Evaluation of Resemblyzer Ability to Authenticate the Person by Short Audio Records*

Mykhailo Trusov*, Oleksii Turuta and Dmytro Uzlov

V.N. Karazin Kharkiv National University, Kharkiv 61022, Ukraine

Abstract

Voice-based control represents an efficient way to interact with IoT devices and autonomous systems, but its use is often limited by privacy, security, and hardware constraints. The present study explores the open-source Python library Resemblyzer for on-device speaker authentication using short voice commands, focusing on offline operation in resource-constrained environments. Experiments with datasets of different audio quality, duration, and size, covering both same-speaker and cross-speaker cases, used cosine similarity to detect false positives, false negatives, and threshold results. We found that reliable authentication is possible with recordings as short as 2.63 seconds and at least 495 KB in size, while shorter clips (1-1.5 seconds) are less dependable, especially against high-quality references. These results show that Resemblyzer can run effectively without cloud access, offering practical guidance on minimal audio requirements for secure, real-time voice verification in IoT and robotic systems.

Keywords

voice-based control, speaker authentication, Resemblyzer, short voice commands, Internet of Things (IoT), resource-constrained devices, voice verification, voiceprint, cosine similarity

1. Introduction

A wide range of portable and connected devices, such as mobile phones, laptops, e-readers, smartwatches, wireless earbuds, robotic vacuum cleaners, smart air conditioners etc., has become ubiquitous in modern life. This list could be extended considerably, however, it is an indisputable fact that the Internet of Things (IoT) has already become an integral component of contemporary society, and that robots, in the broad sense of the term, now serve as indispensable assistants in our daily activities [1-3]. In this context an important arises: how can users effectively interact with such assistants? One of the approaches involves the use of control interfaces, which may range from keyboards and touch panels to specialized devices capable of transmitting commands via radio frequency or infrared channels as well as through voice-based commands [4,5]. Among these, interaction via voice commands represents the most natural and straightforward way for issuing instructions, requesting actions, or retrieving information. In addition, this modality does not require specific skills or additional efforts associated with the utilization of specific input devices, and it typically offers the fastest way of communication [6]. However, voice-based interaction is accompanied by a range of technical and practical challenges [8,9]. For instance, background noise can significantly decrease voice recognition accuracy, while variations in dialects, speech rate, pronunciation, and accents further complicate reliable interpretation. More critically, voice-controlled systems raise substantial privacy and security concerns. Beyond the inherent risk that speaking aloud may inadvertently disclose sensitive information (e.g., passwords, PIN codes, or message content), it becomes critically important in contexts involving more sensitive systems than a coffee machine. Whether issuing commands to a mobile device, an industrial robot, or even a combat drone, it is essential to ensure that the system can reliably authenticate the speaker and respond exclusively to authorized individuals.

* *ProfIT AI'25: 5th International Workshop of IT-professionals on Artificial Intelligence, October 15–17, 2025, Liverpool, UK*

1* Corresponding author.

mykhailo.trusov@karazin.ua (M. Trusov); oleksii.turuta@gmail.com (O. Turuta); dmytro.uzlov@karazin.ua (D. Uzlov)

0009-0001-4390-5307 (M. Trusov); 0000-0002-0970-8617 (O. Turuta); 0000-00Q3-3308-424X (D. Uzlov)

In the present work we evaluated the potential of publicly available Python library Resemblyzer for user authentication based on short voice commands [10]. Specifically, the main goal was to determine empirically the minimal audio conditions under which short voice commands can be used for robust user authentication with Resemblyzer.

2. Methodology

We focused on Resemblyzer library as the core speaker embedding tool due to its strong performance in voiceprint-related tasks, public availability and active maintaining by community, compatibility with Python's scientific ecosystem (e.g., NumPy, Pandas, scikit-learn) and ability to operate in both batch and near real-time contexts [10]. It leverages a pretrained model based on the GE2E (Generalized End-to-End) architecture, which was introduced by Google for speaker verification, enabling it to generate compact, 256-dimensional speaker embeddings that generalize well even for short utterances. Once the embedding is computed, only the small vector (1 KB) needs to be stored or compared – not the full audio. The pretrained model is stored as a PyTorch .pt file and can be located locally, it is reasonably small (~15–20MB) depending on the version. The size of the model is significantly lighter than transformer-based models like Whisper, ECAPA-TDNN, or wav2vec.

Comparison of popular solutions and their pretrained models by size can be found in Table 1.

Table 1
Size comparison for selected speaker embedding and speech processing models

Non-English or Math	Frequency
Resemblyzer (GE2E)	~15
ECAPA-TDNN (SpeechBrain)	50-100+
Wav2Vec2 (Facebook)	300+
Whisper (OpenAI) small	~70
SpeakerNet	~25-60

As can be seen from this table, Resemblyzer model is the smallest one. This advantage plays an important role taking into consideration that it should be used in resource-constrained devices. Moreover, authentication based on voiceprint analyses in Resemblyzer library works purely on-device with no need to send audio to cloud services. In other words, no network communication is required.

The implementation of the work involved a set of tests with short audio records designed to emulate voice commands for a combat drone. All audio records were saved in .wav format and produced with a standard computer microphone by 'Sound Recorder' application which is available at any Windows OS. We deliberately avoided high quality devices with a purpose to simulate the real-life conditions.

Two types of voices, male and female, were included into the experiments. Information about audio records used and identification of the corresponding input sets are summarized below (Tables 2-5).

Table 2
Dataset #1. Voice: male

File name	Text in audio	Length of audio (seconds)
Analyze.wav	Analyze	1,16
Attack.wav	Attack	1,34
Authentication.wav	Authentication	1,50
Autopilot.wav	Autopilot	1,42
Check.wav	Check	1,29
Defence.wav	Defence	1,48
Destroy.wav	Destroy	1,53
Find.wav	Find	1,46
Follow.wav	Follow	1,42
Freeze.wav	Freeze	1,40
Go.wav	Go	1,42
Home.wav	Home	1,34
Identity.wav	Identity	1,59
Listen.wav	Listen	1,20
Stop.wav	Stop	1,02
Wait.wav	Wait	1,18
Watch.wav	Watch	1,23

Table 3
Dataset #2. Voice: male

File name	Text in audio	Length of audio (seconds)
A1-m.waw	The Adventures of Sherlock Holmes" by Arthur Conan Doyle is a collection of detective stories written during the late 19th century. The book introduces the legendary detective Sherlock Holmes and his loyal companion, Dr. John Watson	15,78

Table 4
Dataset #3. Voice: female

File name	Text in audio	Length of audio (seconds)
Analyze.wav	Analyze	1,49
Attack.wav	Attack	1,24
Authentication.wav	Authentication	1,84
Autopilot.wav	Autopilot	1,84
Check.wav	Check	1,19
Defence.wav	Defence	1,52
Destroy.wav	Destroy	1,33
Find.wav	Find	1,37
Follow.wav	Follow	1,38
Freeze.wav	Freeze	0,55
Go.wav	Go	1,2
Home.wav	Home	1,42
Identity.wav	Identity	1,48
Listen.wav	Listen	1,47
Stop.wav	Stop	1,15
Wait.wav	Wait	1,19
Watch.wav	Watch	1,46

Table 5
Dataset #4. Voice: female

File name	Text in audio	Length of audio (seconds)
A1-f.waw	The Adventures of Sherlock Holmes" by Arthur Conan Doyle is a collection of detective stories written during the late 19th century. The book introduces the legendary detective Sherlock Holmes and his loyal companion, Dr. John Watson	12,99

Table 6
Dataset #5. Voice: male

File name	Text in audio	Length of audio (seconds)
Gltm-Analyze.wav	Ginger, listen to me, Analyze	3,16
Gltm-Attack.wav	Ginger, listen to me, Attack	3,07
Gltm-Authentication.wav	Ginger, listen to me, Authentication	3,64
Gltm-Autopilot.wav	Ginger, listen to me, Autopilot	3,02
Gltm-Check.wav	Ginger, listen to me, Check	3,21
Gltm-Defence.wav	Ginger, listen to me, Defence	3,03
Gltm-Destroy.wav	Ginger, listen to me, Destroy	3,16
Gltm-Find.wav	Ginger, listen to me, Find	2,68
Gltm-Follow.wav	Ginger, listen to me, Follow	3,89
Gltm-Freeze.wav	Ginger, listen to me, Freeze	2,86
Gltm-Go.wav	Ginger, listen to me, Go	2,63
Gltm-Home.wav	Ginger, listen to me, Home	2,73
Gltm-Identity.wav	Ginger, listen to me, Identity	2,92
Gltm-Listen.wav	Ginger, listen to me, Listen	2,95
Gltm-Stop.wav	Ginger, listen to me, Stop	2,78
Gltm-Wait.wav	Ginger, listen to me, Wait	2,68
Gltm-Watch.wav	Ginger, listen to me, Watch	2,86

For each audio recording, a voiceprint was generated using the Resemblyzer library, producing a 256-element array of floating values. Each array was saved in a separate .csv file, named identically to its corresponding original audio file.

The Python code used to perform this operation is provided below:

```
import pandas as pd
from resemblyzer import VoiceEncoder, preprocess_wav
import numpy as np
from pathlib import Path
import os

def calculate_n_save_voiceprint(audio_record: str, output_folder: str, csv_separator: str = ";", csv_decimal_symbol: str = ","):
    """Calculation of voiceprint for the passed audio file and saving it to a CSV file.

    Args:
        audio_record (str): Path to the audio record (.wav file).
```

```
        output_folder (str): Path to the output folder where the CSV file will be saved.

    Raises:
        ValueError: If the audio record path is invalid or not a .wav file.
    """
    audio_record_path = Path(audio_record)
    if not audio_record_path.exists() or audio_record_path.suffix.lower() != ".wav":
        raise ValueError(f"Invalid .wav file path: {audio_record_path}")

    output_folder_path = Path(output_folder)
    output_folder_path.mkdir(parents=True, exist_ok=True)

    # Load and process audio
    preprocessed_wav_data = preprocess_wav(audio_record_path)

    # Generate embedding
    encoder = VoiceEncoder()
    embedding = encoder.embed_utterance(preprocessed_wav_data)

    # Create output CSV path: same filename, different folder, .csv extension
    output_csv_path = output_folder_path / (audio_record_path.stem + ".csv")

    # Save embedding with comma as decimal separator
    df = pd.DataFrame([embedding])
        df.to_csv(output_csv_path, sep=csv_separator, decimal=csv_decimal_symbol, header=False, index=False,
float_format="%.8f")

    print(f"Voiceprint for {audio_record_path.name} was saved to: {output_csv_path}")

def calculate_n_save_voiceprints(input_folder: str, output_folder: str, csv_separator: str = ";", csv_decimal_symbol: str
= "," ):
    """Calculate and save voiceprints as separate csv file for all .wav files in the input folder.

    Args:
        input_folder (str): Path to the folder containing .wav files.
        output_folder (str): Path to the folder where voiceprints will be saved.
        csv_separator (str, optional): Separator for the output CSV files. Defaults to ";".
        csv_decimal_symbol (str, optional): Symbol for decimal point in the output CSV files. Defaults to ",".
    """
    for file_name in os.listdir(input_folder):
        if file_name.lower().endswith(".wav"):
            print(f"Processing: {file_name}")
            file_path = os.path.join(input_folder, file_name)
            try:
                calculate_n_save_voiceprint(file_path, output_folder, csv_separator, csv_decimal_symbol)
            except ValueError as e:
                print(e)
            except Exception as e:
                print(f"An error occurred while processing {file_name}: {e}")
```

An example of abridged output voiceprint data in single .csv file may be illustrated as:

```
0,13736048;0,00000000;0,16476172;0,00000000;0,00000000;0,00000000;0,00000000;0,00000000;0,06845610;0,21822383;0,04
552145;0,00026178;0,08779779;0,09342065;0,00000000;...
0,02555513;0,00000000;0,00000000;0,00000000;0,00000000;0,00000000;0,00000000;0,00602875;0,11488174;0,07095279;0,07308
850;0,00000000;0,12232912;0,14700639;0,00000000;0,07068025;0,12317624;0,00709428;0,00000000;0,00000000;0,00000000;0,03
307733;0,08165727;0,00000000;0,06436168;0,00000000
```

The authentication process and evaluation of similarity between two voiceprints can be quantified using cosine similarity, a widely used metric in speaker verification systems:

$$Similarity = \frac{A * B}{\|A\| * \|B\|}. \quad (1)$$

Cosine similarity yileds a value between −1.0 and +1.0, indicating how closely two voiceprint vectors align in the embedding space. Specifically, +1.0 value means almost identical voices, 0.0 indicates no similarity, and −1.0 value represents complete opposition, an outcome unlikely in the context of our work. Heuristic or empirical result equal or greater than 0.75 means successful authentication. In speaker verification systems using cosine similarity, typical same-speaker scores fall around 0.8-0.95, different-speaker scores often fall around 0.3-0.6, so 0.75 is commonly used as a default threshold in prototyping or academic examples.

Python code which compares two voiceprints and calculates the similarity can be found in the listing below:

```
def compare_2_voiceprints(voiceprint_csv_file1: str, voiceprint_csv_file2: str, csv_separator: str = ";", csv_decimal_symbol: str = ",") -> float:
    """
    Compare two voiceprints and return the similarity score.
    Args:
        voiceprint_csv_file1 (str): Path to the first voiceprint CSV file.
        voiceprint_csv_file2 (str): Path to the second voiceprint CSV file.
    """
    df1 = pd.read_csv(voiceprint_csv_file1, sep=csv_separator, decimal=csv_decimal_symbol, header=None)
    df2 = pd.read_csv(voiceprint_csv_file2, sep=csv_separator, decimal=csv_decimal_symbol, header=None)

    if df1.shape[1] != df2.shape[1]:
        raise ValueError("Voiceprints must have the same number of dimensions.")

    # Calculate cosine similarity
    embedding1 = df1.iloc[0].values
    embedding2 = df2.iloc[0].values

    similarity = np.dot(embedding1, embedding2) / (np.linalg.norm(embedding1) * np.linalg.norm(embedding2))

    return similarity
```

3. Results and Discussion

3.1. Experiment 1: calculating the similarity for the short audio records produced by the same person

In our first experiment we compare voiceprints received from dataset #1 between each other. The algorithm was the following:

1. Selection of the first .csv with voiceprint from the target folder (e.g. analyze.csv).
2. Comparison of this voiceprint with those voiceprints stored in all other .csv files in the same target folder
3. Record the calculated similarity scores in the results.csv file as a new row. The header of this file includes names of files taking part in comparison operation (all csv files found in the target folder) referred to here as destination files. The first column contains the names of source file, and each subsequent column stores the value of calculated similarity between the source and destination files.
4. Repetition of the process for the next .csv file with voiceprint.

The format and structure of result.csv file allows us to import easily these data into Excel and utilize more sophisticated tools for the analysis.

An example of output result.csv file received during this experiment can be found below:

```
;analyze;attack;authentication;autopilot;check;defence;destroy;find;follow;freeze;go;home;identity;listen;stop;wait;wat
ch
analyze;1,0;0,74;0,72;0,65;0,67;0,79;0,69;0,8;0,81;0,78;0,79;0,85;0,79;0,86;0,72;0,83;0,8
attack;0,74;1,0;0,67;0,75;0,71;0,69;0,63;0,75;0,73;0,71;0,71;0,64;0,73;0,75;0,84;0,81;0,8
authentication;0,72;0,67;1,0;0,66;0,58;0,8;0,7;0,77;0,64;0,65;0,61;0,62;0,81;0,77;0,58;0,64;0,62
autopilot;0,65;0,75;0,66;1,0;0,62;0,63;0,67;0,71;0,75;0,67;0,69;0,64;0,71;0,6;0,7;0,7;0,7
check;0,67;0,71;0,58;0,62;1,0;0,57;0,65;0,72;0,78;0,76;0,81;0,67;0,62;0,64;0,74;0,81;0,75
defence;0,79;0,69;0,8;0,63;0,57;1,0;0,72;0,82;0,68;0,68;0,66;0,69;0,79;0,82;0,64;0,67;0,63
destroy;0,69;0,63;0,7;0,67;0,65;0,72;1,0;0,75;0,74;0,75;0,77;0,76;0,8;0,75;0,65;0,73;0,7
find;0,8;0,75;0,77;0,71;0,72;0,82;0,75;1,0;0,76;0,79;0,79;0,74;0,84;0,83;0,7;0,8;0,72
follow;0,81;0,73;0,64;0,75;0,78;0,68;0,74;0,76;1,0;0,82;0,89;0,82;0,71;0,73;0,77;0,87;0,83
freeze;0,78;0,71;0,65;0,67;0,76;0,68;0,75;0,79;0,82;1,0;0,86;0,84;0,81;0,81;0,78;0,86;0,84
go;0,79;0,71;0,61;0,69;0,81;0,66;0,77;0,79;0,89;0,86;1,0;0,87;0,75;0,72;0,82;0,88;0,83
home;0,85;0,64;0,62;0,64;0,67;0,69;0,76;0,74;0,82;0,84;0,87;1,0;0,73;0,8;0,76;0,85;0,83
identity;0,79;0,73;0,81;0,71;0,62;0,79;0,8;0,84;0,71;0,81;0,75;0,73;1,0;0,84;0,7;0,76;0,73
listen;0,86;0,75;0,77;0,6;0,64;0,82;0,75;0,83;0,73;0,81;0,72;0,8;0,84;1,0;0,68;0,79;0,79
stop;0,72;0,84;0,58;0,7;0,74;0,64;0,65;0,7;0,77;0,78;0,82;0,76;0,7;0,68;1,0;0,84;0,85
wait;0,83;0,81;0,64;0,7;0,81;0,67;0,73;0,8;0,87;0,86;0,88;0,85;0,76;0,79;0,84;1,0;0,93
watch;0,8;0,8;0,62;0,7;0,75;0,63;0,7;0,72;0,83;0,84;0,83;0,83;0,73;0,79;0,85;0,93;1,0
```

Python code which does comparison and produces result.csv can be found below:

```
def calculate_voiceprints_similarity(input_folder: str, output_csv: str, csv_separator: str = ";", csv_decimal_symbol: str
= ","):
    """Calculate the similarity matrix for voiceprints.
    Args:
        input_folder (str): Path to the folder containing voiceprint CSV files.
        output_csv (str): Path to the output CSV file for the similarity matrix.
        csv_separator (str, optional): Separator for the output CSV file. Defaults to ";".
        csv_decimal_symbol (str, optional): Decimal symbol for the output CSV file. Defaults to
    Raises:
        ValueError: If the input folder is invalid or contains no CSV files.
    """
    folder = Path(input_folder)
    if not folder.is_dir():
        raise ValueError(f"{input_folder} is not a valid folder.")

    csv_files = sorted(folder.glob("*.csv"))
    if not csv_files:
        print("No CSV files found in the folder.")
        return

    file_names = [f.stem for f in csv_files]
    file_paths = {f.stem: str(f) for f in csv_files}

    # Initialize with NaN to ensure float dtype
    df_result = pd.DataFrame(np.nan, index=file_names, columns=file_names)

    for a in file_names:
        for b in file_names:
            if a == b:
                df_result.loc[a, b] = 1.0
            else:
                similarity = compare_2_voiceprints(
                    file_paths[a],
                    file_paths[b],
                    csv_separator=csv_separator,
                    csv_decimal_symbol=csv_decimal_symbol,
                )
                df_result.loc[a, b] = round(similarity, 2)

    df_result.index.name = ""
    df_result.to_csv(output_csv, sep=csv_separator, decimal=csv_decimal_symbol)

    print(f"Similarity matrix saved to {output_csv}")
```

The results of comparison for dataset # 1 are summarized in Table 7.

Table 7
Comparison for dataset # 1

	analyse	attack	auth-tion	autopilot	check	defence	destroy	find	follow	freeze	go	home	identity	listen	stop	wait	watch
analyse	1	0,74	0,72	0,65	0,67	0,79	0,69	0,8	0,81	0,78	0,79	0,85	0,79	0,86	0,72	0,83	0,8
attack	0,74	1	0,67	0,75	0,71	0,69	0,63	0,75	0,73	0,71	0,71	0,64	0,73	0,75	0,84	0,81	0,8
auth-tion	0,72	0,67	1	0,66	0,58	0,8	0,7	0,77	0,64	0,65	0,61	0,62	0,81	0,77	0,58	0,64	0,62
autopilot	0,65	0,75	0,66	1	0,62	0,63	0,67	0,71	0,75	0,67	0,69	0,64	0,71	0,6	0,7	0,7	0,7
check	0,67	0,71	0,58	0,62	1	0,57	0,65	0,72	0,78	0,76	0,81	0,67	0,62	0,64	0,74	0,81	0,75
defence	0,79	0,69	0,8	0,63	0,57	1	0,72	0,82	0,68	0,68	0,66	0,69	0,79	0,82	0,64	0,67	0,63
destroy	0,69	0,63	0,7	0,67	0,65	0,72	1	0,75	0,74	0,75	0,77	0,76	0,8	0,75	0,65	0,73	0,7
find	0,8	0,75	0,77	0,71	0,72	0,82	0,75	1	0,76	0,79	0,79	0,74	0,84	0,83	0,7	0,8	0,72
follow	0,81	0,73	0,64	0,75	0,78	0,68	0,74	0,76	1	0,82	0,89	0,82	0,71	0,73	0,77	0,87	0,83
freeze	0,78	0,71	0,65	0,67	0,76	0,68	0,75	0,79	0,82	1	0,86	0,84	0,81	0,81	0,78	0,86	0,84
go	0,79	0,71	0,61	0,69	0,81	0,66	0,77	0,79	0,89	0,86	1	0,87	0,75	0,72	0,82	0,88	0,83
home	0,85	0,64	0,62	0,64	0,67	0,69	0,76	0,74	0,82	0,84	0,87	1	0,73	0,8	0,76	0,85	0,83
identity	0,79	0,73	0,81	0,71	0,62	0,79	0,8	0,84	0,71	0,81	0,75	0,73	1	0,84	0,7	0,76	0,73
listen	0,86	0,75	0,77	0,6	0,64	0,82	0,75	0,83	0,73	0,81	0,72	0,8	0,84	1	0,68	0,79	0,79
stop	0,72	0,84	0,58	0,7	0,74	0,64	0,65	0,7	0,77	0,78	0,82	0,76	0,7	0,68	1	0,84	0,85
wait	0,83	0,81	0,64	0,7	0,81	0,67	0,73	0,8	0,87	0,86	0,88	0,85	0,76	0,79	0,84	1	0,93
watch	0,8	0,8	0,62	0,7	0,75	0,63	0,7	0,72	0,83	0,84	0,83	0,83	0,73	0,79	0,85	0,93	1

Given that all audio recordings were produced by the same speaker, we expect that all similarity scores would exceed 0,8. However, the results revealed numerous cases with values below this threshold, and in some instances, even below 0,6. In the context of our, scores below 0.6 can be considered false negatives, the range between 0,6 and 0,8 represents an uncertain or threshold zone, and values above 0,8 indicate a successful match. To better understand the distribution of results, we calculated the proportion of scores falling within each range. The following Python function performs this calculation:

```
def analyze_similarity_matrix(similarities_csv_file: str, failedRange: float, succeedRange: float, csv_separator: str = ";", csv_decimal_symbol: str = ","):
    """
    Reads a similarity matrix from CSV and calculates percentage of values
    falling into failed, threshold, and succeed categories.

    Args:
        similarities_csv_file (str): Path to the similarity matrix CSV.
        failedRange (float): Upper bound for failed values (exclusive).
        succeedRange (float): Lower bound for succeed values (inclusive).

    Returns:
        dict: Percentages of failed, threshold, and succeed values.
    """
    df = pd.read_csv(similarities_csv_file, sep=csv_separator, decimal=csv_decimal_symbol, index_col=0)

    # Extract all similarity values except diagonal (self-comparisons)
    values = [
        float(value)
        for i, row in df.iterrows()
        for j, value in row.items()
        if i != j
    ]

    # Classification counters
```

```
failed_count = sum(1 for v in values if v < failedRange)
succeed_count = sum(1 for v in values if v >= succeedRange)
threshold_count = len(values) - failed_count - succeed_count

total_count = len(values)
if total_count == 0:
    return {"total count": 0, "failed": 0.0, "threshold": 0.0, "succeed": 0.0}

# Percentages
return {
    "total count": total_count,
    "failed": round(failed_count / total_count * 100, 2),
    "threshold": round(threshold_count / total_count * 100, 2),
    "succeed": round(succeed_count / total_count * 100, 2)
}
```

Resulting score is as follows (Table 8):

Table 8
Resulting scores

Range	Percentage (%)	Description
[0 : 0,6]	2,94	Failed scenario
(0,6 : 0,8)	66,91	Threshold scenario
[0,8 : 1]	30,15	Succeed scenario
Total number: 272		

Let's do the same experiment for dataset #3 – analogous short audio records recorded by female. The results of comparison for dataset # 3 are given in Table 9.

Table 9
Comparison for dataset #3

	analyse	attack	auth-tion	autopilot	check	defence	destroy	find	follow	freeze	go	home	identity	listen	stop	wait	watch
analyse	1	0,8	0,78	0,8	0,76	0,79	0,79	0,87	0,72	0,72	0,67	0,71	0,77	0,78	0,76	0,73	0,75
attack	0,8	1	0,86	0,92	0,86	0,88	0,88	0,86	0,8	0,8	0,73	0,76	0,86	0,82	0,84	0,81	0,8
auth-tion	0,78	0,86	1	0,9	0,83	0,88	0,84	0,84	0,76	0,78	0,7	0,73	0,91	0,82	0,81	0,8	0,77
autopilot	0,8	0,92	0,9	1	0,8	0,87	0,84	0,85	0,79	0,8	0,71	0,73	0,86	0,81	0,82	0,81	0,79
check	0,76	0,86	0,83	0,8	1	0,86	0,87	0,8	0,87	0,86	0,8	0,86	0,84	0,86	0,88	0,89	0,87
defence	0,79	0,88	0,88	0,87	0,86	1	0,92	0,89	0,75	0,84	0,76	0,75	0,92	0,85	0,82	0,83	0,81
destroy	0,79	0,88	0,84	0,84	0,87	0,92	1	0,85	0,74	0,75	0,79	0,76	0,87	0,79	0,82	0,8	0,8
find	0,87	0,86	0,84	0,85	0,8	0,89	0,85	1	0,77	0,79	0,76	0,77	0,85	0,82	0,8	0,81	0,82
follow	0,72	0,8	0,76	0,79	0,87	0,75	0,74	0,77	1	0,83	0,85	0,9	0,72	0,83	0,84	0,85	0,84
freeze	0,72	0,8	0,78	0,8	0,86	0,84	0,75	0,79	0,83	1	0,74	0,8	0,82	0,87	0,81	0,89	0,87
go	0,67	0,73	0,7	0,71	0,8	0,76	0,79	0,76	0,85	0,74	1	0,88	0,72	0,76	0,8	0,79	0,83
home	0,71	0,76	0,73	0,73	0,86	0,75	0,76	0,77	0,9	0,8	0,88	1	0,73	0,8	0,88	0,85	0,86
identity	0,77	0,86	0,91	0,86	0,84	0,92	0,87	0,85	0,72	0,82	0,72	0,73	1	0,85	0,79	0,82	0,79
listen	0,78	0,82	0,82	0,81	0,86	0,85	0,79	0,82	0,83	0,87	0,76	0,8	0,85	1	0,84	0,9	0,83
stop	0,76	0,84	0,81	0,82	0,88	0,82	0,82	0,8	0,84	0,81	0,8	0,88	0,79	0,84	1	0,84	0,9
wait	0,73	0,81	0,8	0,81	0,89	0,83	0,8	0,81	0,85	0,89	0,79	0,85	0,82	0,9	0,84	1	0,87
watch	0,75	0,8	0,77	0,79	0,87	0,81	0,8	0,82	0,84	0,87	0,83	0,86	0,79	0,83	0,9	0,87	1

The resulting score for this case is presented in Table 10:

Table 10
Resulting scores

Range	Percentage (%)	Description
[0 : 0,6]	0,0	Failed scenario
(0,6 : 0,8)	32,35	Threshold scenario
[0,8 : 1]	67,65	Succeed scenario
Total number: 272		

The results for dataset #3 were found to be notably better than those for dataset #1 – zero failed cases and a threshold range less than half the size of the successful matches. To understand the factors contributing to this improvement, at the next step of the study we compared audio recordings from the two datasets and tried to identify the characteristics that lead to higher similarity scores. The first step involved the detection of the cases with the largest differences between the two resulting similarity matrices. We take the similarity matrix from dataset #3, since it demonstrates superior performance, and subtract the corresponding matrix from dataset #1. In this context, the resulting delta serves as a quality indicator for the comparison (Table 11).

Table 11
Difference of similarity scores between dataset #3 and dataset #1

	analyse	attack	auth-tion	autopilot	check	defence	destroy	find	follow	freeze	go	home	identity	listen	stop	wait	watch
analyse	0	0,06	0,06	0,15	0,09	0	0,1	0,07	-0,09	-0,06	-0,12	-0,14	-0,02	-0,08	0,04	-0,1	-0,05
attack	0,06	0	0,19	0,17	0,15	0,19	0,25	0,11	0,07	0,09	0,02	0,12	0,13	0,07	0	0	0
auth-tion	0,06	0,19	0	0,24	0,25	0,08	0,14	0,07	0,12	0,13	0,09	0,11	0,1	0,05	0,23	0,16	0,15
autopilot	0,15	0,17	0,24	0	0,18	0,24	0,17	0,14	0,04	0,13	0,02	0,09	0,15	0,21	0,12	0,11	0,09
check	0,09	0,15	0,25	0,18	0	0,29	0,22	0,08	0,09	0,1	-0,01	0,19	0,22	0,22	0,14	0,08	0,12
defence	0	0,19	0,08	0,24	0,29	0	0,2	0,07	0,07	0,16	0,1	0,06	0,13	0,03	0,18	0,16	0,18
destroy	0,1	0,25	0,14	0,17	0,22	0,2	0	0,1	0	0	0,02	0	0,07	0,04	0,17	0,07	0,1
find	0,07	0,11	0,07	0,14	0,08	0,07	0,1	0	0,01	0	-0,03	0,03	0,01	-0,01	0,1	0,01	0,1
follow	-0,09	0,07	0,12	0,04	0,09	0,07	0	0,01	0	0,01	-0,04	0,08	0,01	0,1	0,07	-0,02	0,01
freeze	-0,06	0,09	0,13	0,13	0,1	0,16	0	0	0,01	0	-0,12	-0,04	0,01	0,06	0,03	0,03	0,03
go	-0,12	0,02	0,09	0,02	-0,01	0,1	0,02	-0,03	-0,04	-0,12	0	0,01	-0,03	0,04	-0,02	-0,09	0
home	-0,14	0,12	0,11	0,09	0,19	0,06	0	0,03	0,08	-0,04	0,01	0	0	0	0,12	0	0,03
identity	-0,02	0,13	0,1	0,15	0,22	0,13	0,07	0,01	0,01	0,01	-0,03	0	0	0,01	0,09	0,06	0,06
listen	-0,08	0,07	0,05	0,21	0,22	0,03	0,04	-0,01	0,1	0,06	0,04	0	0,01	0	0,16	0,11	0,04
stop	0,04	0	0,23	0,12	0,14	0,18	0,17	0,1	0,07	0,03	-0,02	0,12	0,09	0,16	0	0	0,05
wait	-0,1	0	0,16	0,11	0,08	0,16	0,07	0,01	-0,02	0,03	-0,09	0	0,06	0,11	0	0	-0,06
watch	-0,05	0	0,15	0,09	0,12	0,18	0,1	0,1	0,01	0,03	0	0,03	0,06	0,04	0,05	-0,06	0

By summing the values in each column, we obtain a cumulative weight for the corresponding file, allowing us to identify the cases with the largest overall differences. These results are presented in Fig. 1.

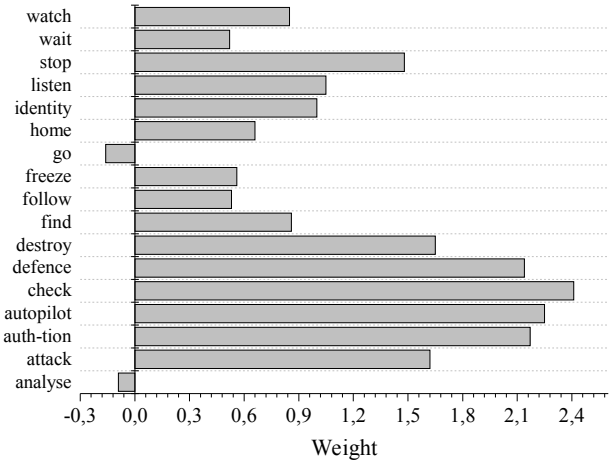

Figure 1: Cumulative weights of similarity differences between dataset #3 and dataset #1

The physical characteristics of input files are given in Table 12.

Table 12
Physical characteristics of the input files from dataset #3 and dataset #1

	Dataset #1. File size (Bytes)	Dataset #3. File size (Bytes)	Delta: Dataset #3 – Dataset #1 File size (Bytes)	Dataset #1. Length of audio track (seconds)	Dataset #3. Length of audio track (seconds)	Delta: Dataset #3 – Dataset #1 Length of audio track (seconds)
analyse	222802	286162	63360	1,16	1,49	0,33
attack	257362	238162	-19200	1,34	1,24	-0,1
auth-tion	288082	353362	65280	1,50	1,84	0,34
autopilot	272722	353362	80640	1,42	1,84	0,42
check	247762	228562	-19200	1,29	1,19	-0,1
defence	284242	291922	7680	1,48	1,52	0,04
destroy	293842	255442	-38400	1,53	1,33	-0,2
find	280402	263122	-17280	1,46	1,37	-0,09
follow	272722	265042	-7680	1,42	1,38	-0,04
freeze	268882	247762	-21120	1,40	0,55	-0,85
go	272722	230482	-42240	1,42	1,2	-0,22
home	257362	272722	15360	1,34	1,42	0,08
identity	305362	284242	-21120	1,59	1,48	-0,11
listen	230482	282322	51840	1,20	1,47	0,27
stop	195922	220882	24960	1,02	1,15	0,13
wait	226642	228562	1920	1,18	1,19	0,01
watch	236242	280402	44160	1,23	1,46	0,23

Based on the above data, no unconditional correlation can be established between file size or audio recording length and the success of authentication. For example, in the case of the “analyse” recording, dataset #3 contains a longer audio clip with a larger file size, yet its similarity score is lower than that of dataset #1. To better understand these correlations, we proceed by calculating the average delta values for file size and audio length across the entire dataset, allowing us to assess the overall differences. It appeared that dataset #3 has bigger average file size (~10 Kb greater) and almost the same length of audio record, but do not have false positive cases. These findings suggest that for short audio records with the length of 1-1,5 seconds, files with acontent size exceeding 270Kb are likely to produce voiceprint suitable for reliable person authentication with a probability of 80%.

3.2. Experiment 2: calculating the similarity for the short audio records produced by different person

In a previous scenario we used voiceprint received from short audio records produced by the same person. Now we try to compare the audio records from different speakers in order to determine the rate of false positive cases under these conditions.

We used the same datasets #1 and #3, but now we compare each file from one dataset with each file from another dataset. The algorithm included the following steps:

1. Selection of the first .csv file containing a voiceprint from the dataset #1 folder (e.g., dataset1/analyse.csv).
2. Comparison of the voiceprint from dataset1/analyse.csv with all voiceprints stored in the dataset #3 folder.
3. Recording the calculated similarity scores in results.csv as a new row. The header of results.csv lists the filenames from dataset #3, the first column contains the name of the source file from dataset #1, and each subsequent column stores the similarity score between the source and each corresponding destination file.
4. Repetition of the process for the next .csv file with voiceprint.

The structure of results.csv is identical to that of the previous experiment. The Python code used to perform these comparisons and generate results.csv is given below:

```
def calculate_voiceprints_similarity_matrix_by_folders(input_folder1: str,
                                                       input_folder2: str,
                                                       output_csv: str,
                                                       csv_separator: str = ";",
                                                       csv_decimal_symbol: str = ",",
                                                       suffix1: str = "_1",
                                                       suffix2: str = "_3"):
    folder1 = Path(input_folder1)
    folder2 = Path(input_folder2)

    if not folder1.is_dir() or not folder2.is_dir():
        raise ValueError("One or both input folders are invalid.")

    # Get common CSV files (intersection of names)
    files1 = {f.name: f for f in folder1.glob("*.csv")}
    files2 = {f.name: f for f in folder2.glob("*.csv")}
    common_files = sorted(set(files1.keys()) & set(files2.keys()))

    if not common_files:
        print("No common CSV files found in both folders.")
        return

    # Prepare labeled names
    row_labels = [f"{name}{suffix1}" for name in common_files]
    col_labels = [f"{name}{suffix2}" for name in common_files]

    similarity_matrix = []

    for file1_name in common_files:
        row_label = f"{file1_name}{suffix1}"
        row = [row_label]  # First cell in the row is the labeled name

                        emb1 = pd.read_csv(files1[file1_name], sep=csv_separator, decimal=csv_decimal_symbol,
header=None).iloc[0].to_numpy()

        for file2_name in common_files:
                            emb2 = pd.read_csv(files2[file2_name], sep=csv_separator, decimal=csv_decimal_symbol,
header=None).iloc[0].to_numpy()
```

```
            if emb1.shape[0] != emb2.shape[0]:
                raise ValueError(f"Dimension mismatch between {file1_name} and {file2_name}")

            similarity = np.dot(emb1, emb2) / (np.linalg.norm(emb1) * np.linalg.norm(emb2))
            row.append(round(similarity, 2))

        similarity_matrix.append(row)

    # Add headers
    columns = ["file"] + col_labels
    df = pd.DataFrame(similarity_matrix, columns=columns)

    # Save
    df.to_csv(output_csv, sep=csv_separator, decimal=csv_decimal_symbol, index=False)

    print(f"Similarity matrix saved to {output_csv}")
```

The results of comparison are given in Table 13.

Table 13
Comparison for dataset # 3 and dataset # 1

		Dataset #3																
Dataset #1		analyse	attack	auth-tion	autopilot	check	defence	destroy	find	follow	freeze	go	home	identity	listen	stop	wait	Watch
	analyse	0,54	0,52	0,47	0,49	0,6	0,53	0,56	0,53	0,62	0,51	0,61	0,65	0,47	0,53	0,57	0,54	0,55
	attack	0,53	0,56	0,46	0,46	0,6	0,49	0,55	0,48	0,56	0,5	0,55	0,55	0,48	0,57	0,54	0,55	0,52
	auth-tion	0,54	0,51	0,49	0,46	0,55	0,52	0,52	0,56	0,49	0,47	0,48	0,53	0,5	0,52	0,49	0,51	0,54
	autopilot	0,56	0,59	0,54	0,5	0,65	0,53	0,54	0,56	0,6	0,55	0,52	0,6	0,53	0,57	0,58	0,56	0,56
	check	0,57	0,55	0,51	0,49	0,63	0,54	0,58	0,53	0,64	0,57	0,6	0,62	0,49	0,58	0,58	0,58	0,6
	defence	0,47	0,39	0,4	0,37	0,46	0,45	0,44	0,46	0,44	0,43	0,45	0,48	0,41	0,46	0,42	0,47	0,46
	destroy	0,59	0,54	0,5	0,47	0,63	0,56	0,51	0,59	0,58	0,62	0,57	0,68	0,54	0,56	0,61	0,6	0,64
	find	0,54	0,51	0,46	0,42	0,61	0,53	0,51	0,54	0,57	0,53	0,56	0,6	0,47	0,53	0,53	0,54	0,54
	follow	0,62	0,6	0,57	0,56	0,72	0,65	0,67	0,61	0,67	0,63	0,67	0,69	0,59	0,61	0,63	0,62	0,66
	freeze	0,51	0,5	0,47	0,44	0,64	0,51	0,54	0,55	0,65	0,59	0,63	0,67	0,47	0,54	0,57	0,6	0,62
	go	0,53	0,53	0,52	0,48	0,7	0,58	0,61	0,56	0,68	0,62	0,69	0,72	0,52	0,58	0,64	0,6	0,64
	home	0,5	0,52	0,47	0,47	0,66	0,53	0,57	0,51	0,63	0,57	0,64	0,71	0,49	0,52	0,65	0,57	0,63
	identity	0,55	0,52	0,49	0,45	0,61	0,53	0,53	0,58	0,59	0,51	0,61	0,67	0,52	0,53	0,59	0,57	0,58
	listen	0,52	0,46	0,41	0,4	0,54	0,49	0,49	0,51	0,51	0,48	0,53	0,57	0,46	0,52	0,5	0,51	0,52
	stop	0,48	0,5	0,43	0,42	0,61	0,47	0,55	0,45	0,59	0,52	0,64	0,62	0,45	0,53	0,6	0,55	0,6
	wait	0,53	0,57	0,51	0,5	0,71	0,57	0,6	0,54	0,67	0,61	0,65	0,68	0,52	0,59	0,64	0,62	0,64
	watch	0,52	0,54	0,46	0,46	0,68	0,53	0,57	0,51	0,65	0,59	0,63	0,67	0,48	0,58	0,63	0,61	0,62

The resulting score for this scenario is presented in Table 14:

Table 14
Resulting scores

Range	Percentage (%)	Description
[0 : 0,6]	77,94	Failed scenario
(0,6 : 0,8)	22,06	Threshold scenario
[0,8 : 1]	0,0	Succeed scenario
Total number: 272		

The results obtained in this test are consistent with these expectations. No successful matches or false positive results were expected when comparing audio recordings from different speakers.

3.3. Experiment 3: calculating the similarity for the short audio records with help of voiceprint built on audio record of good quality

In two previous scenarios we used voiceprints received from short audio records and potentially it could affect the quality of voiceprints due to the lack of data required for the Resemblyzer voiceprint generator to analyze. To address this, we now examine a scenario in which one of the voiceprints in each comparison pair is derived from a longer recording. Specifically, we use dataset #2 (male voice, 15,78 seconds) and dataset #4 (female voice, 12,88 seconds). This experiment involves calculating similarity scores between voiceprints from dataset #1 and the voiceprint from dataset #2, and likewise between voiceprints from dataset #3 and the voiceprint from dataset #4.

To perform the above estimates, the following algorithm was implemented:

1. Selection of the first .csv file containing a voiceprint from the target folder (e.g., analyze.csv).
2. Comparison of the voiceprint from analyze.csv with a good quality voiceprint (e.g., from dataset #2).
3. Storing the similarity value in memory.
4. Proceeding to the next .csv file with voiceprint and repetition of the comparison.
5. Recording the calculated similarities in the results.csv file. The header of this file lists the names of the files involved in the comparison, and the single row (since there is only one in this case) contains the corresponding similarity values.

The Python code used for these comparisons and for generating results.csv is:

```
def calculate_voiceprints_similarity_single_to_all(voiceprint_csv_file: str,
                                                   input_folder: str,
                                                   output_csv: str,
                                                   csv_separator: str = ";",
                                                   csv_decimal_symbol: str = ","):
    """Compare one voiceprint with all voiceprints in a folder and save similarity results.

    Args:
        voiceprint_csv_file (str): Path to the input voiceprint CSV file (one line, no header).
        input_folder (str): Folder containing CSV files to compare against.
        output_csv (str): Path to output CSV file (single-row result).
        csv_separator (str, optional): CSV column separator. Defaults to ";".
        csv_decimal_symbol (str, optional): Decimal symbol. Defaults to ",".
    """
    folder = Path(input_folder)
    if not folder.is_dir():
        raise ValueError(f"{input_folder} is not a valid folder.")

    # Load source voiceprint
    df_source = pd.read_csv(voiceprint_csv_file, sep=csv_separator, decimal=csv_decimal_symbol, header=None)
    source_embedding = df_source.iloc[0].to_numpy()

    # Gather and prepare comparison files
    csv_files = sorted(folder.glob("*.csv"))
    if not csv_files:
        print("No CSV files found in the folder.")
        return

    results = {}
    for file in csv_files:
        df_target = pd.read_csv(file, sep=csv_separator, decimal=csv_decimal_symbol, header=None)
        target_embedding = df_target.iloc[0].to_numpy()

        if source_embedding.shape[0] != target_embedding.shape[0]:
            raise ValueError(f"Dimension mismatch: {file.name}")

        similarity = np.dot(source_embedding, target_embedding) / (
```

```
        np.linalg.norm(source_embedding) * np.linalg.norm(target_embedding)
    )
    results[file.stem] = round(similarity, 2)

# Create single-row DataFrame
df_result = pd.DataFrame([results])
df_result.to_csv(output_csv, sep=csv_separator, decimal=csv_decimal_symbol, index=False)

print(f"Similarity results saved to {output_csv}")
```

Results of comparison of dataset #1 with dataset #2, and dataset #3 with dataset #4 are illustrated in Fig. 2.

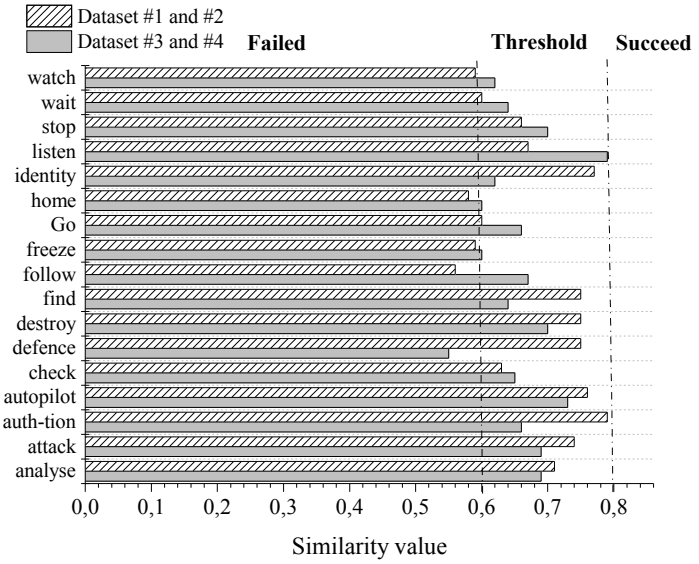

Figure 2: Comparison of dataset #1 with dataset #2, and dataset #3 with dataset #4

The resulting scores are presented in Tables 15 and 16.

Table 15
Dataset #1 and #2

Range	Percentage (%)	Description
[0 : 0,6]	17,65	Failed scenario
(0,6 : 0,8)	82,35	Threshold scenario
[0,8 : 1]	0,0	Succeed scenario
Total number: 17		

Table 16
Dataset #1 and #2

Range	Percentage (%)	Description
[0 : 0,6]	35,29	Failed scenario
(0,6 : 0,8)	64,71	Threshold scenario
[0,8 : 1]	0,0	Succeed scenario
Total number: 17		

In both cases, no successful matches were observed, with a high proportion of results falling into the threshold and false positive categories. These findings indicate that voiceprints generated from short audio recordings of 1-1.5 seconds are not suitable for reliable authentication when compared against a high-quality voiceprint.

3.4. Experiment 4: calculating the similarity for the medium audio records using the voiceprints built on audio record of high quality

In the following, we tried to increase the quality of input audio records by increasing the length of audio records and file size to approximately twice in comparison with the original datasets #1 and #3. Details of dataset #5 are provided in Table 17.

Table 17
Physical characteristics of dataset #5

	Dataset #1. File size (Bytes)	Dataset #3. File size (Bytes)	Dataset #5 File size (Bytes)	Average file size increase (times)	Dataset #1. Length of audio track (seconds)	Dataset #3. Length of audio track (seconds)	Dataset #5 Length of audio track (seconds)	Average length of audio records increase (times)
analyse	222802	286162	606802	2,421994	1,16	1,49	3,16	2,422472
attack	257362	238162	589522	2,382966	1,34	1,24	3,07	2,383426
auth-tion	288082	353362	698962	2,202147	1,50	1,84	3,64	2,202464
autopilot	272722	353362	579922	1,883789	1,42	1,84	3,02	1,884032
check	247762	228562	616402	2,592375	1,29	1,19	3,21	2,592926
defence	284242	291922	581842	2,020069	1,48	1,52	3,03	2,020359
destroy	293842	255442	606802	2,22028	1,53	1,33	3,16	2,22065
find	280402	263122	514642	1,895639	1,46	1,37	2,68	1,89591
follow	272722	265042	554962	2,064382	1,42	1,38	3,89	2,779139
freeze	268882	247762	549202	2,129595	1,40	0,55	2,86	3,621429
go	272722	230482	505042	2,02155	1,42	1,2	2,63	2,02189
home	257362	272722	524242	1,97962	1,34	1,42	2,73	1,979924
identity	305362	284242	560722	1,904473	1,59	1,48	2,92	1,904725
listen	230482	282322	566482	2,232162	1,20	1,47	2,95	2,232568
stop	195922	220882	533842	2,570817	1,02	1,15	2,78	2,571441
wait	226642	228562	514642	2,261189	1,18	1,19	2,68	2,261644
watch	236242	280402	549202	2,141683	1,23	1,46	2,86	2,142054

As a first step, we repeat the initial experiment, calculating similarity scores by comparing the voiceprints from dataset #5 with one another. The results of these comparisons are presented in Table 18.

The resulting score is presented in Table 19.

The results were found to be very promising. Accordingly, no false positive cases were observed, and the proportion of threshold values is minimal. Next, we repeat the experiment by comparing dataset #5 with the high-quality voiceprint obtained from dataset #2. The results of this comparison are presented in Fig. 3.

The resulting score is presented in Table 20.

The obtained findings indicate strong performance, suggesting that audio recordings with a duration of at least 2.63 seconds and a file size of 495 KB or greater can produce reliable voiceprints suitable for secure user authentication.

Table 18
Comparison for dataset #5

	analyse	attack	auth-tion	autopilot	check	defence	destroy	find	follow	freeze	go	home	identity	listen	stop	wait	watch
analyse	1	0,89	0,91	0,86	0,8	0,82	0,84	0,8	0,83	0,86	0,86	0,84	0,83	0,83	0,85	0,83	0,8
attack	0,89	1	0,93	0,88	0,88	0,9	0,9	0,86	0,88	0,92	0,91	0,92	0,93	0,92	0,92	0,9	0,9
auth-tion	0,91	0,93	1	0,9	0,87	0,89	0,91	0,87	0,89	0,9	0,89	0,88	0,89	0,91	0,89	0,88	0,87
autopilot	0,86	0,88	0,9	1	0,88	0,87	0,91	0,84	0,9	0,92	0,89	0,89	0,91	0,88	0,85	0,83	0,81
check	0,8	0,88	0,87	0,88	1	0,9	0,89	0,89	0,91	0,87	0,85	0,87	0,88	0,89	0,82	0,84	0,82
defence	0,82	0,9	0,89	0,87	0,9	1	0,89	0,87	0,9	0,87	0,87	0,86	0,89	0,89	0,85	0,88	0,85
destroy	0,84	0,9	0,91	0,91	0,89	0,89	1	0,9	0,9	0,89	0,86	0,87	0,88	0,9	0,87	0,83	0,83
find	0,8	0,86	0,87	0,84	0,89	0,87	0,9	1	0,85	0,84	0,85	0,86	0,85	0,88	0,85	0,79	0,79
follow	0,83	0,88	0,89	0,9	0,91	0,9	0,9	0,85	1	0,89	0,87	0,88	0,89	0,9	0,83	0,85	0,8
freeze	0,86	0,92	0,9	0,92	0,87	0,87	0,89	0,84	0,89	1	0,94	0,95	0,93	0,93	0,93	0,91	0,9
go	0,86	0,91	0,89	0,89	0,85	0,87	0,86	0,85	0,87	0,94	1	0,94	0,92	0,9	0,91	0,9	0,88
home	0,84	0,92	0,88	0,89	0,87	0,86	0,87	0,86	0,88	0,95	0,94	1	0,94	0,91	0,9	0,89	0,9
identity	0,83	0,93	0,89	0,91	0,88	0,89	0,88	0,85	0,89	0,93	0,92	0,94	1	0,9	0,89	0,89	0,87
listen	0,83	0,92	0,91	0,88	0,89	0,89	0,9	0,88	0,9	0,93	0,9	0,91	0,9	1	0,9	0,87	0,87
stop	0,85	0,92	0,89	0,85	0,82	0,85	0,87	0,85	0,83	0,93	0,91	0,9	0,89	0,9	1	0,88	0,91
wait	0,83	0,9	0,88	0,83	0,84	0,88	0,83	0,79	0,85	0,91	0,9	0,89	0,89	0,87	0,88	1	0,92
watch	0,8	0,9	0,87	0,81	0,82	0,85	0,83	0,79	0,8	0,9	0,88	0,9	0,87	0,87	0,91	0,92	1

Table 19
The resulting score

Range	Percentage (%)	Description
[0 : 0,6]	0,0	Failed scenario
(0,6 : 0,8)	1,47	Threshold scenario
[0,8 : 1]	98,53	Succeed scenario
Total number: 272		

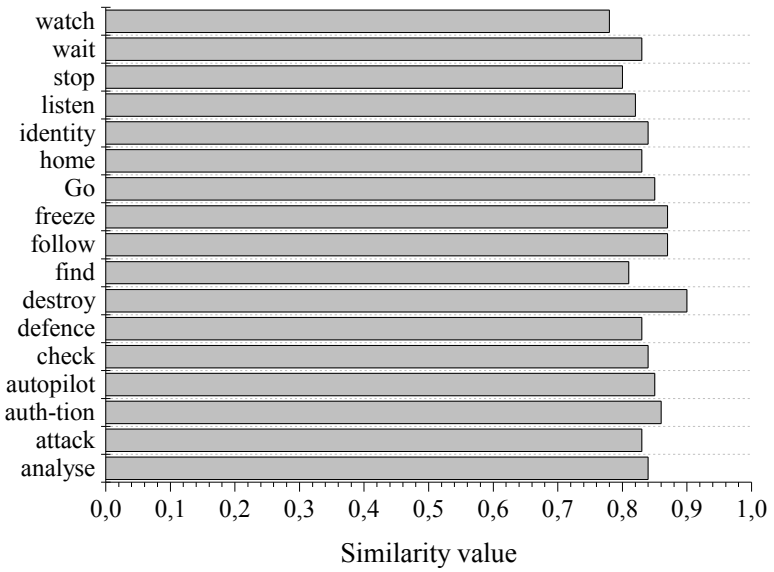

Figure 3: Similarity values obtained from the comparison of dataset #5 and dataset #2

Table 20
The resulting score

Range	Percentage (%)	Description
[0 : 0,6]	0,0	Failed scenario
(0,6 : 0,8)	5,88	Threshold scenario
[0,8 : 1]	94,12	Succeed scenario
Total number: 17		

4. Conclusions

The present study has demonstrated in practice that the Resemblyzer library can be effectively employed on portable devices with limited hardware resources and no access to cloud services. The experiments confirm that the library delivers highly reliable results in voice authentication using relatively short audio recordings (no longer than 3 seconds in duration and under 500 KB in size), which aligns well with the typical timing of short voice commands. These findings highlight the potential of Resemblyzer for integration into autonomous, resource-constrained systems and provide a foundation for further research into its applicability in more complex operational scenarios for independent portable devices.

Declaration on Generative AI

The author(s) have not employed any Generative AI tools.

References

[1] R. Chataut, A. Phoummalayvane, R. Akl, Unleashing the power of IoT: A comprehensive review of IoT applications and future prospects in healthcare, agriculture, smart homes, smart cities, and industry 4.0, Sensors (Basel) 23 (2023) 7194. doi: 10.3390/s23167194.

[2] R. Mouha, Internet of Things (IoT), J. Data Analysis Inform. Proces. 9 (2021) 77. doi: 10.4236/jdaip.2021.92006.

[3] P.K. Sadhu, V.P. Yanambaka, A. Abdelgawad, Internet of Things: security and solutions survey, Sensors (Basel) 22 (2022) 7433. doi: 10.3390/s22197433.

[4] A. Abdulkareem, T.E. Somefun, O.K. Chinedum, F. Agbetuyi, T.E. Somefun, Design and implementation of speech recognition system integrated with internet of things, Internat. J. Electr. Comput. Engineer. (IJECE) 11 (2001) 1796. doi: 10.11591/ijece.v11i2.pp1796-1803.

[5] P. Netinant, T. Utsanok, M. Rukhiran, S. Klongdee, Development and assessment of Internet of Things-driven smart home security and automation with voice commands, IoT 5 (2024) 79. doi: 10.3390/iot5010005.

[6] W. Yang, S. Wang, N.M. Sahri, N.M. Karie, M. Ahmed, C. Valli, Biometrics for Internet-of-Things security: a review, Sensors 21 (2021) 6163. doi: 10.3390/s21186163.

[7] R.M. Hanifa, K. Isa, S. Mohamad, A review on speaker recognition: technology and challenges, Computers Electr. Engineer. 90 (2021) 107005. doi: 10.1016/j.compeleceng.2021.107005.

[8] P. Malhotra, Y. Singh, P. Anand, D.K. Bangotra, P.K. Singh, W.C. Hong, Internet of things: evolution, concerns and security challenges, Sensors 21 (2021) 1809. doi: 10.3390/s21051809.

[9] G. Seo, An end-to-end approach for Korean wakeword systems with speaker authentication. arXiv preprint arXiv:2501.12194. doi: 10.48550/arXiv.2501.12194

[10] S.R. Mogali, O. Ng, J. Tan, T.H. San, K.B. Ng, Voice-over anatomy lectures created by AI-voice cloning technology: a descriptive article, Anatomical Sciences Education 17 (2024) 1686. doi: 10.1002/ase.2524.