VOXBLINK-CN: A LARGE-SCALE CHINESE AUDIO-VISUAL SPEAKER VERIFICATION DATASET

by

Xingyu Shen

Signature Work Product, in partial fulfillment of the
Duke Kunshan University Undergraduate Degree Program

Mar 25, 2024

Signature Work Program
Duke Kunshan University

APPROVALS

Mentor: Prof. Ming Li, Division of Natural and Applied Sciences

Marcia B. France, Dean of Undergraduate Studies

CONTENTS

ABSTRACT

In this paper, we will introduce VoxBlink-CN, a large-scale Chinese audio-visual speaker verification dataset, comprising over 41,739 speakers. The dataset addresses critical limitations inherent in existing Chinese audio datasets, such as the restricted number of speakers, labeling inaccuracies, and challenges in achieving effective generalization across diverse audio-visual contexts. By leveraging an automated pipeline for efficient data extraction from Bilibili—a famous Chinese video-sharing platform—we significantly expanded the diversity and volume of Chinese audio data available for research. Our methodology involves the development of a novel video processing pipeline capable of extracting speaker-specific clips from extensive video materials, ensuring high precision and diversity in dataset composition. The dataset has a great potential to dramatically enhance model performance in Chinese speaker verification tasks, underscoring its considerable scale and scope compared to existing datasets. The comprehensive design of the pipeline also guarantees both efficiency and stability. VoxBlink-CN's development reflects a significant advancement in the resources available for Chinese language audio-visual research, offering a robust foundation for future explorations in speech recognition, speaker verification, and multimodal machine learning applications.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Prof. Ming Li, Yuke Lin, and other research assistants in the SMIIP Lab for their support and collaboration throughout this project. Their insights and feedback have been invaluable in shaping the direction of the research.

This project was funded in part by the National Natural Science Foundation of China (Grant No. 62171207) and the DKU Undergraduate Studies Office through the Summer Research Scholars program.

Additionally, I am thankful to DKU IT for their assistance and the computational resources provided by the Advanced Computing East China Sub-Center, which greatly facilitated the execution of this project.

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

In AI projects, having a substantial amount of accurate data is crucial, especially with the current trend of training large models. This necessity is particularly evident in the realm of speech, where despite the existence of powerful models, the scarcity of extensive, high-precision, and diverse datasets remains a significant constraint. This limitation is especially pronounced in Chinese audio datasets. The Wespeaker [22], a famous open-source speaker embedding learning toolkit, has done rigorous testing using two prominent datasets: CNCeleb [12] and VoxCeleb [14]. CNCeleb and VoxCeleb are widely utilized by researchers and are considered representative datasets for Chinese and English audio respectively. However, if we compare the performance of Wespeaker trained on CN-Celeb and Vox-Celeb datasets as shown in Figure 1.1, the performance gap is striking.

Through analysis, we find a key factor contributing to the disparity is that CN-Celeb lacks adequate speaker and diversity, and suffers from imprecise labeling. Thus, this project aims to develop a Chinese audio dataset to elevate model performance to that of English datasets. The dataset should have the following characteristics:

- **High Precision:** Exclusive inclusion of the target speaker's voice ensures data precision.
- **Large Scale:** Encompassing a large number of distinct speakers and a great many video clips per speaker.
- **Diversity:** Containing speakers from varied backgrounds and characteristics. Each speaker's video content should unfold chronologically to introduce temporal shifts, environmental variations, and contextual changes, thereby enhancing dataset generality.

Table 1. Results achieved using different architectures on the VoxCeleb dataset, "dev" of part 2 is used as the training set

Architecture	voxceleb1_O		voxceleb1_E		voxceleb1_H	
	EER(%)	minDCF	EER(%)	minDCF	EER(%)	minDCF
TDNN	1.590	0.166	1.641	0.170	2.726	0.248
ECAPA-TDNN ([4])	0.870	0.107	1.120	0.132	2.120	0.210
ECAPA-TDNN	0.728	0.099	0.929	0.100	1.721	0.169
ResNet34 ([3])	1.31	0.154	1.38	0.163	2.50	0.233
ResNet34	0.723	0.069	0.867	0.097	1.532	0.146
ResNet221	0.505	0.045	0.676	0.067	1.213	0.111
ResNet293	**0.447**	**0.043**	**0.657**	**0.066**	**1.183**	**0.111**

Table 2. Results on the CNCeleb evaluation set

Architecture	EER(%)	minDCF
TDNN	8.960	0.446
ECAPA-TDNN	7.395	0.372
ResNet34([16])	9.141	0.463
ResNet34	6.492	0.354
ResNet221	**5.655**	**0.330**

Figure 1.1: Performance of Wespeaker trained on VoxCeleb and CNCeleb [22].

- **Inclusion of Audio-Visual Information**: Integration of both audio and visual data enables the exploration of multi-modal approaches.

To construct this database, we've devised an automated pipeline tailored to retrieve videos from Bilibili and process them efficiently. Presently, 162,076 videos have been procured from Bilibili, yielding 41,739 distinct individuals post-processing. This project remains ongoing, with plans for further expansion on both scales. More tests and tasks will be carried out based on this dataset.

This paper will first introduce the pipelines and tricks we developed to create the dataset in the Material and Methods section. Then, in the Results section, we will show some statistics about the dataset we currently have. In the Discussion section, we will compare the pipelines we created, analyze the obtained data, delve into the challenges encountered in pipeline design and future prospects.

Chapter 2

MATERIAL AND METHODS

In this section, I will introduce several automated pipelines designed to obtain and process videos.

Functionally, the pipeline can be divided into two parts: data downloading and data processing. Firstly, we download videos from the famous Chinese video platform Bilibili and then cut them into segments, selecting those that meet our criteria.

Upon completing the dataset, if necessary, we will only disclose the video link information along with our speaker label information and the audio and video data itself will not be disclosed. This disclosure is strictly for non-commercial, pure research purposes. Users are advised to make independent decisions regarding the download of audio and video data based on their local laws and regulations. And it is the user's responsibility to ensure that their data downloading practices comply with legal and regulatory requirements.

2.1 Data Downloading

2.1.1 Utilizing ChromeDriver

The first method we used to get data from Bilibili is to utilize the ChromeDriver [3]. It acts as a bridge between the Google Chrome browser and the automation pipeline, allowing us to control browser operations programmatically. With the ChromeDriver, we could simulate user browsing actions on Bilibili, such as video filtering and user ranking. The entire downloading process is shown in Figure 2.1.

In our pipeline, we first leverage ChatGPT [16] to generate keywords essential for user search. Specifically, I designed some prompts for ChatGPT, prompting it to furnish

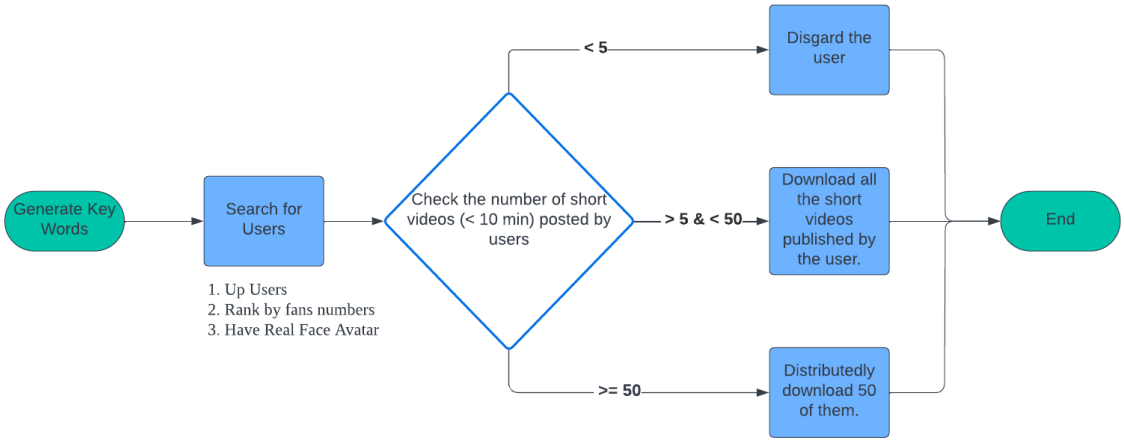

Figure 2.1: The pipeline of utilizing ChromeDriver to retrieve and download data from Bilibili.

commonly occurring keywords in user nicknames. These prompts encompass requests such as 'Provide 100 common Chinese surnames,' 'List 20 prevalent professions,' and 'Identify 20 frequently mentioned animals'. Subsequently, we manually filtered these generated keywords aimed at enhancing program efficiency. This process involves eliminating uncommon keywords found in user identifiers. For example, certain keywords, such as 'King of Glory,' 'Dota,' 'PUBG,' and other game-related terms, are intentionally excluded. This exclusion is motivated by the observation that users who share gaming content typically showcase only their desktop during video recording, rendering facial analysis infeasible during subsequent video processing stages.

After compiling the video keyword list, ChromeDriver is employed to launch the browser and input the keywords into the search bar for exploration. A filter criterion is applied to target Up Users. Up user is a title given by Bilibili to a user who are usually more professional video creators and are more likely to possess advanced video and audio recording setups. This strategic selection enhances the likelihood of gathering higher-quality videos, consequently elevating the standard of the audio-visual data stored in the dataset.

Given that Bilibili restricts search results to a maximum of 42 pages, each page containing 16 users, yielding a total of 672 users, our focus is to identify individuals who have uploaded a substantial volume of top-tier videos. To achieve this, we implemented a filter to arrange the Up Users results retrieved from Bilibili based on the number of followers. Additionally, we conducted a preliminary assessment of the user's avatar, utilizing a lightweight facial recognition technique, CascadeClassifier in the OpenCV package [17]. Only users with authentic facial avatars progressed to subsequent video download procedures. This meticulous screening process is crucial, as the avatars

of Up Users will play a pivotal role in the subsequent stages of video processing, as elucidated in Section 2.2.1. Further details on this aspect will be discussed in the subsequent section.

After confirming the users, the ChromeDriver navigates to these qualified profiles to filter the videos they have posted. Our preference is to gather video clips from various videos rather than acquiring multiple clips from a single source. This is because the diversification of the dataset is essential to introduce temporal variations, environmental changes, and contextual diversity. To achieve this, we let users with fewer than 5 short videos be discarded outright. For those with 5 to 50 short videos, we download all available content. For users exceeding 50 videos, we ranked the videos by their uploading dates and tried to download a distributed selection of up to 50 videos. With this method, the downloaded videos would be spread out through a timeline. Also, we set the criterion for video filtering to within 10 minutes and exclusively download short videos, considering both download speed and storage constraints.

Upon successful downloading, the pipeline gathers metadata for both videos and users, encompassing upload timestamps, durations, titles, avatars, nicknames, and more. These metadata are invaluable for subsequent video processing stages and for crafting challenges based on the dataset.

2.1.2 Utilizing Bilibili API

An API, or Application Programming Interface, serves as a framework defining how software components interact. It acts as a bridge between different systems, enabling data exchange and function invocation. Within this pipeline, as depicted in Figure 2.2, we leverage the Bilibili API [15], specifically a Python-based library facilitating access to various Bilibili APIs, to acquire video metadata and perform video downloads.

Broadly speaking, the video download process via the Bilibili API is similar to the method outlined earlier using the ChromeDriver, as both are designed based on the idea of simulating user actions on the Bilibili. But notably, certain requests made through the Bilibili API require the inclusion of the requester's cookie information. Cookies, small text files transmitted from web servers to browsers and stored on users' computers, are commonly employed for tracking, identifying, and storing website preferences, session data, and other pertinent information. Valid cookie information is imperative for gaining access to certain data on Bilibili.

In a bid to enhance the dataset's breadth and download efficiency, we've made adjustments to the pipeline and incorporated optimizations tailored to the Bilibili API. Initially, we relaxed the requirement mandating users to have profile pictures featuring real human faces during the download phase. Recognizing that Bilibili's user demo-

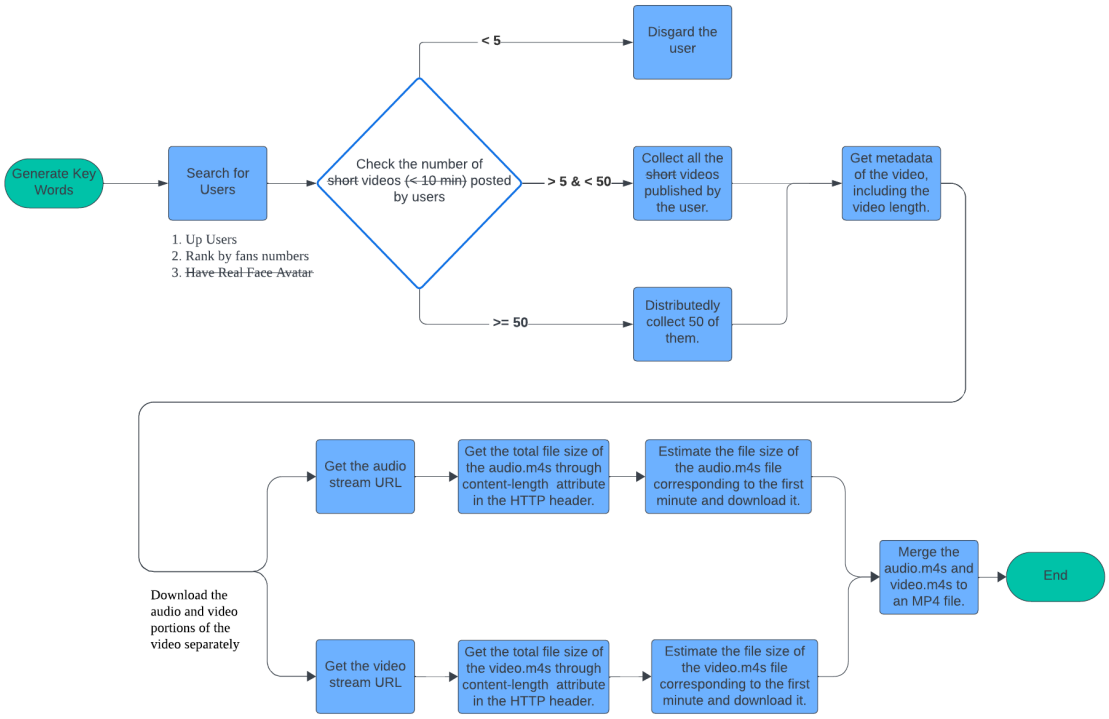

Figure 2.2: The pipeline for downloading the videos through Bilibili-API.

graphic often includes younger individuals who frequently employ anime characters or celebrity images as profile pictures. We also observed a prevalence of creatively altered photos generated by AI tools, such as cartoon renditions based on personal photographs. Thus, integrating a facial recognition system into the pipeline to exclude users lacking authentic human face avatars would inadvertently exclude a substantial subset possessing eligible videos. However, this adjustment also means the video processing approach introduced in Section 2.2.1 would not work anymore in this case, thus prompting the development of a new video processing framework elaborated in Section 2.2.2.

We have also refined our approach to filtering videos, moving away from restricting downloads to short videos under 10 minutes. Instead, our focus now only lies on ensuring the dispersion of videos based on their publication time. We made this adjustment since we observed that during the download process, a significant portion of videos surpassed the 10-minute limit yet still contained video fragments that meet our criteria. For instance, educational lectures often consist of uninterrupted segments featuring the speaker directly addressing the camera, aligning perfectly with our requirements. However, the duration of a single lecture may extend up to 40 minutes, resulting in the exclusion of these valuable video resources. Since we also want to contain these videos in our dataset, we have eliminated the length restriction during downloads.

However, this modification introduces a new challenge: longer videos mean larger video file sizes, which consume more download time and the network bandwidth, and thus slow down program execution. These larger video files would also occupy additional storage space. Moreover, the subsequent video processing stage demands substantial computational resources. We estimated the processing time even on the Sugon supercomputer cluster [20] for an average video length of approximately 10 minutes and found that processing such a vast dataset would span several years. Thus, a balance between video length and practical operability is imperative.

Furthermore, echoing the sentiment expressed in Section 2.1.1, our aim is to foster dataset diversity by prioritizing video clips from various sources over accumulating numerous clips from a single source. In this context, the original length of the video becomes inconsequential. Our observation reveals that most users typically introduce their content at the beginning, and a significant portion of these introductions are suitable for inclusion in the dataset as valid audio segments. Consequently, we have tailored the download process to extract only the initial minute of content from longer videos, aligning with our goal of enhancing dataset diversity while optimizing storage and processing efficiency.

To download only the video segments of the first 1 minute of the video, the pipeline will separately process the m4s files of audio and video. The size of a one-minute video segment is obtained through the following formula:

$$\text{Target\_video\_size} = \text{total\_video\_size} \times \frac{\text{total\_video\_length}}{\text{target\_video\_length}} + \text{buffer\_size} \tag{2.1}$$

In the equation, the *total_video_size* can be obtained from the *content−length* item in the HTTP response header section, indicating the total number of bytes in the entire file. The metadata of videos on Bilibili includes the video duration, which can be converted from the format of *hours* : *minutes* : *seconds* into seconds to obtain *total_video_length*. The *target_video_length* is set to 60 seconds, representing the desired length of the video segment. Additionally, we include a $buffer\_size$ parameter to ensure that all desired content is downloaded. In the actual download process, we conservatively set this $buffer\_size$ to 200,000 bytes. Once we have an estimated size for the video segments, the pipeline downloads 1024 bytes at a time for both audio and video. The downloaded bytes are compared with the *target_video_size*, and the download stops when the number of downloaded bytes exceeds this value. After obtaining the audio and video segment files separately, the widely-used video processing package, FFmpeg [6], is employed to merge them. For videos shorter than one minute, all segments are retained; for longer videos, only the first minute of video content is retained.

2.1.3 Acceleration Methods

We also tried and used many tricks to improve video download speed while maintaining the stability of the pipeline.

Multiprocessing

The first method we utilized to accelerate the downloading process is multiprocessing [8], assigning each thread to handle a distinct keyword for searching and processing on Bilibili concurrently. Multiprocessing enables simultaneous execution of multiple processes within a program, each with its own memory space. This approach enhances system performance and efficiency by facilitating diverse tasks to run concurrently. Such parallel processing is particularly advantageous in scenarios with extensive data processing requirements.

Within Python, the multiprocessing module facilitates the creation and supervision of multiple processes. This module streamlines the implementation of parallel computation and task management in Python programs. Unlike multithreading, multiprocessing offers superior parallelism due to its utilization of separate global interpreter locks (GILs) for each process. This feature allows tasks to execute with complete parallelism across multi-core CPUs, enhancing overall performance significantly.

Asynchronous code

Another optimization we employed was rewriting certain pipeline steps using the asyncio package [7] to introduce asynchronous code. Asynchronous operations are typically managed by an event loop. When an asynchronous operation concludes, it notifies the event loop, which subsequently adds the corresponding callback function to the task queue for execution. Asynchronous code enhances resource utilization, particularly for handling I/O-intensive tasks, thus significantly boosting performance. Unlike synchronous code, where program execution halts until a specific operation completes, asynchronous code allows program execution to continue while awaiting subsequent code execution.

For instance, certain steps such as downloading the 'video.m4s' and 'audio.m4s' files mentioned in Section 2.2 can be made asynchronous, enabling these time-consuming processes to occur concurrently. However, it is essential to note that not all steps can be converted to asynchronous code, especially if subsequent steps rely on the outcomes of previous ones. For example, downloading 'video.m4s' and 'audio.m4s' cannot occur asynchronously with the merge step responsible for combining these files into the final 'mp4' file. Nevertheless, overall, this approach significantly enhances the efficiency of the pipeline.

In comparison to the original method, this optimized pipeline, incorporating several strategies, significantly enhances the downloading speed. Nonetheless, it introduces new challenges, such as the risk of the associated account being flagged for excessive access requests, potentially leading to a temporary block. While implementing a waiting period using time.sleep package [18] between steps could mitigate this issue, it compromises download speed, which is contrary to our objectives. Therefore, alternative approaches must be explored to maintain high-speed operation while guaranteeing stability and uninterrupted program execution. Fortunately, Bilibili's account bans are typically temporary, lasting only a few hours. This allows us ample opportunity to experiment with various adjustments, comprehend the underlying mechanisms of Bilibili's account management, fine-tune our program, and ensure consistent and reliable performance.

Proxy

Initially, we experimented with utilizing diverse proxies to facilitate our requests. Proxies, acting as intermediaries between clients and target servers, intercept client requests, transmit them to the target server, and relay the server's responses back to the client.

There are numerous free proxy IP lists available online, such as hidemy [9] and ihuan [10]. We curated a collection of these available proxy server IP addresses, assessing their connectivity to Bilibili. Following this, we meticulously tallied the successfully connected IP addresses, organizing them into a pool based on stability. Despite our efforts, these IPs exhibited a lack of stability, achieving only approximately a 50% success rate. To mitigate this, we implemented a waiting period during IP connection attempts. Should no response occur within the stipulated timeframe, we defaulted to using our lab server's IP to access Bilibili. Nevertheless, this approach proved to be inadequately reliable, the unstable connections caused a lot of timeout. Extending the timeout duration or incorporating waits directly into the program would significantly impede execution speed.

Subsequently, we redirected our focus towards professional proxy service providers. Following a thorough comparison of various suppliers, we opted for Kuai Proxy [11], renowned for its diverse range of proxy services encompassing HTTP, HTTPS, SOCKS, and more. After testing, we found the connections established through Kuai Proxy to Bilibili boasted an impressive success rate exceeding 90%, coupled with commendable response speeds. We managed to integrate Kuai Proxy's API into our pipeline, enabling dynamic proxy IP switching during program execution. However, operational challenges arose as the accounts associated with cookie information frequently signaled rapid IP address changes, resulting in account suspensions. Consequently, we

were compelled to discontinue this approach.

Cookie

As previously discussed, Bilibili scrutinizes the cookie data within incoming requests to determine the information to provide in the response message. To mitigate the risk of account blocking due to frequent request transmissions, we devised a strategy to reduce request frequency within a specified timeframe by altering the cookie information. This necessitated the establishment of an extensive cookie repository, from which randomized cookie data could be retrieved for request transmission. To assemble this repository, a survey was devised and sent among Bilibili users. In the survey, we include detailed survey content, encompassing explanations on cookies, collection methodology, and instructions tailored to various browsers such as Google Chrome, Safari, Edge, and Firefox, along with illustrative images.

Over 40 sets of cookie data were amassed, sufficient to sustain 80 concurrent program operations. Nevertheless, there persisted a risk of account blockages for participants. While users could promptly unblock their accounts via mobile devices, this inconvenience was deemed undesirable. Furthermore, unblocking accounts rendered the original cookie data obsolete, requiring the respondents to provide the cookie information again.

After running this method for a while, we found that the SESSDATA parameter within the cookie data is the only attribute that affects Bilibili's authentication process in the requests we send. Notably, the SESSDATA acquired from the same Bilibili account logging in through different browsers is different. Thus, instead of collecting cookie information from 40 different people, I only enlisted assistance from family members to install multiple browsers on their devices, encompassing popular options like 360 Speed Browser, Maxthon Browser, DuckDuckGo, QQBrowser, Opera, alongside the previously mentioned major browsers. This approach facilitated the collection of ample cookie data solely through our Bilibili accounts, enabling unhindered program execution. Subsequently, when cookies expired, I could independently update the parameter information without inconveniencing others.

In theory, with an abundant supply of cookie data, coupled with adequate CPU resources and network bandwidth, download speeds could be augmented limitlessly. And there is no need to be concerned about program instability due to excessive speed anymore. However, in response to concerns raised by DKU's IT department regarding potential network congestion, we were advised to restrict downloads to nighttime hours and moderate download speeds. From the perspective of our pipeline program, this directive underscores the attainment of maximum achievable download speeds.

2.2 Data Processing

After downloading the videos, the subsequent task involves processing them to extract only the segments where the target speaker is speaking. In this regard, we devised two distinct methods for video processing.

2.2.1 VoxBlink's method

The first method we tried was to adopt the video processing pipeline used to create the VoxBlink dataset [13], which was used to extract video clips from short videos downloaded from YouTube. The specific workflow is as shown in the Figure 2.3:

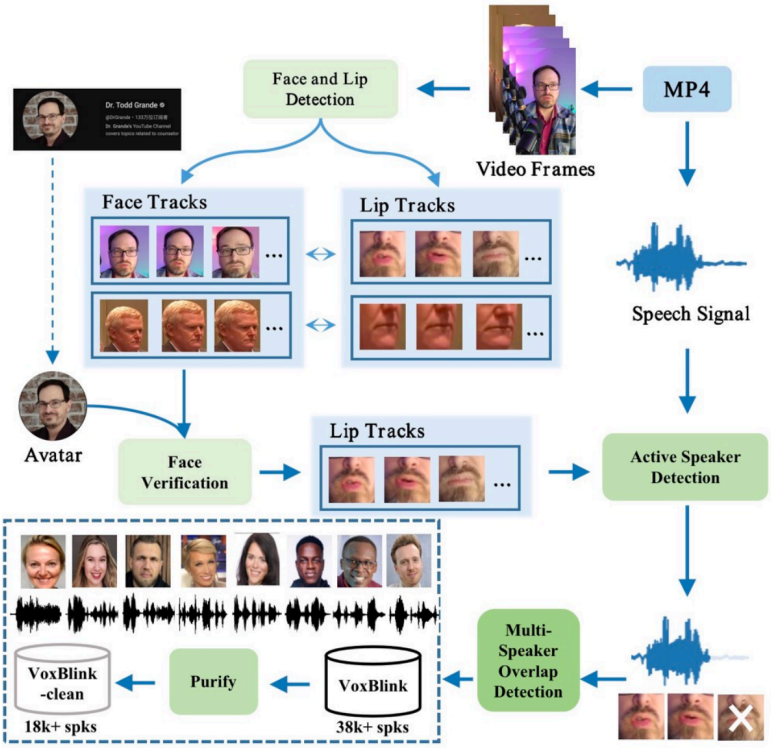

Figure 2.3: The automatic pipeline for the VoxBlink dataset.[13]

Upon completing video downloading, we employ scene detection algorithms based on average light changes to segment individual videos into multiple parts, a process akin to data augmentation. Subsequently, each video segment undergoes facial and lip movement detection using the Retina Face model [5]. We tuned the threshold for the minimum intersection over union (IOU) value between consecutive detections to ensure that each trajectory contains only one facial or lip sequence. A higher threshold leads to looser tracking, which may merge different facial or lip sequences into the same tracking sequence, while a lower threshold may result in stricter tracking, potentially causing discontinuities or missed detections in tracking. However, setting

a higher IOU also means that more frames are detected, increasing computational load and CPU burden, thereby slowing down program execution. Therefore, we initially processed a small number of videos and selected appropriate thresholds based on program performance.

Utilizing the Retina Face model, we generate facial and lip trajectories for each segment after obtaining facial and lip trajectories. Then, the ResNet-IRSE50 model [4] extracts facial embeddings frame by frame from these trajectories. Cosine similarity with template embeddings derived from avatars is computed to identify the target speaker's speech, with non-target trajectories discarded based on the average cosine similarity score.

In the Active Speaker Detection step, we employ a Seq2Seq audio-visual speaker diarization model [2] which utilizes both lip movements and audio clues to identify active speech within trajectories, filtering out silent or voice-over sections.

Lastly, Multi-Speaker Overlap Detection is performed to mitigate interference from overlapping speech data and improve speech segment quality. We utilize a conformer-based overlap speech detection (OSD) tool [1], alongside filtering out speeches with durations less than one second for the final extracted video segments.

With this comprehensive pipeline, we successfully extract video clips containing only the target speaker's speech.

2.2.2 Clustering

One significant limitation in the video processing pipeline mentioned in Section 2.2.1 is the requirement for the user to possess an avatar for identifying the target speaker. However, in practice, users may not necessarily utilize their own faces as avatars. Additionally, numerous video creators of Bilibili solely record their voices without appearing in the video. For instance, some gaming Up Users merely record their computer screens along with their voices, while others, such as food Up Users, primarily focus the camera on their cooking process, providing audio explanations without their faces showing up in the camera directly. Given our aim to maximize the dataset's size, a distinct pipeline must be devised.

Nevertheless, we posit that the voice of the video creator, who is the target speaker, likely constitutes the majority of all the voices heard in the video. And these video creator's voices are more likely to be heard in different videos uploaded by them. Building on this premise, we propose a clustering-based video processing pipeline. This pipeline autonomously identifies the target speaker first and subsequently extracts video clips based on their voice features. The specific workflow is depicted in Figure

2.4.

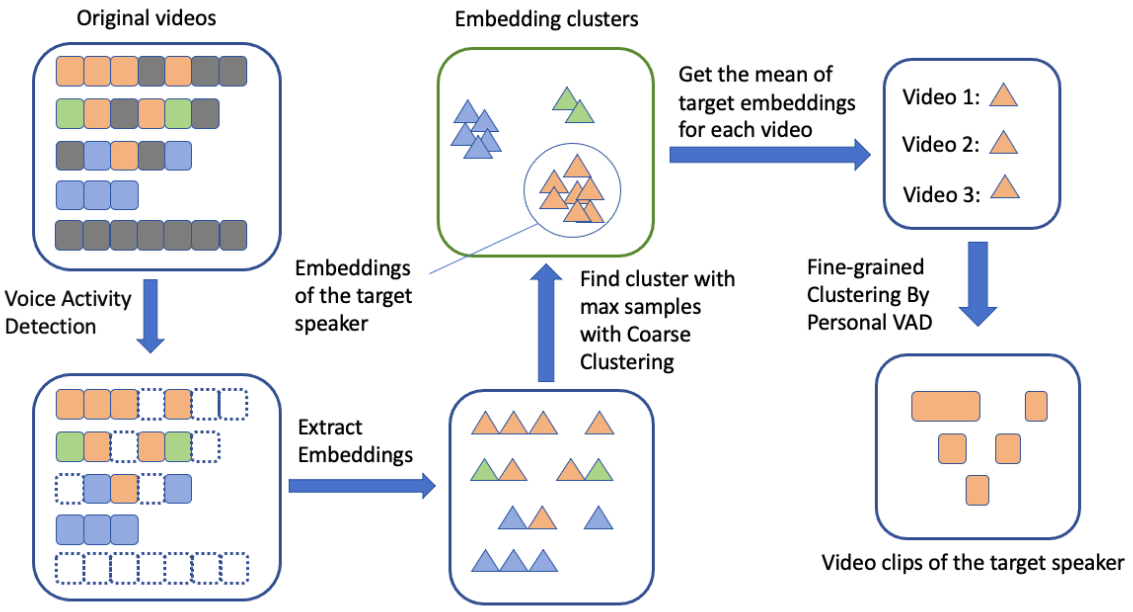

Figure 2.4: The automatic pipeline based on clustering.

For an Up User, their content typically comprises segments with no voice, segments with the voice of the Up User, and occasionally includes guest appearances, introducing voices of other individuals. To isolate segments containing only the video voice of the Up User, the pipeline initially has to identify the video creator' s voice.

The first step involves segmenting the video into 4-second clips, followed by employing voice activity detection (VAD) to discern speech activity within the extracted audio data, thereby eliminating segments devoid of voice. Subsequently, for segments containing voice, a voiceprint recognition model processes the audio data to extract voiceprint feature vectors. Given that the voice of the same speaker exhibits relatively minor variations, their corresponding voiceprint feature vectors tend to be more similar. Thus, coarse clustering is performed on the extracted voiceprint features, grouping them based on similarity. The group with the most voice feature samples corresponds to the voice feature of the target speaker.

It is noteworthy that we opt for a 4-second window size for video segmentation, representing a relatively coarse partitioning. This choice ensures segment processing quality while reducing computational complexity to expedite target speaker identification. Similarly, coarse clustering [19] is selected as the clustering method, operating on the principle of partitioning a dataset into preliminary clusters based on a simplified or rough approximation of the underlying data structure. This approach facilitates swift and efficient dataset partitioning into manageable clusters without necessitating de-

tailed computations or fine distinctions between data points.

However, this preliminary partitioning is inherently coarse. Following the acquisition of the voiceprint features of the target speaker, fine-grained clustering via personal VAD [23] is executed to obtain precise video segments of the target speaker.

Chapter 3

RESULTS

In practice, our primary approach involved utilizing the pipeline, which leverages the Bilibili API method 2.1.2, in conjunction with the acceleration technique described in Section 2.1.3, and the clustering method outlined in the Section 2.2.2, to efficiently download and process videos. Over an extended period of operation, this automated pipeline successfully downloaded 162,076 speaker videos, subsequently processing them to compile a substantial database. We conducted a preliminary analysis of this dataset using Python, yielding basic insights. Below are some key findings from our dataset analysis:

Dataset	*Voxblink-CN*
# of SPKs	41,634
# of videos	789,321
# of hours	11,385
# of utterances	3,641,941
Avg # of videos per SPK	18.96
Avg # of utterances per SPK	87.48
Avg # of duration per utterance(s)	11.25
Max duration of utterance(s)	589.78
Min duration of utterance(s)	1.5
Med duration of utterance(s)	5.95

Table 3.1: Statistics for the VoxBlink-CN dataset.

The initial four rows of the table provide an overarching analysis of the dataset, presenting the count of distinct speakers, unique videos, total video duration (in hours), and total utterances. Subsequently, the third and fourth rows offer insights into the average number of videos per speaker post-processing and the average number of utterances per speaker, respectively. The last four rows analyze the average length, max-

imum length, minimum length, and median length from the perspective of utterance.

We used the python matplotlib [21] package to draw a histogram 3.1 showing the distribution of the number of different videos where the utterances come from for each speaker in the dataset.

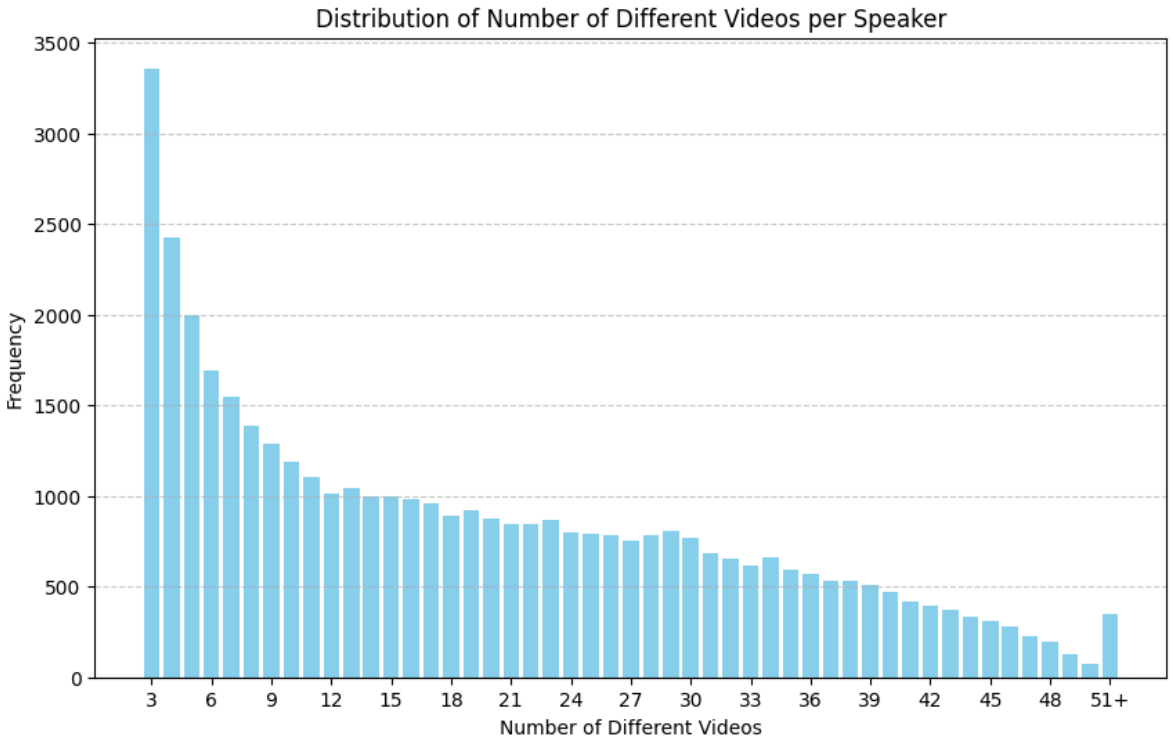

Figure 3.1: Distribution of the number of different videos where the utterances come from for each speaker.

To ensure diversity among speaker videos, we implemented an exclusion criterion: speakers with fewer than two videos were excluded. In the final version of the pipeline, we capped the maximum number of videos for a speaker at 50. However, for some of the previous versions, there were no such limitations. This causes a few individual speakers to have more than 50 videos. Due to their rarity, we categorized all instances exceeding 50 videos as 51+ videos.

Additionally, to visually analyze utterance length within the dataset, we generated another histogram 3.2 representing utterance length distribution.

Given that excessively short video clips typically offer limited utility for tasks like speaker recognition, we opted to exclude video clips smaller than 1 second during video processing. Consequently, when analyzing and visualizing the dataset, we tabulated the number of utterances across various length intervals, commencing from 1 second and incrementing by one-second units. However, due to the relatively scattered distribution of utterances over 30 seconds, we grouped utterances exceeding 30 seconds into

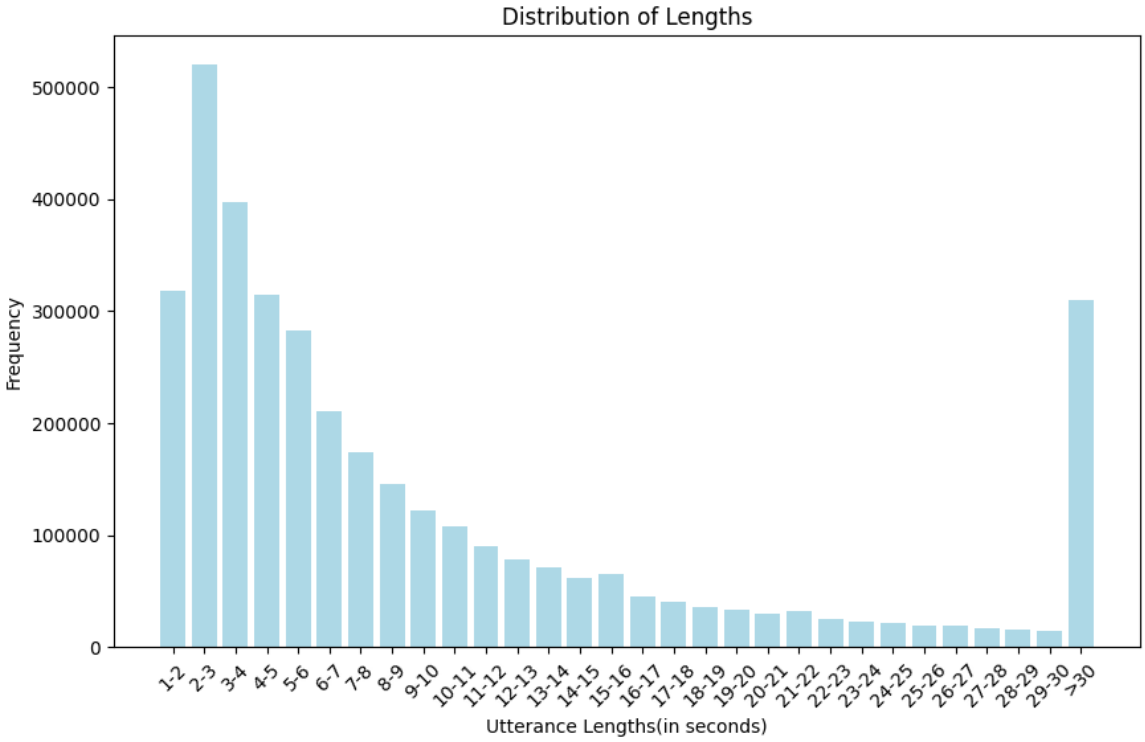

Figure 3.2: Distribution of the utterance's lengths.

a single category for ease of classification and interpretation.

Chapter 4

DISCUSSION

In this section, we will compare the pipeline we developed and delve into the journey of its creation, discussing the challenges encountered in pipeline design. Subsequently, we will analyze the VoxBlink-CN dataset we created, providing insights into our future work.

4.1 Pipeline analysis

To construct the dataset, we developed an automated pipeline capable of efficiently downloading videos from Bilibili and processing them to extract the desired video clips. Each step of the pipeline was implemented using a combination of methods tailored to its specific requirements.

Initially, we employed the ChromeDriver method 2.1.1 for video downloading due to its simplicity and ease of implementation. However, this approach presented challenges, particularly for large-scale data mining. ChromeDriver normally necessitated an open Chrome window throughout the process, which was inconvenient. Moreover, the code execution was sluggish as it depended on complete page loading, thus making it susceptible to internet speed fluctuations and potential errors.

Moreover, the ChromeDriver version of the pipeline posed limitations, primarily its reliance on videos with real face avatars. This restriction excluded many potentially valid videos from Bilibili. While removing the face recognition function in the pipeline could solve this issue, the speed constraints prompted us to explore alternatives. Subsequently, we transitioned to the Bilibili API version 2.1.2, which significantly enhanced processing speed. However, this approach introduced new challenges, such as the risk of Bilibili account suspension due to heightened activity. Despite implementing

breaks and proxy methods 2.1.3 to circumvent anti-crawling measures, we eventually adopted the cookie method 2.1.3 to overcome these hurdles successfully.

During the video processing stage, our initial attempt involved implementing the method utilized in VoxBlink 2.2.1. However, as previously mentioned, many videos on Bilibili lack the speaker's face on screen, yet contain high-quality audio of the speaker. These videos risk being excluded by the VoxBlink pipeline, which relies on avatars to determine the target speaker. Consequently, we developed the Clustering method 2.2.2 as an alternative approach.

The Clustering method entails creating a large-scale audio dataset and subsequently identifying videos with real face information. This process results in the creation of the VoxBlink-CN-AV dataset, which serves as a subset of the VoxBlink-CN dataset. By considering visual information during video processing, the VoxBlink-CN-AV dataset offers enhanced precision compared to its counterpart.

Another significant limitation we encountered was the storage and processing constraints of our devices when constructing a speaker verification system. Processing videos to build a comprehensive database proved to be time-consuming, with a single video taking several minutes to process. Scaling this to handle the entire dataset would be impractical and financially unviable. To address this, we restricted processing to the initial two minutes of each video, striking a balance between project objectives and hardware limitations. Additionally, storage space posed a formidable challenge, with each keyword-associated video consuming approximately 30 gigabytes. To manage this, we reduced the maximum video length from 10 minutes to 2 minutes, ensuring compatibility with our hardware constraints.

Although ChromeDriver, proxy methods, and the VoxBlink pipeline were inadequate for our needs, we integrated the Bilibili API method, the cookie method, and the clustering method into our final pipeline, successfully collecting the majority of data for the VoxBlink-CN dataset. While these methods proved effective for our task, other approaches may be beneficial for different tasks and warrant exploration in future endeavors.

4.2 VoxBlink-CN dataset analysis

The analysis of the VoxBlink-CN dataset is primarily based on the tables and figures presented in the Results section.

Table 3.1 illustrates the substantial scale of VoxCeleb-CN, showcasing 41,634 distinct speakers, 36,461,941 utterances, and a total of 11,385 hours of processed video content. This represents a significant enhancement in size compared to other Chinese

datasets, such as the well-known CN-Celeb dataset, which features 3,000 speakers, 600,000+ utterances, and 1,200+ hours of video [12]. The remarkable expansion in size is attributed to our approach of sourcing video bloggers from various platforms, particularly Bilibili. Unlike datasets focused on specific subsets, like Chinese celebrities, our strategy broadens the search scope, contributing to the dataset's substantial growth.

Furthermore, VoxCeleb-CN exhibits a broad spectrum of topics, rendering the database more diverse. We employ over 200 keywords for Bilibili searches. As a result, search results encompass bloggers from diverse backgrounds, enriching the dataset's thematic diversity.

Figure 3.1 underscores that video clips attributed to individual speakers stem from multiple distinct videos. This ensures diversification, encompassing temporal and spatial variations, thereby enhancing the variability of videos depicting the same individual. Such diversity contributes to bolstering the robustness of models trained on this dataset.

Regarding utterance duration, the majority of clips cluster within 10 seconds, aligning well with the requirements of speech recognition tasks. However, there remain numerous clips exceeding this threshold, some even surpassing 30 seconds. In future endeavors, we may consider segmenting these longer clips to ensure compliance with the 30-second limit.

4.3 Future work

In the future, in addition to the methods mentioned above, such as incorporating visual information to create the VoxBlink-CN-VA dataset and segmenting excessively long video clips to less than 30 seconds, there are several other avenues for further enhancing the construction of the VoxBlink-CN dataset.

Firstly, we can expand the size of the dataset even further. Currently, the videos we download originate from over two hundred keyword searches. By generating additional keywords and conducting more extensive searches on Bilibili, we can acquire more speakers and videos to augment the dataset.

Secondly, we can explore methods to improve the efficiency of the pipeline operation for downloading and processing videos. It's notable that out of the 162,076 initially downloaded speakers, only 41,634 speakers meet the criteria for inclusion in the dataset after processing. Approximately three-fourths of the speakers are discarded during processing due to not meeting our inclusion criteria. To address this, we can implement further screening of videos or speakers prior to downloading to enhance

video availability. Additionally, optimizing the video processing procedure can enable more videos to pass through without compromising the quality of the dataset.

Moreover, while the dataset size continues to expand, we have yet to objectively evaluate the data quality. Currently, assessment relies on subjective judgment alone. To address this, we can employ the same modeling framework to train both our VoxBlink-CN dataset and other prominent Chinese datasets, such as CN-Celeb, separately. By comparing their modeling performance, we can objectively assess the data quality of our dataset.

Furthermore, with the advancement and widespread adoption of speech synthesis technology, many video creators are employing it to create videos. They may provide video scripts for AI to read aloud, manipulate their own voices, or mimic the voices of celebrities. This can lead to less authentic-sounding voices. Additionally, commercial AI often offers a limited selection of timbres, potentially causing different speakers' speech segments to share the same timbre, thereby impacting model performance. To mitigate this issue, we can utilize anti-spoofing models to screen speech data in the database and remove non-authentic human voice segments.

Chapter 5

CONCLUSIONS

Speaker verification technology is gaining increasing significance across various sectors, including intelligent assistants, banking, healthcare, and e-commerce. Despite their importance, these systems encounter challenges stemming from data limitations and hardware constraints. Particularly in the realm of Chinese speech datasets, there exists a considerable gap in both scale and accuracy when compared to English datasets.

To tackle these challenges head-on, we devised an automated multimodal data mining pipeline to develop VoxBlink-CN, a substantial Chinese speech dataset primarily tailored for speaker verification tasks.

The journey of constructing a dataset like VoxBlink-CN has been a rigorous and ongoing endeavor. However, the creation of such a large-scale, scalable dataset characterized by precision, diversity, and scale offers significant promise for advancing the field of Chinese speaker verification. Furthermore, the construction process of the entire pipeline and the diverse methods employed hold valuable reference value for the future development of voice datasets.

REFERENCES

[1] M. Cheng et al. “The dku-msxf diarization system for the voxceleb speaker recognition challenge 2023”. In: *arXiv preprint arXiv:2308.07595* (2023).

[2] M. Cheng et al. “The whu-alibaba audio-visual speaker diarization system for the misp 2022 challenge”. In: *Proc. ICASSP*. 2023, pp. 1–2.

[3] *ChromeDriver - WebDriver for Chrome*. URL: https://chromedriver.chromium.org (visited on 03/21/2024).

[4] J. Deng et al. “Arcface: Additive angular margin loss for deep face recognition”. In: *Proc. CVPR*. 2019, pp. 4685–4694.

[5] J. Deng et al. “Retinaface: Single-shot multi-level face localisation in the wild”. In: *Proc. CVPR*. 2020, pp. 5202–5211.

[6] *FFmpeg*. https://ffmpeg.org/. Accessed on: March 21, 2024.

[7] Python Software Foundation. *asyncio - Asynchronous I/O, event loop, coroutines and tasks —Python 3.10.0 documentation*. [Online; accessed January 2022]. 2022. URL: https://docs.python.org/3/library/asyncio.html.

[8] Python Software Foundation. *multiprocessing - Process-based parallelism —Python 3.10.0 documentation*. [Online; accessed January 2022]. 2022. URL: https://docs.python.org/3/library/multiprocessing.html.

[9] *HideMy.io - Proxy List*. https://hidemy.io/cn/proxy-list/. Accessed: March 22, 2024.

[10] *IHuan IP - Free Proxy List*. https://ip.ihuan.me/. Accessed: March 22, 2024.

[11] *KuaiDaili - Proxy Service Provider*. https://www.kuaidaili.com/. Accessed: March 22, 2024.

[12] Lantian Li et al. “Cn-celeb: multi-genre speaker recognition”. In: *Speech Communication* 137 (2022), pp. 77–91.

[13] Yuke Lin et al. “VoxBlink: A Large Scale Speaker Verification Dataset on Camera”. In: *arXiv preprint arXiv:2308.07056* (2024). arXiv: 2308.07056 [eess.AS].

[14] Arsha Nagrani et al. “Voxceleb: Large-scale speaker verification in the wild”. In: *Computer Speech & Language* 60 (2020), p. 101027.

[15] Nemo2011. *Bilibili API*. https://github.com/Nemo2011/bilibili-api. 2024.

[16] OpenAI. *ChatGPT*. 2024. URL: https://openai.com/chatgpt (visited on 03/21/2024).
[17] OpenCV Contributors. *OpenCV Library*. URL: https://opencv.org. OpenCV. 2022.
[18] *Python Documentation: time module*. https://docs.python.org/3/library/time.html. Accessed: March 22, 2024.
[19] scikit-learn contributors. *scikit-learn: Machine Learning in Python*. Version 1.4.1. 2024.
[20] *Sugon Advanced Computing Co., Ltd.* https://ac.sugon.com/home/index.html. Accessed on: March 21, 2024.
[21] Matplotlib Development Team. *Matplotlib: Visualization with Python*. Matplotlib. 2022.
[22] Hongji Wang et al. *Wespeaker: A Research and Production oriented Speaker Embedding Learning Toolkit*. Submitted on 31 Oct 2022 (v1), last revised 1 Nov 2022 (this version, v2). 2022. arXiv: 2210.17016 [cs.SD].
[23] Bang Zeng et al. "Efficient Personal Voice Activity Detection with Wake Word Reference Speech". In: (2024).