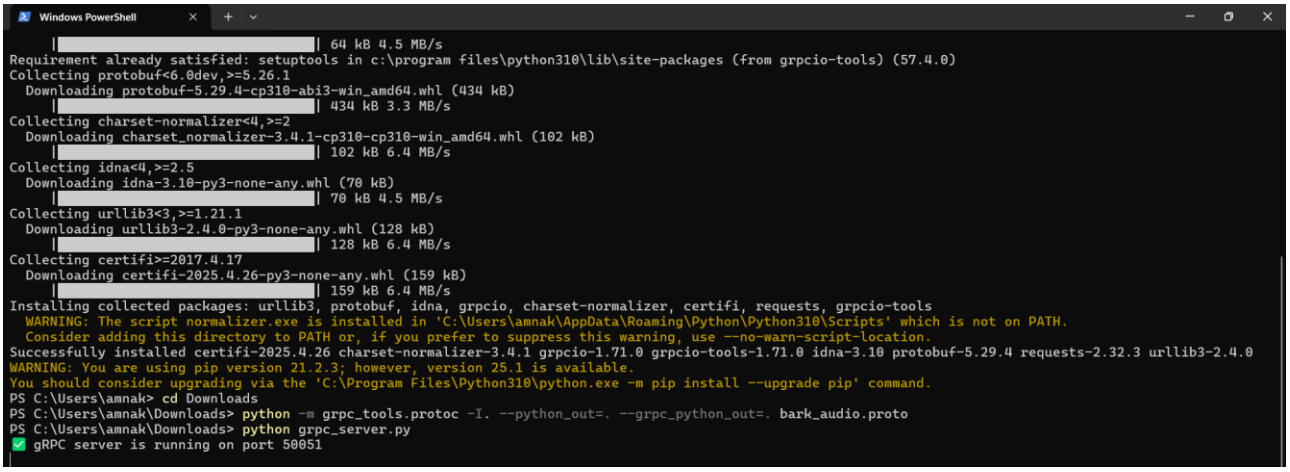
i. A client sends a `Generate` request via gRPC.
ii. The server processes the request asynchronously.

iii. The TTS module detects emotions and generates audio.
iv. The generated audio is returned to the client.

3 Implementation Details

3.1 gRPC API

- Endpoint: `Generate`
- Request: `AudioRequest` with `text` and `voice`
- Response: `AudioResponse` with `success`, `status`, `text`, and `audio`



3.2 Asynchronous Processing

The server utilizes Python's `asyncio` to handle multiple requests concurrently.

3.3 Error Handling

Invalid Input checks for empty or invalid inputs and processing errors catches exceptions and returns error messages.

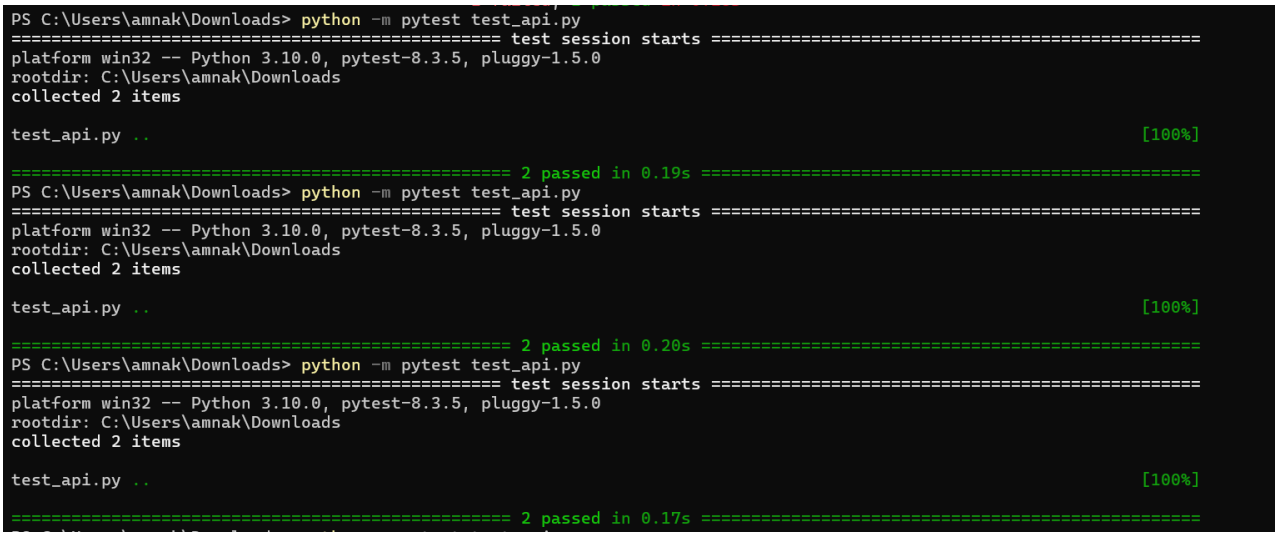
3.4 JSON Responses

Responses are structured with clear status messages and indicators despite using Protocol Buffers.

4 Testing Strategy

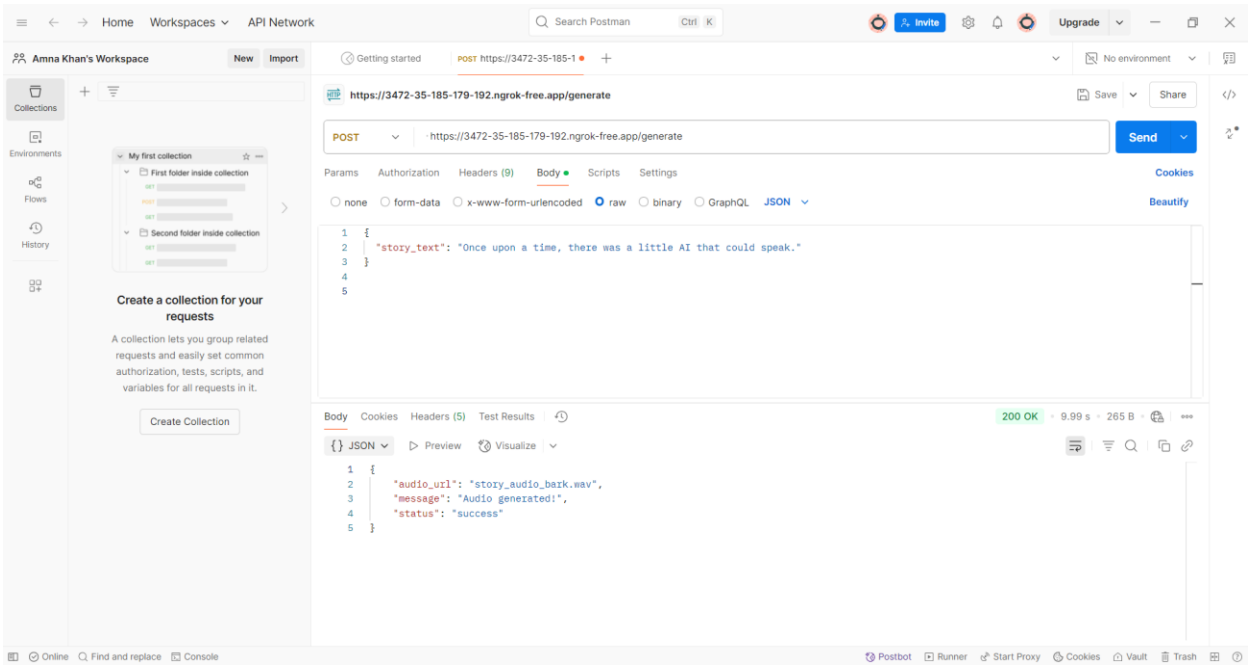
4.1 Unit Testing

i. `test_api.py`: Tests gRPC endpoint behavior.
ii. `test_tts.py`: Validates text-to-audio conversion.
iii. `test_async.py`: Tests asynchronous behavior.



4.2 Postman Testing

The testing checks if the service works well by sending both correct and incorrect requests. For correct inputs, it makes sure that the audio is generated properly, and the response includes the audio, status message, and the text used. For incorrect inputs like empty text or wrong voice names it checks that the service doesn't crash and shows a clear error message. This helps us know that the service works as expected and can handle mistakes in a safe way.



4.3 Performance Testing with Locust

Performance testing with Locust was done to check how well the service handles many users at the same time. It simulates multiple users using the system together to see how fast it responds and how many requests it can handle in a short time. This helps to find any slow parts or issues that could cause delays when the service is under heavy use. To see the evaluation report click:

file:///C:/Users/amnak/Downloads/Locust_2025-05-05-02h29_locustfile.py_https___415d-34-56-24-95.ngrok-free.app.html

5. Deployment

5.1 Dockerization

Docker was used to package the entire project into a container, making it easy to run anywhere without setting up the environment manually. The Dockerfile defines all necessary dependencies and setup steps. The project is built and run using the commands:

- docker build -t story2audio
- docker run -p 50051:50051 story2audio

5.2. Makefile

The project uses Makefile to simplify common development tasks through easy-to-run commands. make install installs all required Python dependencies, while make run-server starts the gRPC server. To ensure code correctness, make run-tests runs all test cases. They make download-models command preloads the necessary audio and emotion detection models. For gRPC setup, make build generates the required gRPC Python files from the proto definition. Finally, making docker-build creates a Docker image of the project, and make docker-run runs it inside a container.

6. Frontend Interface

The project includes a simple and user-friendly web interface built using Gradio. This interface allows users to input text, select a voice type from a dropdown menu, and play back the generated audio directly in the browser. It provides an intuitive way for users to interact with the text-to-audio service without needing to use gRPC or Postman, making it accessible even to non-technical users.

7. Conclusion

The Bark Text-to-Audio microservice meets all requirements. It provides a gRPC interface, supports concurrent processing, includes error handling, and offers a frontend. It is containerized for easy deployment and has been validated through