# Shazam Signature Representation of Audio for Documents

Ricardo Román-Brenes
Bilkent University
Ankara, Turkey
ricardo@bilkent.edu.tr

Selim Furkan Tekin
Bilkent University
Ankara, Turkey
tekin@ee.bilkent.edu.tr

Syed Asad Shah
Bilkent University
Ankara, Turkey
asad.shah@bilkent.edu.tr

Halil Ibrahim Kuru
Bilkent University
Ankara, Turkey
ibrahim.kuru@bilkent.edu.tr

Asma Jodeiri
Bilkent University
Ankara, Turkey
asma.jodeiri@bilkent.edu.tr

## ABSTRACT

An text document search engine based on Shazam audio fingerprint algorithm was developed. It is capable of matching a query by using a fragment of the original document's content. The Google Text to Speech API[1] service, as well as a raw-to-audio method are used to transforms text into audio. The audio fingerprints method of Shazam is used to generate signature of a text from the audio file. The engine achieved a 60% match hit ratio when using queries of around 80% of the original file. The system is scalable; storage and computationally efficient as the search is performed using signature instead of the document itself.

## KEYWORDS

Shazam, text query, matching, audio fingerprint

## 1 INTRODUCTION

*S*ound is a pressure wave with distinct frequencies and human ear perceive it in 1-D wave-form, convert it into frequency dependent nerve firing and the auditory cortex of the brain will do the further refinement. Whereas computer seen it as sinusoidal waveform. The attributes which makes sound musical are pitch, timbre and loudness.

Shazam is a widely known and used service for music identification [2]. The system is based on converting the documents (in this case, songs) into a signature in order to reduce the dimensionality of each input [6]. A database is built using these signatures for just as many songs as possible. When a query is performed, the song is converted as well and checked against the database. Since comparing the signature of a song is a lot easier than comparing the song itself, the system responds quickly. The signatures are different enough to be able to avoid incorrect identification, yet noise, or bad quality of a recording might affect the results in a negative way.

Using this as base, content-based document query search can also be performed by converting the text into audio and then generate signature of it, loaded into a database where they can be queried.

The motivation behind this project is to explore how algorithms that were built for different purposes can be applied to different data with little effort and hopefully with good rate of success. This type of fingerprinting system might help build document search engines that respond faster than the standard databases.

Throughout the years, the Shazam[6] algorithm has also been used in fields other than music retrieval. A project[2] has used a fingerprint algorithm similar to Shazam for face recognition to increase performance. This approach creates fingerprints of faces and it is more capable of dealing with larger degree of variability in ambient lighting, pose, expression, occlusion, face size, and distance from the camera than other algorithms. In another research[3], the Shazam algorithm is being used for retrieving movie information and optimization based on movie audio which improves the average retrieval accuracy of the algorithm by 0.82%.

[1]https://gtts.readthedocs.io/en/latest/
[2]http://www.Shazam.com

## 2 OBJECTIVE

The main objective of this work is to study how the algorithm behind the service *Shazam* would perform when applied to text documents in an Information Retrieval System (IRS).

In order to achieve this, several steps must be taken, which are described in the following section.

## 3 METHODOLOGY

For Shazam's audio fingerprinting to work on text files, first said files were converted to audio. After that, all the collection of documents will be transformed into their fingerprints and stored. Lastly, from each document 3 types of queries were generated and matched against the collection to check the system's peroformance.

The general workflow of this process is shown in figure 1. The finer points of the dataset, conversion, fingerprinting and UI presentation will be laid out below.

### 3.1 Text-to-Audio

*3.1.1 Google Text to Speech.* We initially start to convert each abstract to its audio file. We write a script that parses 500 PubMed Central Full Text Test Collection. For each abstract, the PMID is a unique identifier. Therefore, we extract PMID part as the unique identifier of the audio files. Then, we extract the abstract text for each PMID. Our script uses an API to convert abstracts into audio and save them as '.mp3' files.

We have a couple of APIs currently available for text-to-speech conversion in Python language. The Google Text to Speech API[3],

[3]https://gtts.readthedocs.io/en/latest/

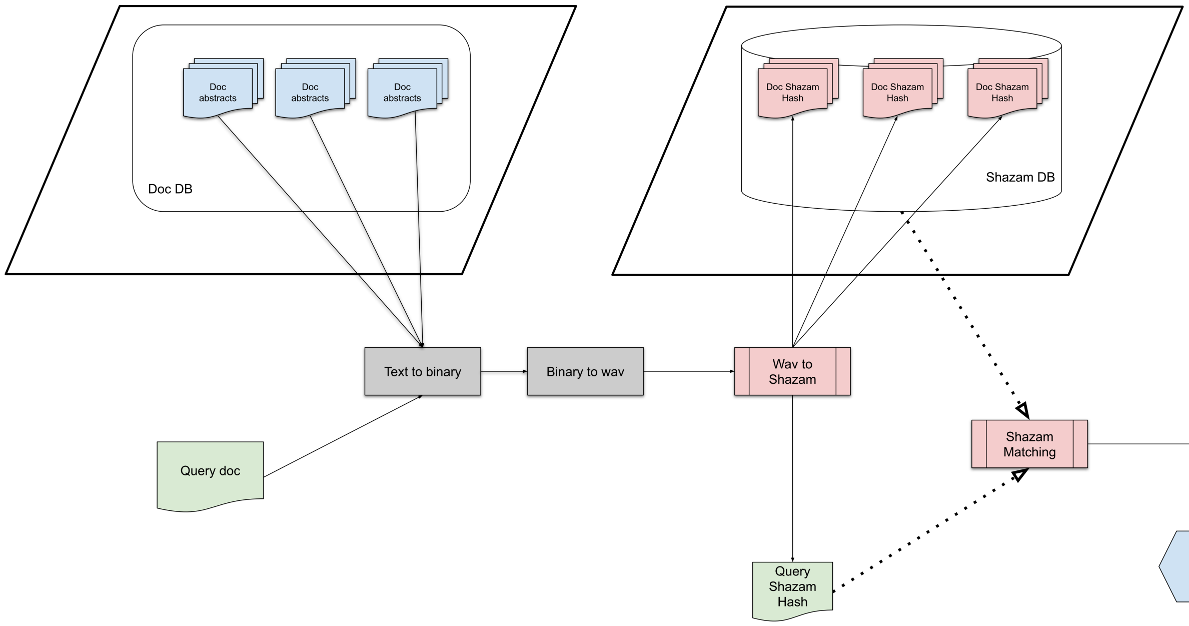


**Figure 1: General flow of the IR system.**

also known as the gTTS API, is one of these APIs. gTTS is a simple tool that transforms text into audio files and we save them as mp3 files. The gTTS API provides text-to-speech conversion in a variety of languages such and English, French, German, Mandarin and even Turkish. We can also change the speed of speech as either fast of slow.

However we should note that it gets *HTTP request error* after some number of requests since it sends a request to Google Cloud for conversion. Therefore, we could not run our script end-to-end. We split the abstracts into chunks, each containing 50 abstracts. We run script for the a chunk, save abstracts as mp3 files, and then wait around 1 hour. Then, we continue with another chunk so that we did not get *HTTP* error.

Figure 2 shows a sample audio in time and frequency domain. This audio is speech of a woman robot voice reading the input text. Thus, we expect the frequency range between 180 - 250 Hertz. However, as in figure 2 our speech signal also includes high frequencies.

Figure 3 shows a spectrogram of a sample audio as a 2-D array with amplitude as a function of frequency and time. In spectrogram, X-axis represents time, Y-axis represents frequency and the density of the shading represents the amplitude.

*3.1.2 Google Text to Speech.* Another method of generating the audio files from text was implemented using a simple byte conversion. The text is converted to the integer ASCII (or Unicode) values of each character and passed to Numpy [4] to generate an array. This array is later interpreted as data by Pydub [5] which in turn adds the header of an MP3 file. These "audio" files have no real audio data hence no sound is produced when played.

[4]http://https://numpy.org/
[5]http://pydub.com/

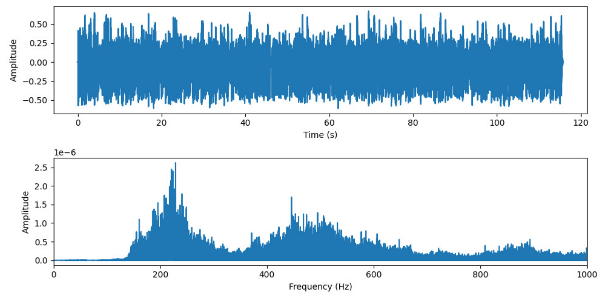


**Figure 2: Sample audio in time and frequency domain. Signal contains wide range of frequencies from 100-1000 Hertz and there is high dense of signals located in range 150-350 Hertz and 450-600 Hertz.**

## 3.2 Audio fingerprinting

Audio or acoustic fingerprint works by extracting relevant characteristics in an audio fragment. Presented afterwards with an unidentified piece of audio content, the characteristics of that piece are calculated and matched against those previously stored [5].

*3.2.1 Features of audio Fingerprinting mechanism:*

- The fingerprint generation and searching process of an audio for the finest match from the large database of fingerprints should be very efficient.
- The document identification process should be accurate.
- It should be expandable enough to maintain the large fingerprint database.

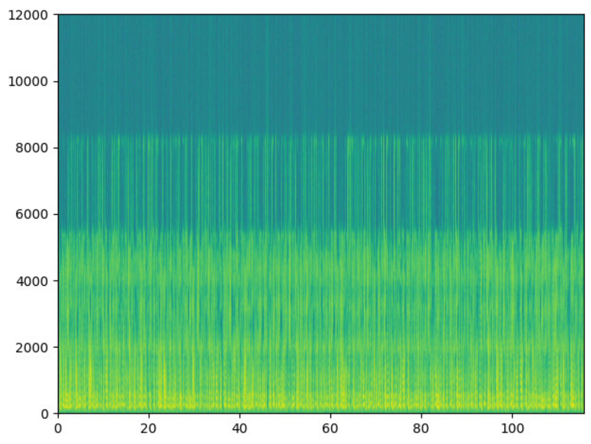


**Figure 3: spectrogram of a sample audio**

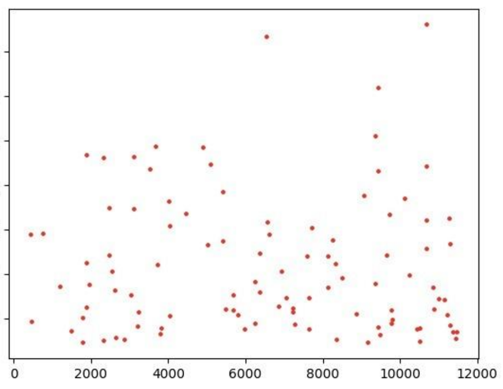


**Figure 4: Amplitude of FFTs**

- Its performance is measured by number of wrong and correct identifications.

*3.2.2 Fingerprinting generation mechanism :* As we know Shazam's fingerprinting method only work with frequencies, we convert the analog signal into digital and then use Fourier Transform to convert function of time into function of frequencies. In order to generate the audio fingerprint of a document, we read the obtained sampling frequency and build a spectrogram from the obtained sampling frequency by applying the FFT (fast fourier transform) method on the digital sound to reduce the spectrum leakage for a good frequency resolution.

In particular, Shazam, exploits the highest peaks in the audio spectrogram to store, the frequency, the time and the intensity of those peaks, effectively reducing both dimensionality and size of each audio piece [6] [4].When we compare spectrograms with the other speech signals, we noticed that, points with high amplitudes can be representative for the text audio. Thus, we select the points with the highest amplitude as our anchors for the hashing. We follow the same procedure in Shazam algorithm as discussed below. The code for our hashing and analysis are available at https://github.com/rica01/BU-spring2021-cs533.

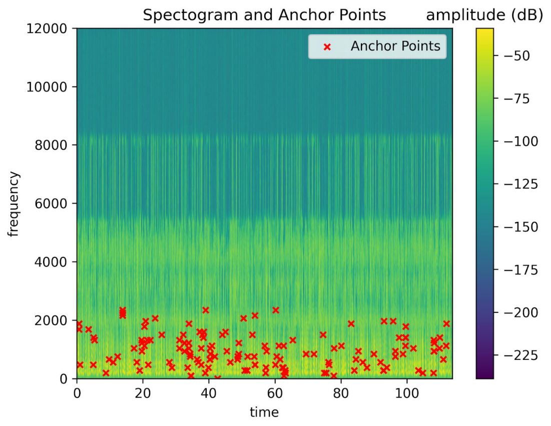


**Figure 5: spectrogram and Anchor points.**

We need to keep the loudest notes from different frequency bands of an audio which requires filtering. For that, we perform the 512 bins histogram inside 6 logarithmic bands for each FFT results. For each band, we kept the strongest bin of frequencies and then compute the average value of these 6 strongest bins. We keep the bins whose values are greater than the mean as you can see in figure 4.

- very low sound band (from bin 0 to 10)
- low sound band (from bin 10 to 20)
- low-mid sound band (from bin 20 to 40)
- mid sound band (from bin 40 to 80)
- mid-high sound band (from bin 80 to 512)

The selected anchor points corresponds to frequency-time points on spectrogram. We show an example of anchor points in figure 5.

The set of points in the neighborhood of the anchor points are target zone used to generate a combinatory hash. The algorithm goes through every single point and look for the target zone of points in the neighboured as you can see in the figure 6. As when we record an audio the points that are closely related in time are likely to be captured. So a small portion of an audio file can be seen as pair of points between a start point and a target zone as shown in figure 7.It will enable the system to create a hash identifier of each point in the target zone using:

- The frequency at which the anchor point is located.
- The frequency at which the point in the target zone is located.
- The time difference between the time when the point in the target zone is located in the audio.
- The time when the anchor point is located in the audio.

## 3.3 Storing Fingerprints

After having multiple target zones, we generate the addresses for each point based on the target zone. The size of the generated

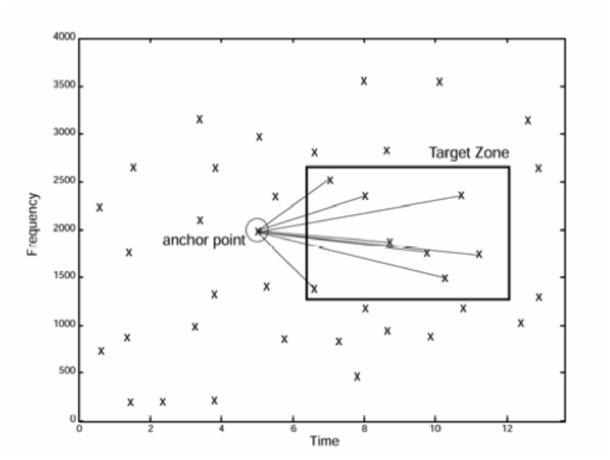


Figure 6: Anchor Point and Target Zone.

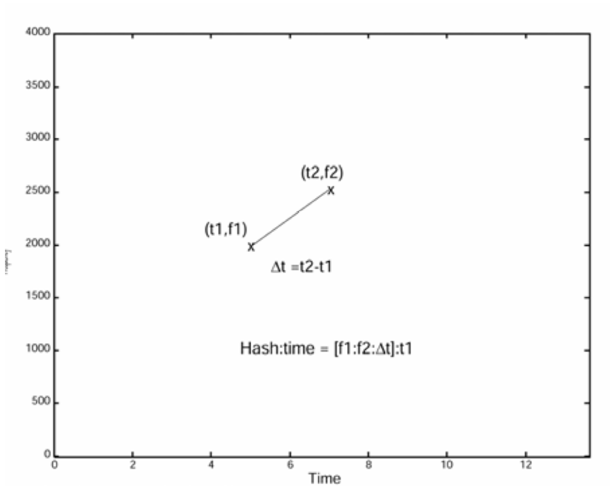


Figure 7: Hash Time.

address is 32 bit integer, 9 bits for the frequency as we have only 512 possible frequencies, and 14-bits for the delta time.

- 9 bits for the "frequency of the anchor"
- 9 bits for the "frequency of the point"
- 14 bits for the "delta time between the anchor and the point"

Similarly the couple ("time of anchor" ; "text Id") can be coded in a 64-bit integer (32 bit for each part).We have implemented the fingerprint table as simple array list of 64-bit integers as shown in figure 8 where:

- the index of the array is the 32-bit integer address
- the list of 64-bits integers is all the couples for this address .

## 3.4 Query Matching

When a new sample comes into the system, the same steps replicated to generate the fingerprint and each hash in the sample is searched in the database. After that, a collection of matches is obtained, where each match contains a address that points to couples.

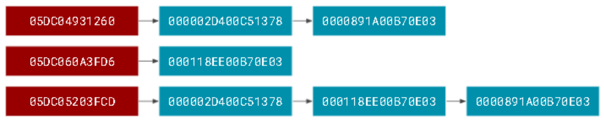
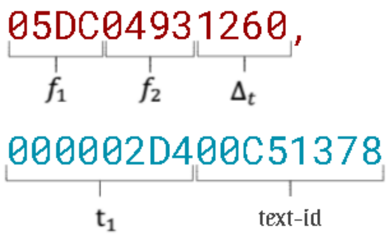
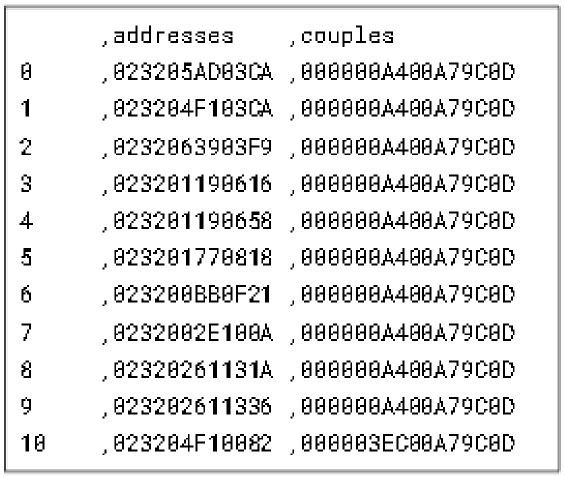


Figure 8: Fingerprints in Database.



(a)



```
      ,addresses    ,couples
0     ,023205AD03CA ,000000A400A79C0D
1     ,023204F103CA ,000000A400A79C0D
2     ,0232063903F9 ,000000A400A79C0D
3     ,023201190616 ,000000A400A79C0D
4     ,023201190658 ,000000A400A79C0D
5     ,023201770818 ,000000A400A79C0D
6     ,023200BB0F21 ,000000A400A79C0D
7     ,0232002E100A ,000000A400A79C0D
8     ,02320261131A ,000000A400A79C0D
9     ,023202611336 ,000000A400A79C0D
10    ,023204F10082 ,000003EC00A79C0D
```

(b)

Figure 9

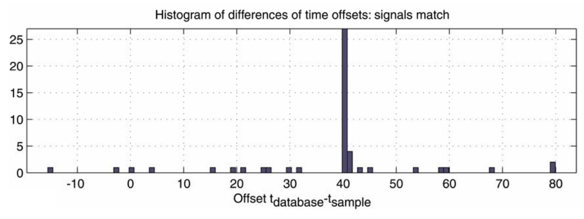


Figure 10: Histogram of matched record

Then, we extract time value from each couple and take the time difference between the anchor of record. This process result time differences for each audio id. When we plot the histogram of an matched song, as shown in figure 10, we observe a high intensity in a song range.

This process is repeated for each audio in the couples, and we note the highest count in each audio. Finally, we ranked the matched audio and return the one with the highest counts as the searched audio.

### 3.5 Web UI

In order to provide portability, the system, in particular its UI is developed in Python, using Flask[6] as its web server. A standard interaction using forms guides the user through the process of selecting the files to be fingerprinted, transform them, select a query and see the results. This procedure can be seen in the YouTube video at https://youtu.be/2iD4MAEYlZQ

## 4 EXPERIMENTATION AND RESULTS

In experimentation we performed our algorithm on a benchmark dataset. For evaluation and comparison of our algorithm, we selected a baseline method as *doc2vec*.

### 4.1 Dataset

The chosen test collection is the 500 PubMed Central Full Text Test Collection, and can be found at https://ii.nlm.nih.gov/DataSets/FullText/PubMedCentral.medline. It was used in the Full Text experiment to date reported on in the 2005 AMIA paper, "Semi-Automatic Indexing of Full Text Biomedical Articles, AMIA 2005" [1]. In this collection, papers from several medical topics are stored as abstract and metadata. The abstract will be used as the input for the algorithm.

A fragment of a record of said collection is shown next:

```
PMID- 10984461
OWN - NLM
STAT- MEDLINE
DA - 20001016
... ... ...
DP - 2000 Sep-Oct
TI - Opportunities at the intersection of bioinformatics and health informatics: a case study.
PG - 431-8
AB - This paper provides a "viewpoint discussion" based on a presentation made to the 2000 Symposium of
the American College of Medical Informatics. It discusses potential opportunities for researchers in health
informatics to become involved in the rapidly growing field of bioinformatics, using the activities of the Yale
Center for Medical Informatics as a case study. One set of opportunities occurs where bioinformatics research
itself intersects with the clinical world.
... ... ...
AD - Yale University, New Haven, Connecticut, USA.
... ... ...
MH - Research Support, U.S. Gov't, P.H.S.
EDAT- 2000/09/14 11:00
MHDA- 2000/10/21 11:01
PST - ppublish
SO - J Am Med Inform Assoc 2000 Sep-Oct;7(5):431-8.
```

Each abstract in the dataset is converted to audio as we explained in section 3.1. Using these audio, we created our hash dataset of our Shazam algorithm. Then, we created queries using sections of abstracts. Since Shazam algorithm is record length dependent, we used queries with section length equal to 50% and 80% of an abstract. As the input, we used the audio of queries belonging to an abstract. Then, we tried to retrieve the same abstract that we used for creating the query. Abstracts with the highest matched rate are retrieved. If the abstract correspond to query in retrieved $k$ documents, we accepted the query as hit.

[6] https://flask.palletsprojects.com/

| | Shazam | | Doc2Vec | |
|---|---|---|---|---|
| Evaluation | 50% | 80% | 50% | 80% |
| top-5 | 31.26 | 57.14 | 88.63 | 91.15 |
| top-10 | 36.59 | 64.84 | 91.15 | 91.36 |

**Table 1: Top-5 and top-10 scores of Shazam and Doc2vec methods on queries with different proportion of lengths**

### 4.2 Baseline Method: Doc2vec

Using doc2vec [7], a baseline for comparison was established. Each document was used to train a model and then said model was queried with documents created from using 50% and 80% of the text of the original. As can be seen in table 1, scores of around 90% were hit both on the top-5 and top-10 documents using either queries with 50% lenght or 80%.

### 4.3 Results

We show the results of experiments on 1. Shazam score performed worse compared to our baseline methodology. However the doc2vec has a different input domain than Shazam algorithm, while Shazam algorithm takes audio as input, input of doc2vec is raw text. This domain difference can create high gap between performances. In our experiments, we observed that we need to perform further analysis on effect of query length, document length, sampling frequency and bit rate. All of these factors effect the performance of Shazam algorithm. Furthermore, Shazam algorithm is targeted on songs which has different characteristics compared to voice. Shazam's target is to differentiate songs according to their sound characteristics. However, in our methodology we kept our sound characteristics the same between samples. To solve this problem, in further studies, another time series input can be obtained from input text instead of voice. An example time series can be the distance change between two subsequent word vectors. After that, Shazam algorithm can distinguish texts according to different characteristics of spectrogram of input series.

## 5 CONCLUSIONS

During the course of this project, the use of Shazam's algorithm for text retrieval was demonstrated as a proof of concept.
Even if the results show some positives results, the effects on several factors like document and query length, sampling rate of audio files, bit rate of audio files, amplitude transformations, low or high pass filters, among others must be further studied.
Results also point that the query size is a big influence on hit rate.
Using the byte-to-audio conversion, a phenomenon can arise where the ASCII value of the lower case letters at the end of the alphabet is higher than the capital case letters at the beginning of the alphabet. This can cause a bad configuration of parameters that might miss the peaks in frequency thus missing in turn good anchor points.
In the speech-to-audio conversion, a local method is advised, since Google Cloud have limited use per time frame plus the network transmissions cause delay on building hashes and querying.
Finally, despite using an unoptimized model, a 60% match hit ratio was achieved.

[7] https://radimrehurek.com/gensim/models/doc2vec.html

## REFERENCES

[1] C. W. Gay, M. Kayaalp, and A. R. Aronson. 2005. Semi-automatic indexing of full text biomedical articles. *AMIA Annu Symp Proc* (2005), 271–275.

[2] Tomas Larrain, John S. Bernhard, Domingo Mery, and Kevin W. Bowyer. 2017. Face Recognition Using Sparse Fingerprint Classification Algorithm. *IEEE Transactions on Information Forensics and Security* 12, 7 (July 2017), 1646–1657. https://doi.org/10.1109/tifs.2017.2680403

[3] X. Sun, W. Zhang, and D. Chen. 2018. Movie Retrieval Based on Shazam Algorithm. In *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)*. 1129–1133. https://doi.org/10.1109/ITOEC.2018.8740531

[4] Nicolae Surdu. 2021. How does Shazam work to recognize a song ? https://web.archive.org/web/20161024115723/http://www.soyoucode.com/2011/how-does-shazam-recognize-song Retrieved from the Internet Archive https://web.archive.org/.

[5] Maciej Walczynski and Dagmara Ryba. 2019. Effectiveness of the acoustic fingerprint in various acoustical environments. In *2019 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*. IEEE. https://doi.org/10.23919/spa.2019.8936781

[6] A. L. Wang. 2003. An industrial-strength audio search algorithm. In *ISMIR 2003, 4th Symposium Conference on Music Information Retrieval*. 7–13. in , S. Choudhury and S. Manus, Eds., The International Society for Music Information Retrieval. http://www.ismir.net: ISMIR, October , pp. . [Online]. Available: http://www.ee.columbia.edu/ dpwe/papers/Wang03-shazam.pdf.