Pipeline Performance & Design Report

Date: **July 24, 2025**

Author: **santhoshkumar**

Project: **India Speaks - Indic-Language Preprocessing Pipeline**

1. Executive Summary

This report details the design, implementation, and performance of the india_speaks_cleaner pipeline. The objective was to process a raw dataset of 2,000 multilingual audio-text pairs, validate their integrity, normalize transcriptions according to internal standards, and produce a clean dataset ready for ASR/TTS model training.

The pipeline successfully processed the initial dataset, yielding 1,298 clean, train-ready utterances and rejecting 702 utterances due to specific quality control failures. This automated process has significantly improved the quality and consistency of the data that will be fed into our machine learning models.

2. Pipeline Performance Analysis

The pipeline's primary function is to act as a quality gate. The analysis of the rejected data provides valuable insights into the quality of the raw data collection.

Initial Dataset Size: 2,000 utterances Accepted Utterances: 1,298 (64.9%) Rejected Utterances: 702 (35.1%)

Breakdown of Rejection Reasons:

The rejected.csv file reveals the primary reasons for data exclusion:

- LOW_QUALITY_FLAG (702 rows): This was the sole reason for rejection in the provided output. 100% of the rejected files (35.1% of the total dataset) were discarded because they were explicitly marked with quality_flag = 0 by crowd workers. This indicates a significant issue in the raw data collection phase that needs to be addressed.
- Other Potential Reasons: While not present in this run, the pipeline is equipped to handle other issues like missing transcriptions, invalid audio paths, and out-of-range durations. The dominance of the LOW_QUALITY_FLAG suggests it is the most immediate problem to solve in our data sourcing.

3. Normalization Verification

The text normalization module successfully cleaned the transcriptions for the accepted data, as seen in the train_ready.csv file.

Example 1: Hindi Normalization

- Language: hi
- Raw Transcription: naam hai kaise
- Normalized Transcription: nam hai kaise
- Analysis: The pipeline correctly applied the language-specific rule to normalize the common transliteration aa to a, ensuring phonemic consistency.

Example 2: Generic Cleaning (English)

- Language: en
- Raw Transcription: swagat ho aaya dilli
- Normalized Transcription: swagat ho aaya dilli
- Analysis: The script correctly lowercased the text and verified that it contained only the characters permitted by the IndiaSpeaks_Data_Standards.pdf.

4. Design Decisions

The pipeline's architecture was designed for scalability, maintainability, and ease of use.

- Modular Package Structure: The code is organized into a reusable Python package with separate modules for validation (validation.py), text processing (text_normalizer.py), and orchestration (cleaner.py), allowing for easy updates to specific components.
- Configuration-Driven: Key parameters like duration limits and allowed punctuation are stored in an external config.yaml file. This allows for adjustments to the pipeline's behavior without altering the source code.
- CLI Entry Point: The script is executed via a command-line interface (main.py), making it simple to integrate into automated scripts and larger MLOps workflows.

5. Known Limitations & Future Work

While effective, the pipeline has areas for future enhancement:

- Advanced Normalization: The current normalization rules are foundational. For production, more complex, language-specific rules (e.g., handling of conjuncts in Bengali, chillu characters in Malayalam) should be implemented as detailed in the data standards document.
- ASR-Based Mismatch Detection: The pipeline currently relies on human-provided quality flags. The next iteration should incorporate a pre-trained multilingual ASR model to calculate a Word Error Rate (WER) between the ASR's prediction and the provided transcript. A high WER would serve as a powerful, automated flag for mismatched audio-text pairs.

- Data Collection Feedback Loop: The insights from the rejected.csv file should be used to create a feedback loop with the data collection team. The high number of LOW_QUALITY_FLAG rejections is a clear action item to improve crowd-worker guidelines and review processes.