AI Cricket Commentary Playback: System Documentation and User Manual

System Documentation and Application Team

May 2025

Version 1.0

Contents

1 Introduction to System Documentation

This section provides a detailed explanation of the core models and technological components that constitute the AI Cricket Commentary Playback application. The primary goal is to fetch cricket match commentary data from the official IPL website (https://www.iplt20.com/) and present it as sequential audio playback using Text-to-Speech (TTS) synthesis. This documentation is intended for developers and technical users seeking to understand the system's architecture and components.

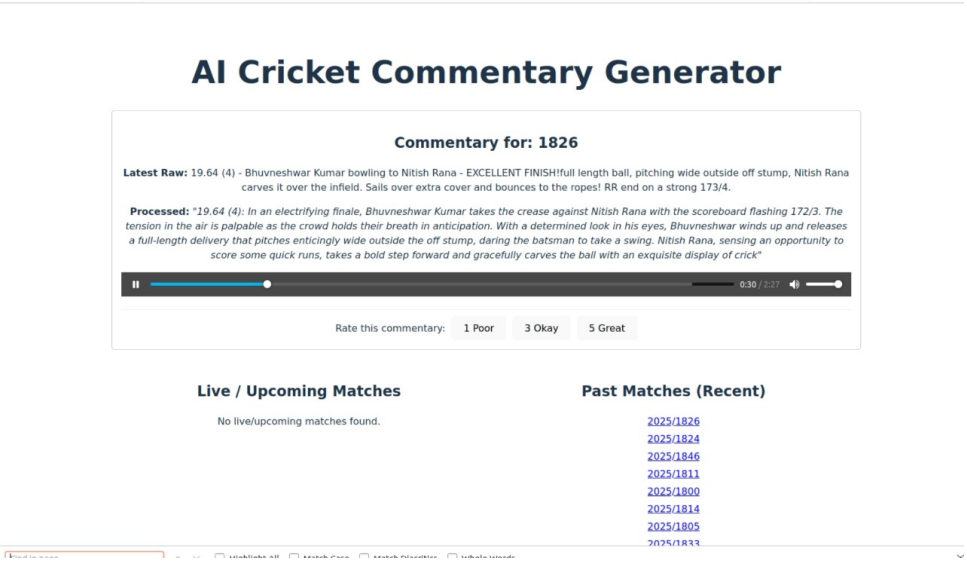

Figure 1: WOrking image of the application showing the transcript of the commentary and also the audio being played from the TTS model

1.1 Overview

The application integrates several key components:

- **Web Scraping Component:** Fetches match data and commentary text.
- **Large Language Model (LLM):** Loaded but not used in primary playback flow, intended for potential future enhancements.
- **Text-to-Speech (TTS) Model - Pre-trained:** Converts commentary text to audio.
- **Text-to-Speech (TTS) Model - Fine-tuned:** Optional for custom voice output.
- **Monitoring Components:** Provide operational visibility via Prometheus and Grafana.

2 Web Scraping Component

The web scraping component is the foundation for data acquisition, though it is not an AI model.

Technology: Python libraries `Selenium` (browser automation for JavaScript-heavy pages) and `BeautifulSoup4` (HTML parsing).

Purpose:
- Navigate to https://www.iplt20.com/.
- Extract lists of past and upcoming matches, including URLs and descriptions (e.g., "Team A vs Team B (Result/Date)") via `scrape_matches_with_details` in `scraper.py`.

- Extract ball-by-ball commentary for a selected match, handling dynamic content (scrolling, "Load More" buttons) via `scrape_full_match_commentary` in `scraper.py`.

Implementation File: `backend/scraper.py`

Key Challenge: Dependency on the website's HTML structure and CSS selectors, requiring maintenance if the site changes.

3 Large Language Model (LLM)

A Large Language Model is included but not used in the primary playback workflow due to latency concerns.

Model Used: `Mistral-7B-Instruct-v0.2` (GGUF quantized, e.g., `TheBloke/Mistral-7B-Instruct-v0.2-G` `Q4_K_M`).

Purpose (Potential):
- Rephrase commentary for naturalness.
- Answer match-related questions or generate summaries (not implemented).

Implementation Library: `llama-cpp-python` for efficient GGUF model inference.

Configuration:
- `LLM_MODEL_PATH`: Path to `.gguf` file.
- `N_GPU_LAYERS`: Layers offloaded to GPU.
- `N_CTX`: Context window size.
- `chat_format`: Set to `"mistral-instruct"`.

Current Usage: Loaded at startup but bypassed by the `/synthesize` endpoint.

4 Text-to-Speech (TTS) Model

The TTS model converts commentary text into audible speech.

4.1 Pre-trained Model (Default)

Model Used: `tts_models/multilingual/multi-dataset/xtts_v2` (Coqui TTS).

Technology: XTTS v2, a multilingual voice cloning and synthesis model.

Purpose: Generate WAV audio files from commentary text.

Implementation Library: `TTS` (Coqui AI).

Configuration: `TTS_MODEL_NAME` specifies the model.

Current Usage: Used by `/synthesize` endpoint, runs on CPU (`gpu=False`).

4.2 Fine-tuned TTS Model (Optional)

Model Used: Fine-tuned `Coqui XTTS v2` on custom audio data.

Purpose: Mimic specific voices.

Process (External): Collect audio, segment, transcribe, format, fine-tune using Coqui TTS scripts on a GPU machine.

Implementation Files: `best_model.pth`, `config.json`.

Configuration: Set `TTS_MODEL_PATH` and `TTS_CONFIG_PATH` in `.env`.

Current Usage: Loaded if paths are set.

5 Monitoring Components

Prometheus: (http://localhost:9090) Time-series database for metrics.

Grafana: (http://localhost:3000) Visualization of metrics.

Node Exporter: Collects host metrics.

Backend Instrumentation: Exposes custom metrics via `/metrics`.

6 System Documentation Conclusion

The application integrates web scraping, a pre-trained TTS model, and monitoring. The LLM is loaded for potential enhancements, and fine-tuned TTS models are supported. Data acquisition reliability depends on the IPL website's stability.

7 User Manual Introduction

Welcome to the AI Cricket Commentary Playback application! This user manual guides end-users on how to set up, use, and troubleshoot the application. It enables users to select past IPL matches and listen to simulated ball-by-ball audio commentary generated via TTS.

7.1 Application Overview

The application:

- Displays recent past IPL matches with descriptions.
- Allows selection of a match for commentary playback.
- Scrapes ball-by-ball commentary from https://www.iplt20.com/.
- Uses AI TTS to read commentary sequentially.
- Offers audio controls (Play/Pause, Volume).

8 System Requirements

- **Docker Desktop / Docker Engine:** Must be installed and running.
- **Web Browser:** Chrome, Firefox, Edge, or Safari.
- **Internet Connection:** Required for fetching match data and downloading models initially.
- **Hugging Face CLI (for initial model download):**

```
pip install huggingface-hub
```

- **Node.js and npm (for frontend setup):** Node.js version 18 (LTS) or newer recommended.

9 Getting Started: Initial Setup Launching

9.1 Prerequisites (One-time Setup)

1. **Navigate to Project Directory:** Open your terminal/command prompt and go to the main folder where you extracted the application files (the one containing the `docker-compose.yml` file).

```
cd /path/to/ai-cricket-commentary
```

2. **Download LLM Model Files:** The application requires a Large Language Model (LLM) file. The recommended model is `Mistral-7B-Instruct-v0.2`. If you do not have it, download it using the Hugging Face CLI.

   (a) Create a directory for models if it doesn't exist:

```
mkdir -p models
```

   (b) Download the model (this file is approx. 4.37 GB):

```
huggingface-cli download TheBloke/Mistral-7B-Instruct-v0.2-GGUF
    mistral-7b-instruct-v0.2.Q4_K_M.gguf --local-dir ./models --
    local-dir-use-symlinks False
```

Note: Ensure `LLM_MODEL_PATH` *in your* `.env` *file (see next step) points to this downloaded file, e.g.,* `/app/models/mistral-7b-instruct-v0.2.Q4_K_M.gguf`

3. **Configure Environment Variables (`.env` file):** Create a file named `.env` in the project root directory. Copy and paste the following content, adjusting `N_GPU_LAYERS` based on your VRAM (start with 0 for CPU only if unsure):

```
# LLM Configuration
LLM_MODEL_PATH=/app/models/mistral-7b-instruct-v0.2.Q4_K_M.gguf
N_GPU_LAYERS=0
N_CTX=4096

# TTS Configuration
TTS_MODEL_NAME=tts_models/multilingual/multi-dataset/xtts_v2
TTS_MODEL_PATH=
TTS_CONFIG_PATH=
```

Listing 1: .env file content

4. **Install Frontend Dependencies:** The frontend user interface requires Node.js packages.
   (a) Navigate to the frontend directory:

```
cd frontend
```

   (b) Install dependencies:

```
npm install
```

   (c) Navigate back to the project root:

```
cd ..
```

9.2 Launching the Application

Once prerequisites are met:

5. **Run Docker Compose:** In the project root directory, execute:

```
sudo docker compose up --build
```

(Use `sudo` *if required.* `-build` *is recommended first time.)*

6. **Monitor Startup Logs:** Wait for the backend to finish initializing components. Look for messages like:

```
INFO: uvicorn.error: Uvicorn running on http://0.0.0.0:8000
INFO: uvicorn.error: Application startup complete.
```

The first time you run with a new `TTS_MODEL_NAME`, Coqui TTS will download the model files (e.g., XTTS v2 is approx. 2GB), so startup might take longer.

7. **Access the Application:** Open your web browser and navigate to:

http://localhost

10 Using the Application

1. **Main Interface:** Displays app title and match lists.
2. **Match Lists:** Shows "Live / Upcoming Matches" and "Past Matches (Recent)" with descriptions like "CSK vs PBKS (Result)". If scraping fails, lists may be empty.
3. **Selecting a Match:** Click a match under "Past Matches (Recent)".
4. **Starting Playback:** Shows "Fetching commentary...". Scraping the full match commentary can take **30 seconds to several minutes**. Once fetched, the "Now Playing Commentary" section appears, and playback of the first line should begin.
5. **Playback Controls:** Standard audio player for Play/Pause, Volume. Commentary advances line by line. Autoplay may be blocked by browsers; click play if needed.
6. **Selecting Another Match:** Stops current playback, fetches new commentary.
7. **Feedback (Optional):** Rate lines via buttons (1–5).

11 Troubleshooting

- **Application Not Loading:** Check Docker status, Docker Compose logs (`sudo docker compose logs -f`), and port conflicts (80, 8000, 9090, 3000).
- **No Matches Listed:** Scraper selectors in `backend/scraper.py` likely outdated due to changes on `iplt20.com` (requires developer update). Check internet and backend logs.
- **Error Fetching Commentary List:** Specific match page scraper failed (website changes or no data).
- **No Audio Playback:** Check system volume, click play button in UI. Check backend logs for "TTS synthesis" errors.
- **Slow Playback:** Scraping and TTS processing are time-consuming.

12 Stopping the Application

1. Return to the terminal where `docker compose up` is running.
2. Press `Ctrl + C` (once or twice).
3. Clean up:

```
sudo docker compose down
```

13 User Manual Conclusion

This manual provides instructions for setting up and using the AI Cricket Commentary Playback application. The system's functionality relies on external website stability for data scraping.

— End of User Manual —