An Orchestrated Framework for Automated Speech Data Processing and Alignment

Quoc-Khanh NGUYEN
VinBigData, VinGroup
Ha Noi, Vietnam
v.khanhnq35@vinbigdata.org

Van-Huy NGUYEN
VinBigData, VinGroup
Thai Nguyen University of Technology, Vietnam
Ha Noi, Vietnam
huynguyen@tnut.edu.vn

Van-Tuan PHAN
VinBigData, VinGroup
Ha Noi, Vietnam
v.tuanpv32@vinbigdata.org

Tri-Nhan DO
VinBigData, VinGroup
Ha Noi, Vietnam
v.nhandt23@vinbigdata.org

Dang-Khoa MAC
VinBigData, VinGroup
Ha Noi, Vietnam
v.khoamd@vinbigdata.org

***Abstract*—The creation of large-scale, diverse speech datasets, crucial for state-of-the-art Automatic Speech Recognition (ASR), remains a significant bottleneck. This paper introduces a novel, orchestrated multi-pipeline framework designed to fully automate this process, from YouTube content discovery to the generation of phonetically aligned data suitable for ASR training. Our integrated system seamlessly combines key components: a configurable data crawler equipped with robust proxy and cookie management for efficient content acquisition; a neural processing pipeline incorporating Voice Activity Detection (VAD), ASR, speaker diarization, and automated quality assessment; and a specialized pronunciation alignment system leveraging the Montreal Forced Aligner (MFA) to produce precise word-level timing annotations. Implemented as containerized services managed by an Apache Airflow orchestration framework, the system achieves remarkable efficiency and scalability. Demonstrating its capabilities, the framework processed over 1000 hours of initial Vietnamese YouTube audio, yielding 813 hours of high-quality, aligned data with an end-to-end processing throughput exceeding 4.x times real-time and achieving 98% automation across the workflow. This represents a significant reduction in manual effort compared to traditional methods, enabling systematic quality control through integrated filtering mechanisms. The architecture's inherent modularity and scalability make it readily adaptable to various languages and extendable beyond ASR to other audio-based machine learning applications.**

***Index Terms*—Speech automated workflows, Pronunciation alignment, Speech automated annotation, Standardizing data, Data collection**

I. Introduction

Modern Automatic Speech Recognition (ASR) systems require large and diverse datasets for training. The demand for large-scale, diverse speech datasets continues to grow, driving the development of automated collection and processing frameworks, with recent efforts culminating in datasets exceeding 100,000 hours for tasks like speech generation [1]. While public datasets exist, they often lack domain-specific content, language diversity, or sufficient volume for building robust models. YouTube represents an enormous repository of spoken content across languages, topics, and acoustic conditions, making it an invaluable source for speech data. However, harnessing this content poses significant technical challenges, including efficiently searching and downloading relevant content, extracting clean audio segments, producing accurate transcriptions and labels, and creating precise word-level alignments.

Previous approaches to YouTube data collection for ASR have typically involved manual or semi-automated processes that are time-consuming, inconsistent, and difficult to scale [2], [3]. Furthermore, the quality of collected data often varies significantly, requiring extensive post-processing or manual verification. The lack of standardized pipelines for this task has hindered reproducibility and systematic improvements in dataset creation methodology.

In this paper, we present an end-to-end system that automates the entire workflow from YouTube content discovery to the production of labeled and phonetically aligned speech data suitable for ASR model training. Our pipeline consists of three major components:

1) **YouTube Data Crawler**: An automated pipeline for searching and downloading content from YouTube using flexible filtering criteria.
2) **Audio Labeling Pipeline**: A modular workflow that processes the collected audio through VAD, ASR, alignment, speaker diarization, and quality scoring.
3) **Pronunciation Alignment Pipeline**: A specialized pipeline that performs forced alignment between audio and text transcripts at the word level using the Montreal Forced Aligner (MFA).

All components are implemented as containerized services orchestrated by Apache Airflow, allowing for parallel processing, robust error handling, and easy deployment across computing environments. Our approach prioritizes automation, scalability, and data quality, addressing key challenges in speech dataset creation.

The main contributions of this paper are:

- A fully automated, configurable pipeline for collecting domain-specific speech data from YouTube
- A modular audio processing pipeline that generates aligned transcriptions and speaker labels
- A pronunciation alignment pipeline that creates precise word-level timing information
- An architecture that enables systematic quality filtering of speech segments
- A containerized implementation that ensures reproducibility and scalability

The remainder of this paper is organized as follows: Section II reviews related work in the fields of web data crawling, audio labeling, and forced alignment. Section III details the system architecture of all three components. Section IV describes the implementation details and key algorithms. Section V discusses experimental results and performance metrics. Section VI concludes with limitations and future work.

II. Related Work

A. Web Data Crawling for ASR

Creating large-scale speech datasets is crucial for deep learning ASR, leveraging web data's value. Recent efforts like GigaSpeech 2 [4] demonstrate automated pipelines for crawling and refining unlabeled YouTube audio for low-resource ASR, reducing manual verification. Prior works collected YouTube data using custom scripts for low-resource languages [2] or specific domains like egocentric interactions [3], potentially lacking our work's comprehensive orchestration. Community-driven efforts like Common Voice [5], while valuable, require significant coordination and differ in scale from automated crawling. Specifically for Vietnamese speaker recognition, Vietnam-Celeb [6] utilized YouTube/TikTok with visual-aided processing for speaker label accuracy, contrasting with our focus on an orchestrated, audio-based pipeline for generating aligned ASR data.

B. Audio Processing and Labeling

Automated audio labeling is key; Park et al. [7] showed pretrained ASR models effective for pseudo-labeling. While VAD is a critical preprocessing step, advancements address complex tasks like overlapping speech segmentation, exemplified by Bredin and Laurent's [8] end-to-end model using the pyannote.audio toolkit. Our work integrates such components within its pipeline.

C. Forced Alignment and Pronunciation Modeling

Forced alignment maps text to audio segments. Montreal Forced Aligner (MFA) [9] is a prominent HMM-based tool using Kaldi. More recently, deep learning approaches like CTC-segmentation [10] have shown improved performance. For phonologically complex languages like Vietnamese, accurate G2P is crucial, demonstrated by its role in systems like the TTS solution by Do et al. [11]. Our work implements an automated Vietnamese forced alignment pipeline.

D. Workflow Orchestration for ML Pipelines

Frameworks like Apache Airflow [12] (DAG-based), Luigi [13], and Kubeflow [14] (Kubernetes-native) orchestrate ML workflows. Our work uniquely combines crawling, comprehensive audio processing, and pronunciation alignment within a unified orchestration framework enabling repeatability, scalability, and quality control.

III. System Architecture

Our system consists of three main components: the YouTube Data Crawler, the Audio Labeling Pipeline, and the Pronunciation Alignment Pipeline. All are designed with modularity, scalability, and automation as primary considerations, using Apache Airflow for workflow orchestration and Docker for containerization.

A. YouTube Data Crawler

The crawler component is responsible for searching YouTube, collecting metadata, and downloading relevant audio content. It features a modular architecture with several key components, as illustrated in Fig. 1.

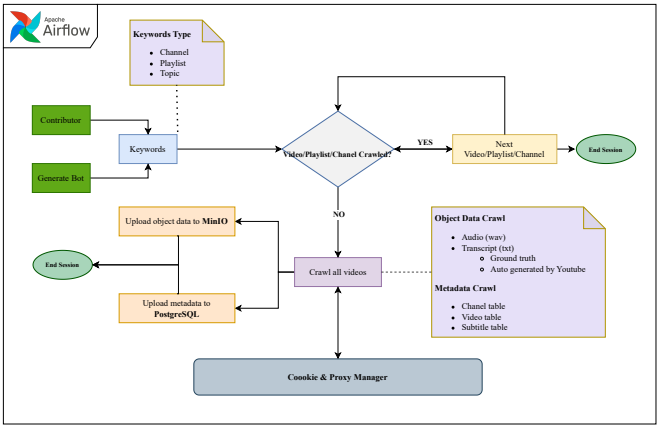

Fig. 1. Architecture of the YouTube Data Crawler pipeline (Component A).

1) Crawler Component: The core crawler module searches YouTube based on various criteria specified by the user, such as keywords, channel IDs, playlist IDs, or specific video IDs. Additional parameters allow filtering by language, region, and content category. The crawler uses asynchronous processing with Python's asyncio to efficiently handle network requests. Crucially, it incorporates modules for managing YouTube Cookies and Proxies to handle potential access restrictions and enhance crawling robustness.

Additional parameters allow filtering by language, region, and content category. The crawler uses asynchronous processing with Python's asyncio to efficiently handle network requests. Crucially, it incorporates modules for managing YouTube Cookies and Proxies to handle potential access restrictions and enhance crawling robustness.

2) Audio Downloader: This component extracts audio streams from the videos identified by the crawler. It uses the yt-dlp library, which provides robust handling of YouTube's changing interface and format options. The downloader supports parallel processing through a configurable number of jobs.

3) Post-Processor: The post-processor handles audio format normalization, converting downloaded audio to a standardized format (24kHz, mono, 16-bit PCM). This standardization ensures compatibility with downstream processing tools and models in the labeling pipeline.

4) Storage Components: The crawler utilizes two primary storage mechanisms:

- PostgreSQL Database: Stores structured metadata about channels, videos, and subtitles.
- MinIO Object Storage: Houses the actual audio files and potentially subtitle/transcript files.

5) Workflow Orchestration: Apache Airflow manages the execution sequence of the crawling pipeline. A typical workflow involves executing the crawler to gather video metadata while utilizing cookie and proxy management, uploading this metadata to PostgreSQL, downloading the audio corresponding to identified videos, performing post-processing on the audio files, and finally uploading the processed files to MinIO storage. This workflow can be triggered manually or scheduled for periodic execution, enabling continuous data collection.

The workflow can be triggered manually or scheduled to run periodically, allowing for continuous data collection over time.

B. Audio Labeling Pipeline

The labeling pipeline processes the collected audio to produce structured data suitable for training ASR models. Its architecture is illustrated in Fig. 2.

1) File Retrieval: This component identifies and retrieves audio files from MinIO storage based on configurable criteria such as bucket, prefix, and file extension. It produces a list of files to be processed by subsequent components.

2) Voice Activity Detection (VAD): The VAD module identifies segments of audio containing speech, filtering out silence, music, and noise. It utilizes deep learning models, likely based on pyannote.audio, and outputs timestamp information for speech segments. Model inference is accelerated using NVIDIA Triton Inference Server, allowing for GPU-based processing.

3) Automatic Speech Recognition (ASR): This component transcribes the speech content in the audio files. It interfaces with a pre-trained ASR model served via NVIDIA Triton, supporting various model types as indicated by the "generic" parameter in the configuration. The output is raw text transcription.

4) Alignment: The alignment module combines the outputs from VAD and ASR to produce time-aligned transcripts, typically in formats like SRT or JSON. This alignment enables precise mapping between text and audio timestamps, which is essential for training and evaluating ASR models.

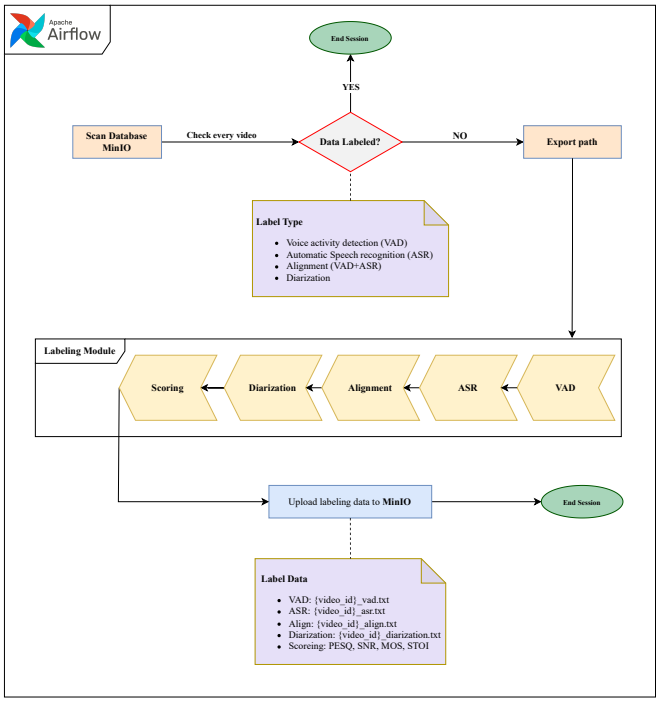

Fig. 2. Architecture of the Audio Labeling pipeline (Component B).

5) Speaker Diarization: This component identifies speaker turns within the audio, answering "who spoke when." It likely uses neural diarization approaches based on pyannote.audio or similar frameworks, with model inference potentially accelerated through Triton.

6) Segment Scoring: The scoring module evaluates various quality metrics for the speech segments, potentially including signal-to-noise ratio, Mean Opinion Score estimates (via distillmos), and other acoustic quality measures. These scores enable filtering of low-quality segments that might degrade ASR model training.

7) MinIO Upload: This final component collects all generated artifacts—VAD results, ASR transcripts, alignments, diarization outputs, and quality scores—and uploads them to a structured location in MinIO storage. The organization preserves the relationship between these artifacts and the source audio file.

8) Workflow Orchestration: Apache Airflow orchestrates the labeling pipeline, with the Celery Executor enabling distributed task processing across multiple workers. This approach allows for parallel processing of audio files, significantly improving throughput for large datasets.

C. Pronunciation Alignment Pipeline

The pronunciation alignment pipeline builds upon the outputs of the previous components to create precise word-level alignments between audio and text. Its architecture is illustrated in Fig. 3.

1) Pronunciation Dictionary Initialization: This component loads and prepares the Vietnamese pronunciation dictio-

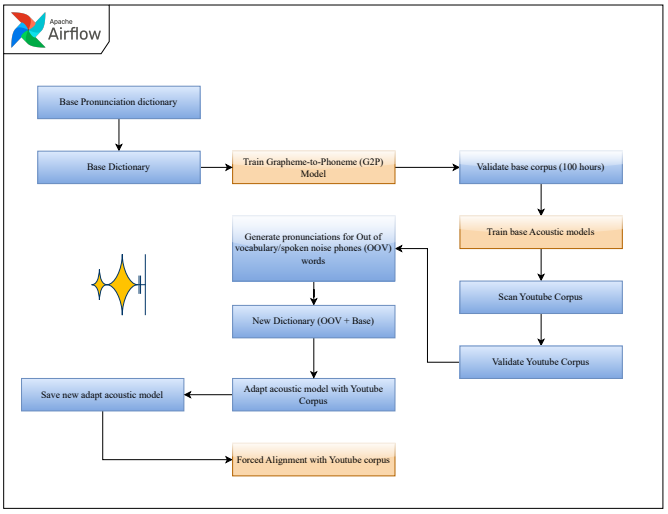

Fig. 3. Architecture of the Pronunciation Alignment Pipeline using Montreal Forced Aligner (Component C).

nary that maps words to their phonetic representations. The dictionary serves as a foundation for both G2P model training and acoustic model training.

2) Grapheme-to-Phoneme (G2P) Model Training: The G2P model component creates a statistical model that can predict pronunciations for words not found in the dictionary. This is crucial for handling out-of-vocabulary (OOV) words encountered in YouTube transcripts.

3) Base Acoustic Model Training: This component trains the initial acoustic model using a curated dataset (set1) with known high-quality audio and transcripts. The acoustic model maps audio features to phonetic units and serves as the foundation for the alignment process.

4) YouTube Data Preparation: This module retrieves the audio files and transcripts from MinIO storage and prepares them in a format suitable for the Montreal Forced Aligner. It creates a specific directory structure and generates transcript (.lab) files from the JSON data.

5) Out-of-Vocabulary (OOV) Word Processing: This component identifies words in the YouTube transcripts that are not present in the pronunciation dictionary. It then uses the G2P model to generate pronunciations for these words and adds them to an expanded dictionary.

6) Forced Alignment: The core alignment component uses the acoustic model and expanded dictionary to precisely align each word in the transcript with its corresponding segment in the audio file. The output is in TextGrid format, containing detailed timing information for each word.

7) Result Conversion: This final component converts the TextGrid files to a more readable text format that includes timing information alongside the transcribed words. These files are then stored for downstream applications.

8) Workflow Orchestration: Apache Airflow orchestrates the entire alignment process, managing dependencies between tasks and ensuring efficient resource utilization. The workflow is designed to handle errors gracefully and continue processing when possible.

IV. Implementation

A. YouTube Data Crawler Implementation

1) Search and Metadata Collection: The crawler implementation leverages multiple Python libraries for YouTube interaction:

- Youtube-python for searching videos based on keywords
- yt-dlp for extracting video metadata and downloading content

It also integrates libraries for managing cookies (e.g., loading from browsers or files) and rotating proxies to ensure continuous access.

The crawling process utilizes asynchronous implementation, which improves throughput by allowing multiple concurrent API requests. The crawler collects comprehensive metadata including video/channel info, duration, and subtitle availability. This information is stored in JSON files and uploaded to a PostgreSQL database.

2) Database Schema: The PostgreSQL database schema includes tables for `channels`, `videos` (linked to channels), and `subtitles`. This relational structure enables efficient querying and filtering.

3) Audio Processing: For audio download, yt-dlp was configured with parallel processing support. Post-processing employed ffmpeg/sox tools to standardize audio formats (to 24kHz, mono, 16-bit PCM), ensuring compatibility and reducing storage requirements before upload to MinIO.

4) Airflow DAG Implementation: The crawler workflow is implemented as an Airflow Directed Acyclic Graph (DAG) with tasks corresponding to the components described above:

```
metadata_crawling >> transcript_crawling
>> database_upload >> audio_download
>> audio_processing >> minio_upload
```

The DAG can be configured through Airflow variables, allowing flexible adaptation to different collection requirements without code modification.

B. Audio Labeling Pipeline Implementation

1) Model Integration: VAD, ASR, and Diarization models (configurable type, likely based on pyannote.audio) were integrated by serving them via NVIDIA Triton. Communication utilized the tritonclient[grpc] library for efficient, GPU-accelerated inference.

2) Parallel Processing: Implements file-level parallelism (Airflow Celery Executor distributing tasks) and GPU batch processing within components (VAD, ASR, Diarization) for efficiency on large datasets.

3) Quality Scoring: Evaluates segment quality using metrics such as Signal-to-Noise Ratio (SNR), Mean Opinion Score (MOS) estimates (e.g., via ‘distillmos‘), Short-Time Objective Intelligibility (STOI), and Perceptual Evaluation of Speech Quality (PESQ) for downstream filtering of low-quality segments.

4) Airflow DAG Implementation: The labeling workflow was implemented as an Airflow DAG with tasks structured sequentially: data preparation, Voice Activity Detection (VAD), Automatic Speech Recognition (ASR), Speaker Diarization, text alignment, quality scoring, and final MinIO upload of all generated results.

C. Pronunciation Alignment Pipeline Implementation

1) Montreal Forced Aligner Integration: The Montreal Forced Aligner (MFA) was integrated using a dedicated Docker environment, with Miniconda managing specific dependencies (e.g., Python version). Bash scripts handled environment switching, and specific directory structures were employed to organize MFA corpora according to its requirements.

2) Pronunciation Dictionary and G2P Model: A text file served as the Vietnamese pronunciation dictionary, mapping words to phonemes. Grapheme-to-Phoneme (G2P) model training utilized MFA's built-in tools based on this dictionary to predict pronunciations for OOV words found in YouTube transcripts.

3) Acoustic Model Training: Base acoustic model training was performed using standard MFA procedures on a curated dataset, leveraging MFA's options for parallel processing and data cleaning steps.

4) OOV Word Processing: Out-of-Vocabulary (OOV) word processing involved identifying words in the input transcripts absent from the pronunciation dictionary, predicting pronunciations using the trained G2P model, and incorporating these into an expanded dictionary used for alignment.

5) Forced Alignment: Forced alignment was executed using MFA's parallel alignment functions ('mfa align'), leveraging the trained acoustic model and the expanded dictionary to generate TextGrid outputs containing precise word timings.

6) Airflow DAG Implementation: The pronunciation alignment sequence was defined in an Airflow DAG with the following primary task order: Grapheme-to-Phoneme (G2P) model training, corpus validation, base acoustic model training, preparation of YouTube data, processing of Out-of-Vocabulary (OOV) words, and finally, the forced alignment process itself.

V. Evaluation and Results

A. YouTube Data Crawler Performance

1) Crawling Efficiency: We evaluated the crawler's efficiency across different search types and parameters. Table I presents the average processing times and success rates.

The asynchronous implementation significantly improved throughput compared to synchronous approaches, with up to 8x speedup for keyword searches that require multiple API requests.

2) Data Volume and Quality: Using the crawler, we collected datasets across multiple languages and content categories. Table II summarizes the volume and characteristics of the collected data.

The crawler successfully retrieved diverse content with varying characteristics, demonstrating its flexibility. Subtitle availability varied significantly across content categories, highlighting the importance of the subsequent labeling pipeline for generating consistent annotations.

TABLE I
YouTube Data Crawler Performance Metrics

Search Type	Avg. Processing Time (s/video)	Success Rate (%)	Metadata Completeness (%)
Keyword (Top 100)	2.3	94.2	98.1
Channel ID	1.8	97.6	95.4
Playlist ID	1.5	98.9	97.7
Video ID	1.2	99.5	100

TABLE II
Dataset Collection Statistics

Content Category	Number of Videos	Hours of Audio	Subtitle Availability (%)
Conversation of channels	67,261	16,502	100
Audiobooks	5,503	8,061	100
Conversation of playlist	1,745	1,078	100

B. Audio Labeling Pipeline Performance

1) Processing Throughput: We evaluated the labeling pipeline's throughput under different worker configurations. With 8 GPU-equipped workers, the system processed approximately 4 hours of audio per hour of wall clock time, representing a 4x real-time factor. This performance enables rapid processing of large datasets within reasonable timeframes.

The ASR component represents the most computationally intensive step, utilizing the highest proportion of GPU resources. However, its parallel implementation ensures that it does not become a significant bottleneck in the overall pipeline.

The quality metrics indicate that the pipeline produces labels of sufficient quality for many ASR training applications, particularly when combined with the quality scoring component to filter out lower-quality segments.

C. Pronunciation Alignment Pipeline Performance

1) Processing Throughput: We evaluated the pronunciation alignment pipeline's throughput on a representative subset of Vietnamese YouTube data. Acoustic model training is the most time-consuming component but reusable for multiple alignment tasks. The overall end-to-end alignment throughput was approx. 2x real-time. (Table III shows estimated times for 100h).

D. End-to-End System Performance

Evaluation demonstrates that our automated system significantly reduces dataset creation time, achieving 98% automation across the entire workflow, from data collection through processing and pronunciation alignment.

TABLE III
ESTIMATED PRONUNCIATION ALIGNMENT COMPONENT PROCESSING TIMES FOR 100 HOURS OF AUDIO

Component	Est. Processing Time (hours)	Throughput (hours audio/hour)
G2P Base Model Training	0.1	N/A
Acoustic Base Model Training	9.4	3.2
OOV Word Processing	11.6	N/A
Forced Alignment	14.3	7.0
Total Pipeline (E2E)	50.0	2.0

Table IV summarizes the end-to-end processing statistics for a pipeline run initiated with 1000 hours of successfully crawled YouTube audio data.

TABLE IV
END-TO-END PROCESSING STATISTICS (STARTING WITH 1000HR CRAWLED DATA)

Stage	Input Data Volume	Output Data Volume	Processing Time (hrs)	Automation Level (%)
YT Data Crawling	N/A	1,000 hrs	32	95
Audio Labeling	1,000 hrs	920 hrs	63	100
Pronunciation Align	920 hrs	813 hrs	80	100
Total Pipeline	N/A	813 hrs	175	98

The high level of automation across all components demonstrates the system's efficiency in creating large-scale speech datasets with minimal human intervention.

VI. CONCLUSION AND FUTURE WORK

Conclusion. Addressing the critical need for large-scale, diverse datasets for data-hungry Automatic Speech Recognition (ASR) systems, this paper presented a comprehensively automated and orchestrated framework for acquiring and processing speech data from web sources like YouTube. The system demonstrated its capability to handle significant data volumes, yielding high-quality, phonetically aligned speech data suitable for ASR training through efficient processing and a high degree of automation across the entire workflow, significantly minimizing manual effort. Integrating essential components for automated annotation—including voice activity detection, ASR transcription, and speaker diarization—along with a robust pipeline for precise word-level pronunciation alignments, the framework is built for modularity and scalability. Its adaptable architecture inherently supports potential multilingual applications. By drastically reducing manual intervention and enforcing data standardization, this work provides a powerful, automated tool for efficiently constructing the extensive, diverse datasets essential for advancing state-of-the-art ASR and other speech-related machine learning tasks.

Future Work. Despite its effectiveness, the system relies on platform APIs requiring maintenance and its output quality depends on the underlying pre-trained models. Scalability for datasets significantly exceeding 10,000 hours and handling highly overlapped speech remain challenges. The pronunciation alignment component also faces resource constraints and requires language-specific adaptation. Future directions include integrating active learning for targeted quality improvement, expanding to other video platforms, enhancing filtering techniques, developing domain adaptation methods, and creating a user-friendly interface. We also aim to extend the pronunciation alignment pipeline to more languages, explore end-to-end neural alignment, develop external APIs, improve alignment throughput via distributed processing, and release an open-source version of the framework to benefit the research community.

REFERENCES

[1] H. He, Z. Shang, C. Wang, X. Li, Y. Gu, H. Hua, L. Liu, C. Yang, J. Li, P. Shi, Y. Wang, K. Chen, P. Zhang, and Z. Wu, "Emilia: An extensive, multilingual, and diverse speech dataset for large-scale speech generation," in *2024 IEEE Spoken Language Technology Workshop (SLT)*, 2024, pp. 885–890.

[2] H. Liao, E. McDermott, and A. Senior, "Large scale deep neural network acoustic modeling with semi-supervised training data for youtube video transcription," in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, 2013, pp. 368–373.

[3] D. Hernandez, J. Hoffman, and T. Darrell, "Collecting large-scale egocentric data for long-term interaction modeling," 2018. [Online]. Available: https://arxiv.org/abs/1805.04699

[4] Y. Yang, Z. Song, J. Zhuo, M. Cui, J. Li, B. Yang, Y. Du, Z. Ma, X. Liu, Z. Wang, K. Li, S. Fan, K. Yu, W.-Q. Zhang, G. Chen, and X. Chen, "Gigaspeech 2: An evolving, large-scale and multi-domain asr corpus for low-resource languages with automated crawling, transcription and refinement," 2024. [Online]. Available: https://arxiv.org/abs/2406.11546

[5] R. Ardila, M. Branson, K. Davis, M. Henretty, M. Kohler, J. Meyer, R. Morais, L. Saunders, F. M. Tyers, and G. Weber, "Common voice: A massively-multilingual speech corpus," 2019. [Online]. Available: https://arxiv.org/abs/1912.06670

[6] V. T. Pham, X. T. H. Nguyen, V. Hoang, and T. T. T. Nguyen, "Vietnam-celeb: a large-scale dataset for vietnamese speaker recognition," in *Interspeech 2023*, 2023, pp. 1918–1922.

[7] D. S. Park, Y. Zhang, Y. Jia, W. Han, C.-C. Chiu, B. Li, Y. Wu, and Q. V. Le, "Improved noisy student training for automatic speech recognition," 2020. [Online]. Available: https://arxiv.org/abs/2005.09629

[8] H. Bredin, R. Yin, J. M. Coria, G. Gelly, P. Korshunov, M. Lavechin, D. Fustes, H. Titeux, W. Bouaziz, and M.-P. Gill, "Pyannote.audio: Neural building blocks for speaker diarization," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 7124–7128.

[9] M. McAuliffe, M. Socolof, S. Mihuc, M. Wagner, and M. Sonderegger, "Montreal Forced Aligner: Trainable Text-Speech Alignment Using Kaldi," in *Proceedings of Interspeech 2017*, 2017, pp. 498–502. [Online]. Available: https://doi.org/10.21437/Interspeech.2017-1386

[10] L. Kürzinger, D. Winkelbauer, T. Watzel, D. Müller, N. Volland, A. beğen, M. Zager, R. Schlüter, and H. Ney, "CTC-Segmentation of large corpora for german end-to-end speech recognition," 2020. [Online]. Available: https://arxiv.org/abs/2007.09127

[11] V. Huy Nguyen, "An end-to-end model for vietnamese speech recognition," in *2019 IEEE-RIVF International Conference on Computing and Communication Technologies (RIVF)*, 2019, pp. 1–6.

[12] M. Beauchemin. (2015) Airflow: A workflow management platform. The Airbnb Tech Blog. [Online]. Available: https://medium.com/airbnb-engineering/airflow-a-workflow-management-platform-46318b977fd8

[13] Spotify. Luigi: Luigi is a Python module that helps you build complex pipelines of batch jobs. GitHub.

[14] E. Bisong, "Kubeflow: A machine learning toolkit for Kubernetes," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*. Apress, 2019.