MetaVoice- Take Home Task

Design Document

**Prepared By**: Arjuna Pandian Nehru

**Prepared on**: 27/10/2023

**Mobile**: 07957577948

**Email**: arjunapandian@hotmail.com

**Overview**

The MetaVoice Audio Processing Pipeline is a comprehensive system designed to process audio files stored in an S3 bucket. It employs various operations, including transcription, tokenization, preprocessing, and summary generation. This pipeline is constructed using a combination of Python, essential libraries (e.g., Pandas, PyTorch), Apache Airflow, and AWS services.

**Components**

1. **audiopipeline.py**

This Python script constitutes the heart of the audio processing pipeline. It carries out the following tasks:

Transcription and Tokenization:

* + Employs the Whisper library to transcribe audio files.
  + Tokenizes the audio data.

Preprocessing:

* + Converts audio files to a standardized format (WAV) to ensure uniform processing.

File Information Retrieval:

* + Retrieves file size and creation date for each processed audio file, as these attributes are expected to be valuable for downstream machine learning tasks.

Summary Generation:

* + Composes statistics on the processed data, encompassing total files processed, average transcription length, total token count, file size, and creation date.

Logging:

* + Leverages the Python logging module to chronicle various stages of the pipeline's execution.

**2. lambda\_function.py**

This script serves as the entry point for executing the audio processing pipeline in an AWS Lambda function. It calls the `main()` function from `audiopipeline.py`.

**3. metavoice\_dag.py**

This DAG (Directed Acyclic Graph) file outlines the workflow for running the pipeline using Apache Airflow. It encompasses a task called `execute\_pipeline`, which triggers the `main()` function from `audiopipeline.py`.

**4. config.json**

This JSON file holds configuration parameters vital to the pipeline, including AWS credentials, S3 bucket details, file paths, and valid audio file extensions.

**5. requirements.txt**

This file enumerates all the Python libraries and packages required to run the pipeline.

**Execution Flow**

**1. Lambda Function Trigger -** An event instigates the Lambda function, which subsequently executes the audio processing pipeline.

**2. Pipeline Initialization -** `audiopipeline.py` reads the configuration from `config.json` and sets up logging.

**3. S3 File Processing Loop:**

- Cycles through objects in the specified S3 bucket.

- Checks if the file extension is in the list of valid audio extensions.

**4. Audio File Download and Processing:**

- Retrieves the file from S3 to a local staging folder.

- Preprocesses the audio file (conversion to WAV format).

**5. Transcription and Tokenization:**

- Utilizes the Whisper library to perform transcription.

- Generates random tokenized audio data (for demonstration purposes).

**6. File Information Retrieval:**

- Retrieves file size and creation date.

**7. Data Collection:**

- Aggregates processed data for further analysis.

**8. Parquet File Generation:**

- Organizes processed data into partitioned Parquet files.

**9. Temporary Folder Cleanup:**

- Removes the local staging folder after processing is complete.

**10. Summary Generation:**

- Computes summary statistics based on the processed data.

**11. Airflow DAG Execution:**

- The Airflow DAG, `metavoice\_pipeline\_dag`, is scheduled to run once a day.

**12. Airflow Task Execution:**

- The `execute\_pipeline` task in the DAG triggers the execution of the audio processing pipeline.

**Data Storage**

Processed audio data and summary statistics are stored in partitioned Parquet files in the specified S3 bucket.

**Error Handling**

Exceptions are caught and logged at different stages of the pipeline (transcription, preprocessing, file operations, etc.). If an error occurs, it is logged, and the pipeline raises an exception.

**Scalability**

The pipeline can be scaled by adjusting the configuration settings, using AWS services like S3 for storage, and leveraging the distributed processing capabilities of Apache Spark through PySpark.

**Monitoring and Logging**

The pipeline uses the Python logging module to log information, warnings, and errors at various stages of execution. Logs are written to the specified log file.

**Dependencies**

The pipeline relies on external libraries and packages, which are listed in `requirements.txt`. These dependencies include libraries for audio processing, data manipulation, and interactions with AWS services.

**Suggestions for Futuristic Works**

1. **Sentiment Analysis for Transcriptions -** Integrate sentiment analysis to automatically assess the emotional tone of transcriptions. This could provide valuable insights for understanding user sentiment in audio data.

1. **Speaker Identification -** Implement speaker identification algorithms to differentiate between multiple speakers in a single audio file. This feature would be particularly useful for multi-party conversations or interviews.
2. **Keyword Extraction and Categorization -** Enhance the pipeline to identify and categorize key topics or keywords within transcriptions. This can help in content tagging, indexing, and facilitating search functionality.
3. **Integration with Machine Learning Models -** Incorporate machine learning models for tasks like automatic speech recognition (ASR) and natural language processing (NLP). This could lead to significant improvements in transcription accuracy and language understanding.

**5. Real-time Processing and Streaming -** Develop capabilities for real-time audio processing and streaming. This would enable the pipeline to handle live audio feeds, making it suitable for applications like call centers, live event coverage, and voice assistants.

**Conclusion**

The MetaVoice Audio Processing Pipeline is a versatile and scalable system designed to efficiently process audio data. It leverages a combination of Python libraries, Apache Airflow, and AWS services to automate the processing workflow. The pipeline can be further extended and optimized to meet specific requirements and handle large-scale audio processing tasks.