# Performance of the Dejavu audio fingerprinting framework in music identification in movies

**Natália Struharová**[1]
[1]Delft University of Technology

## Abstract

Audio fingerprinting is one of the standard solutions for music identification. The underlying technique is designed to be robust to signal degradation such that music can be identified despite its presence. One of the newly emerged applications of a possibly challenging nature is music identification in movies. This paper examines the audio fingerprinting framework Dejavu by evaluating its performance against an existing benchmark created for the context of music identification in movies. The results show that Dejavu's performance matches the expectations derived from the implementation and previous testing, and can be reconfigured to improve the performance in terms of the benchmark.

## 1 Introduction

Audio fingerprinting is a music information retrieval technique well known for its capabilities in music identification. One of the main advantages of an audio fingerprint is its compactness. A fingerprint compresses perceptual information from an audio file into a numeric sequence that is much shorter than the original waveform data. With this, the comparison of different audio files becomes more efficient and effective in the fingerprint domain [1]. Audio fingerprinting is also known for its robustness to distortions. The extracted features that are eventually summarised in form of the fingerprint are "as robust as possible to typical distortions to typical distortions" [2, p. 3], such as those that arise from imperfect transmission channels or background noise.

Both of these features make audio fingerprinting a suitable technique for music identification in challenging conditions and demanding contexts, which is likely the reason why many popular music identification platforms, such as Shazam [3], base their algorithms on this technique [2].

However, as audio fingerprinting became a standard solution for music identification in contexts such as broadcast monitoring [4], new, seemingly similar implementations, were not addressed quite as extensively. Muziekweb [5], the music library of the Netherlands, has recently tested audio fingerprinting in a new context. Based on satisfactory experience with audio fingerprinting in identifying music in radio streaming, they attempted to identify music in movies with the same technique. Contrary to the expectations, audio fingerprinting did not yield satisfactory results in the informal experiments.

The motivation for using audio fingerprinting for music identification in movies is further explained in the paper on the benchmark developed specifically for this context [6]. Using this benchmark, the following research evaluates an open-source audio fingerprinting framework called Dejavu [1] to better understand its suitability to the context in question.

This study investigates Dejavu by addressing the following research questions:

- How does Dejavu perform in identifying music in movies in terms of the benchmark?
- How can Dejavu be configured to compensate for its weaknesses in music identification in movies?

The remaining part of this paper is structured as follows. Section 2 describes related work with its contributions and limitations. In section 3, the Dejavu framework is presented and analysed. Section 4 gives an overview of methods employed in order to answer the research questions. Section 5 elaborates on the practical experiments and their results. In section 6, the results obtained are discussed in terms of findings and their limitations to the scope of research. Section 7 explains the ethical issues related to the research and our efforts in mitigating them. Finally, section 8 concludes the research and suggests directions for future work.

## 2 Related work

While there is no published data on audio fingerprinting in movie music identification, some existing research addresses music identification in similarly challenging environments.

In 2014 Guzman et al. [7] proposed an audio fingerprinting system robust against several signal degradations, including noise addition and temporal desynchronization such as pitch shifting. While the noise categories are treated modularly, only two samples of real-world noise were used in testing. Therefore, the research covers only a very limited subset of categories using only one sample rather than several distinct ones. Moreover, the experiments do not examine various signal-to-noise ratios (SNR). Such experimentation setups do

[1]https://github.com/worldveil/dejavu

not give sufficient insight into how different amplitudes of noise affect the performance.

Another reportedly robust audio fingerprinting framework was presented by Rafi et al. [8] in the same year. Their approach to identifying live versions of songs with audio fingerprinting was tested against a data set containing several live music performances. These often contained background noise together with structural changes such as pitch shifting and tempo changing. The limitation of this evaluation is that the results are presented in terms of overall performance on the data set rather than a modular overview of the effects of different degradation categories. Similarly to the previous study, the SNR is also not considered in this work, further generalising the framework's behaviour rather than systematically analysing the effect of the relative noise amplitude.

While these works are indirectly related to the problem addressed by this research, the experimental setups of the studies was not sufficiently similar to the modular, systematic approach proposed in this paper.

# 3 Dejavu

Dejavu is an open-source audio fingerprinting framework written in Python. The following section provides a brief overview of the framework's identification pipeline as well as a more detailed review of its implementation and performance.

## 3.1 Identification pipeline

The music identification pipeline of Dejavu strongly resembles the basic scenario described in the 'Identification' usage model by Cano and Batlle [1].

Specifically, Dejavu first 'memorises' songs by extracting their fingerprints and storing them in a database. The database also contains a table with the songs' metadata which is linked to their corresponding fingerprints. After populating the database, recognition can be done by querying the audio to be identified. Dejavu first extracts the fingerprints from the input and then compares them to the database to find the best-matching set of fingerprints. Finally, it outputs the metadata of songs from the database that was found to have the best matching fingerprints. As a result, Dejavu returns a list of songs that were matched. The result contains the match name and query time, as well as more technical details that would normally not be of interest to the user. The ranking of a match among other results is determined by the input confidence. This coefficient represents the percentage of the input that was matched to the given result. For example, if our input was completely accurately matched to a given result, the input confidence would be 1.0. The higher this coefficient is, the higher the particular result ranks. Dejavu also calculates fingerprinted confidence, which represents the number of the hashes matched relatively to the entire song in the database. This means that if we input 10 seconds out of a 20 seconds long song, and it is matched perfectly, the fingerprinted confidence would be 0.5. Dejavu's implementation was designed to always return a match, regardless of the confidence coefficients. Therefore, to classify a result as a False Negative (FN), or in other words a missed match, it is necessary to impose threshold on confidence, as this coefficient essentially indicates how likely it is that the match is correct.

Considering this pipeline, the framework's capabilities can be roughly classified into two main tasks: remembering a song by fingerprinting it and storing its metadata and analysing a queried song by fingerprinting it and comparing these fingerprints to the database to retrieve the best match. Both of these tasks make use of the same audio fingerprinting procedure.

## 3.2 Fingerprinting

The purpose of audio fingerprinting is to reduce the dimensionality of audio files such that they can be stored and compared more efficiently. However, to allow for these benefits, unique perceptual features must be preserved as well as possible. Dejavu's fingerprinting mechanism operates under this requirement, as it gradually decomposes the signal until a compact fingerprint of such nature can be formed.

### 3.2.1 Preprocessing

To ready the audio for preprocessing, it is first discretised by sampling. By default, Dejavu samples the signal with a frequency of 44.1 kHz, meaning that 44,100 samples of the signal are extracted per second. This choice of this frequency is justified using the *Nyquist-Shannon sampling theorem*, according to which the signal should be sampled at double the maximum frequency we aim to capture. Since humans generally cannot hear frequencies above 20,000 Hz, the maximum frequency was established at 22,050 Hz, such that no human-audible frequencies would be missed when sampling. Using the theorem, Dejavu's default sampling frequency is double the maximum frequency, or numerically, 2 * 22,050 = 44100 Hz.

### 3.2.2 Frequency domain spectrogram

After preprocessing, Dejavu generates a frequency domain representation of the given audio file by applying Fast Fourier Transform (FFT) to overlapping windows across the signal [9]. This approach is identical to the extraction technique described by Haitsma and Kalker [4], where they motivate this choice by reasoning that the most perceptually significant audio qualities are present in the frequency domain. The transformed windows are combined such that they form a frequency spectrogram of the entire audio file.

The X-axis of the spectrogram represents time relative to the audio file, and the Y-axis represents the frequency values. With that, each pixel within the spectrogram image corresponds to a given time-frequency pair. The colour of this pixel indicates the amplitude of a given frequency at the corresponding time. These spectrograms are then used to extract the features that will constitute the fingerprint.

### 3.2.3 Peak finding

The peak finding algorithm chooses the time-frequency pair coordinate corresponding to an amplitude value that is greater than the amplitudes of its local neighbouring points. This strategy is based on the fact that these high amplitude frequencies are more likely to survive distortion than the low amplitude frequencies around them. Dejavu uses the image of the spectrogram and analyses its pixels to find these peaks.

Specifically, it first applies a high pass filter that emphasises the high amplitude points, and then it extracts the local maxima, which ultimately become the noise-resistant peaks.

Depending on the frequency composition, the number of peaks can range between thousands to tens of thousands per song. However, despite this amount of data, peak extraction stores only a subset of information unique to the song. The rest of the information is discarded, which directly increases the likelihood of peaks of different songs being similar or even identical. If the framework was to match tracks based on the extracted peaks, there would likely be many collisions and incorrect matches. To avoid these collisions as much as possible, Dejavu encapsulates the peaks in the fingerprint using a hash function.

#### 3.2.4 Fingerprint hashing

The peaks are combined into a fingerprint using a hash function. The hash functions used here take a number as input and return a number as output. A good hash function will always return the same output for a given input. In addition, it should also ensure that only a few distinct inputs will be hashed into the same output.

Given a set of peaks and the time differences between them, Dejavu creates several hashes, which constitute the unique fingerprints for the particular track [10].

### 3.3 Configurable parameters

Based on the manual written for Dejavu, several parameters can be configured. The overview of parameters and their effects is available in Table 3.

Most of the configurable parameters present similar trade-offs. If they are set such that more information is stored, the fingerprints take up more storage space and the matching will be more computationally expensive. The upside to such configuration, however, would be better accuracy in matching, as there will likely be fewer collisions of different songs in terms of identical or similar fingerprints.

Knowing the parameters and their trade-offs, it may be possible to configure these parameters to improve the benchmark score based on its results.

### 3.4 Performance

#### 3.4.1 Recall

The developer of Dejavu has tested the framework in different experimental setups. Perhaps the experiment most significant for this research was the one testing microphone-recorded queries that were captured in the presence of humming and talking. The results showed that with only 1 second of a random part of the song, Dejavu can identify it correctly in 60% of the cases. Queries with lengths between 2 and 4 seconds were identified with recall above 95%. In the last two test categories of 5 and 6 seconds, all queries were identified correctly [10].

Given that these queries were noisy and still identified correctly by Dejavu, there is a reason to expect the framework to provide satisfactory performance when tested against the noise categories in the benchmark.

#### 3.4.2 Implementation speed

In terms of fingerprinting, the performance bottleneck is the peak finding stage. However, according to the developer, the matching time that resulted from an experiment on one song with different query lengths was linear. The equation of the linear regression showed that matching time takes approximately 3-times as long as reading the song, with a small constant overhead [10].

Generally, it is advised to make use of a local database, as this decreases latency in the total time of querying as opposed to using remote storage.

### 3.5 Limitations

The developer emphasised that Dejavu performs best when identifying audio with little to no noise. While this statement does not directly provide the range of distortion Dejavu can tackle, it admits that the performance of the framework is bounded by some amount of noise [11]. In the context of music identification in movies, this could translate into a performance drop when identifying music degraded by noise or signal modifications, especially when the distortion is louder than the music. It is also said to perform better on exact signals, indicating that modifications such as pitch shifting and tempo changing may pose a challenge to Dejavu.

## 4 Methodology

In this research, experiments are evaluated in terms of the benchmark for audio fingerprinting frameworks in the context of music identification in movies [6]. To evaluate the frameworks, the benchmark employs a set of criteria that are meant to capture the essential features of a framework in terms of identifying music in movies. The motivation of criteria choice is further explained in [6]. For the purposes of this research, it is sufficient to understand the criteria and their metrics, which are explained in Section 4.1.

The first research question was addressed by testing Dejavu against the benchmark. First, a preliminary test with real movie data was conducted to observe Dejavu's capabilities in a real-world scenario of identifying music in movies. Then, the performance of Dejavu in its default configuration was evaluated against the benchmark where the framework was tested with a set of synthesised data meant to simulate the conditions present in movies.

To tackle the second research question, more experiments were devised based on the initial benchmarking in order to bring Dejavu's performance closer to the optimum. After obtaining the results of these experiments, the configuration that reached the best benchmark scores was marked as the optimal one. This configuration was also used to obtain the search speed and scalability score for the framework. As a final step, this configuration was used to observe the changes in Dejavu's performance on data extracted from real movies.

### 4.1 Criteria

The collective benchmark evaluates a framework in terms of three criteria: robustness, reliability and search speed and scalability. All three criteria are measured with their corresponding metrics.

The **robustness** criterion measures the framework's ability to correctly identify music despite the presence of signal distortion in the input query. The metric used to measure robustness is Recall.

**Reliability** indicates to what extent can the framework be trusted in terms of the correctness of the match. This criterion is measured by Precision.

The final criterion, **search speed and scalability**, measures the execution time of the framework with respect to its search space. It does so by recording the average query time of the data set in four different database setups further explained in [6].

## 4.2 Evaluation set up

### 4.2.1 Data set

The data set that will be queried in evaluation consists of 14,014 tracks that were generated to represent each of the 17 distortion categories dictated by the benchmark. An overview of these categories can be found in Table 4. To generate this data set, different types of noise and signal modifications were applied to the signals of 98 different songs. These songs were randomly selected from movie soundtracks provided by Muziekweb [5].

In each experiment, the evaluation is run with this data set unless explicitly stated otherwise.

### 4.2.2 Database

A local MySQL database was used to store the fingerprints and their corresponding metadata. For all experiments, except for search speed and scalability measuring, conducted in this research, the database of Dejavu was populated with the same dataset of 1407 songs in total. The first part of the data set consists of 907 songs extracted from various movie soundtracks provided by Muziekweb. This set of tracks from movies contains all the tracks used to generate test set. Such a database setup allows all tested songs to be correctly matched and discards the possibility of true negatives. Therefore, the only results we can obtain are True Positives (TP), False Positives (FP) or False Negatives (FN). The second part of the data set constitutes 500 random tracks, also provided by Muziekweb. These were added in order to provide a more realistic setup, where the database contains more different tracks, and consequently, more different fingerprints to compare the query with [3].

### 4.2.3 Dejavu configuration

For the experiments, Dejavu will be tested in its default configuration specified by the developer on GitHub [11] unless explicitly stated otherwise.

As mentioned in Section 3.1, Dejavu does not report FNs by default. To enforce this, a threshold has to be applied to the confidence coefficient of a result. For the initial benchmarking in section 5.2, the threshold of 0 was applied to the input confidence, as this is the coefficient significant enough to have been chosen as the ranking parameter in Dejavu's match sorting. In practice, this means that any result with input confidence equal to 0 will be classified as FN, regardless of its real correctness.

### 4.2.4 Hardware specifications

Search speed and scalability is often largely dependent on the hardware used to run the given framework. The following experiments were run on a laptop running macOS Catalina, with a 2.3GHz 8-core 9th generation Intel Core i9 processor and 16 GB of DDR4 RAM memory.

# 5 Experimental Setup and Results

## 5.1 Experiment 1: Real movie data

To get a preliminary overview of Dejavu's identification of music in movies, a data set consisting of manually extracted and labelled tracks from five movies of the benchmark data was tested.

### 5.1.1 Data set

A total of 130 labelled audio excerpts from five different movies (chosen from the Muziekweb-provided data set) were tested with Dejavu.

### 5.1.2 Results

Dejavu has exhibited a serious decline in performance compared to the results of the developer's recall testing presented in Section 3.4.1. All 130 samples were identified incorrectly, therefore there were no true positives. With the data set extracted directly from movies, it is very difficult for us to identify what type of signal degradations or SNRs rendered these excerpts so challenging for Dejavu. The results of real data testing highlight the necessity of a modular approach to analysing the performance of the framework.

## 5.2 Experiment 2: The default configuration performance

In this experiment, we measured the performance of Dejavu in all benchmark categories in terms of the metrics defined in Section 4.1.

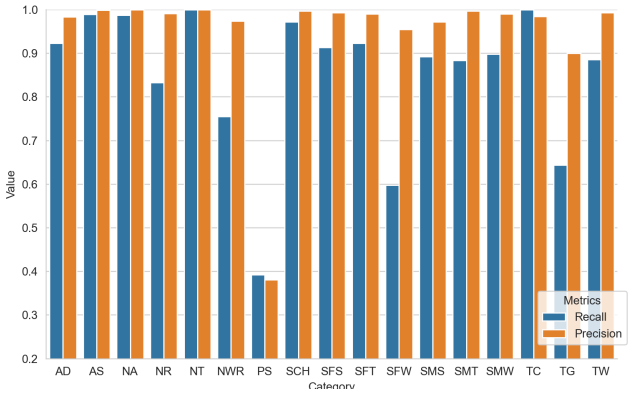
### 5.2.1 Results

For a better overview of overall performance, recall and precision were averaged over the three SNRs for every category. Besides this, the SNRs per category were looked at in terms of the benchmark metrics. The overview with average metrics per category and detailed performance in terms of SNR is shown in Figure 1.

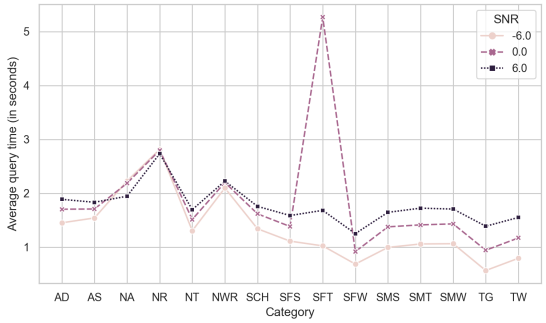
Overall, Dejavu performed quite well on several different categories where the noise was overlapped with the original track. The average recall and precision of the categories was 0.85 and 0.94 rounded to two decimal places respectively. Based on these results, it can be concluded that in its default configuration, Dejavu's recall is stronger than its precision.

As expected based on its previously mentioned limitations, Dejavu's performance deteriorates in lower SNRs relatively to the higher SNRs. When comparing the recall and precision in Figure 1c and Figure 1d, it is evident that recall is more sensitive to SNR than precision. When looking at Figure 1b, it is also visible that querying time is lower for negative SNR which generally scores worse than the positive SNR.

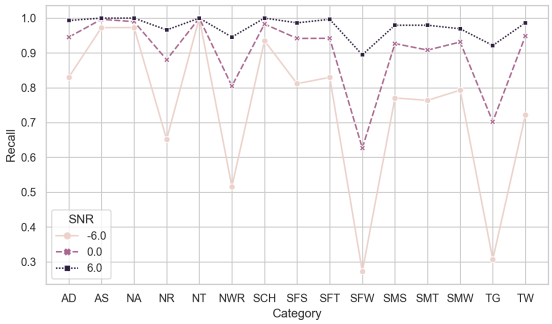
An interesting finding is that the Tempo Changing category (TC) was one of the statistically most precise categories with the highest possible recall, outperforming even some of the



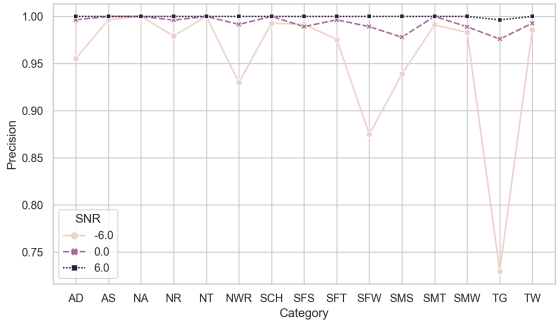

(a) Recall and precision averaged over SNRs




(b) Query time averaged over SNRs




(c) Recall over SNRs




(d) Precision over SNRs

Figure 1: Benchmark score of the default configuration

noise overlapping categories. Despite the forecast based on limitations, Dejavu scored with a perfect recall of 1.0 and near-perfect precision of almost 0.98.

In general, there were two noticeable weaknesses that emerged from the benchmark evaluation:

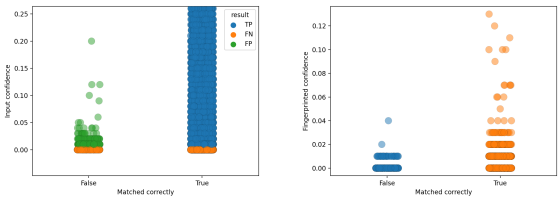
1. Upon analysing the output, it appeared that many False Negative (FN) results have the input confidence of 0 despite returning the correct match from the database. Therefore, there is a reason to suspect that an adjustment to thresholds may improve the metrics of the benchmark.
2. The weakest category for Dejavu was PS with a recall of 0.39 and precision of 0.38, significantly lower in comparison to the average metrics.

To examine these issues further, they will be gradually tackled in the next two experiments.

### 5.3 Experiment 3: Examination of confidence

For the initial benchmarking, a threshold to determine whether the result is a false negative was only imposed on input confidence. Recall that this confidence indicates the ratio of hashes matched with regards to all hashes in the input.

However, with this threshold configuration, a substantial amount of results were classified as false negatives despite being matched with the correct song. A close up of the relevant results in terms of their input confidence and correctness is depicted in Figure 2a.




(a) Results in terms of input confidence

(b) FNs in terms of fingerprinted confidence

Figure 2: Analysis of confidences

These results suggest that the height of the threshold causes many true positives to be classified as false negatives. However, the threshold cannot be lowered from 0.0 with its current precision of two decimal places.

To mitigate the inaccuracy of the input confidence, the fingerprinted confidence was examined in more detail. This was done in order to observe its behaviour in relation to the correctness of matches in false negatives. The results are visualised in Figure 2b.

From these results, it is apparent that positive fingerprinted confidence is more likely to indicate a correct match as opposed to incorrect matches. Therefore, applying a threshold to fingerprinted confidence may lead to an optimisation of the

| | Recall | Precision | Average |
|---|---|---|---|
| **IC = 0 FC = /** | 0.8465 | 0.9373 | 0.8919 |
| **IC = 0 FC = 0** | *0.8973* | 0.9368 | 0.9171 |
| **IC = 0.025 FC = 0** | 0.8776 | 0.9606 | 0.9191 |
| **IC = 0.025 FC = 0.01** | 0.7832 | 0.9793 | 0.8813 |
| **IC = 0.05 FC = 0** | 0.8767 | 0.9621 | **0.9194** |
| **IC = 0.05 FC 0.01** | 0.7825 | *0.9865* | 0.8845 |

Table 1: Confidence threshold configurations

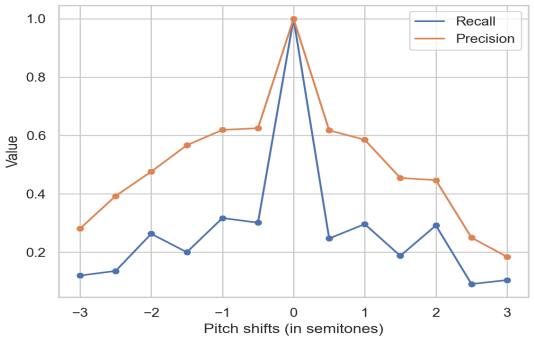



Figure 3: Pitch shifting performance

benchmark metrics. It is expected that there will be a trade-off between recall and precision, as decreasing the number of false negatives will, in this case, lead to some increase in false positives. However, the trade-off may still be favourable, assuming that the optimal configuration is one that maximises the average of these two metrics.

Based on these observations, our hypothesis for the next experiment is that adding an additional confidence threshold may optimise the benchmark metrics. To test this hypothesis, the recall and precision metrics were recalculated with an additional threshold on fingerprinted confidence. Due to the distribution of results, several different combinations of input confidence and fingerprinted confidence threshold were calculated. The results of these recalculations in comparison to the initial benchmarking can be found in Table 1.

When comparing these results, it is apparent that most confidence threshold configurations are superior to the initial configuration where only the input confidence is thresholded. What is also visible is the trade-off between recall and precision, as the highest recall corresponds to the lowest precision and vice versa.

In terms of the optimal configuration, we assume that in the context of identifying music in movies, a balanced trade-off would be desirable. Therefore, the optimal threshold configuration used in the rest of the experiments is the one with input confidence and fingerprinted confidence set to 0.05 and 0 respectively.

## 5.4 Experiment 4: Examining pitch shifting

To further examine Dejavu's behaviour when identifying pitch-shifted tracks, additional semitone deviations were generated and tested. The category was extended to test a range from -3 to +3 semitone pitch shifts with 0.5 semitone increments. The results are visualised in Figure 3.

### 5.4.1 Results

The detailed overview of a greater range of pitch shifts with more granular increments indicates an apparent trend in performance. Precision has the tendency to decrease as the values move away from 0. Recall values, while more erratic, follow a similar trend. This is justified as more pitch-shifting in any direction degrades the signal further from the original.

## 5.5 Experiment 5: Improving pitch shifting

In this experiment, Dejavu will be reconfigured based on literature findings and the trade-off in parameters described in Section 3.3.

### 5.5.1 Pitch shifting in audio fingerprinting

The challenge of identifying pitch-shifted audio has been addressed by several audio fingerprinting frameworks. The task itself appears to be quite complex based on the research conducted by Zhu et al. [12]. In this research, they remark that even one of the most famous, reportedly robust frameworks [4] becomes less accurate in identification when even slight pitch-shifting is present.

As Fenet et al. [13] have remarked, pitch-shifting can be done by scaling all frequencies in a signal with a factor *K*. For example, shifting a tone upwards by an octave corresponds to doubling the tone's frequency, i.e. A4 with frequency 440 Hz would be translated to A5 with frequency 2 * 440 = 880 Hz. Therefore, pitch-shifting can be treated solely in the frequency domain [13].

Since Dejavu uses FFT for transforming the signal into its frequency components, parameters that affect the configuration of FFT are likely to have an effect on how those frequencies are captured. A factor that has a considerable effect on FFT is the window size, which determines how many samples are considered per FFT window. The default value of this parameter is 4096, meaning that 4096 samples are taken into account per FFT window calculation. The size of the window also determines the number of frequency bins returned, or in other words, the frequency resolution. The number of bins is equal to half the window size, which means that 4096 will yield 2048 equally spaced frequency bins [14]. Therefore, by decreasing the window size, we are essentially decreasing the number of frequency bins.

In practice, this would mean that frequency X may be saved in the same bin as a pitch-shifted version of X, X * *K*, given that X and X * *K* are in close enough proximity to be classified in the same bin. Given its implementation, Dejavu would then consider these frequencies to be identical, resulting in treating them as identical peaks, and with that, eventually, as similar or identical fingerprints that are more likely to be matched.

Based on this theory, the next experiment will test the hypothesis that Dejavu performs better on the category of pitch shifting using a smaller FFT window size. Specifically, Dejavu will be tested with an alternative value for the DEFAULT_WINDOW_SIZE, where this parameter will be set to 2048 (half of the original value).

### 5.5.2 Results

The results of the experiment are visualised in Figure 4. These results prove our hypothesis, as Dejavu's performance in PS improved relative to its default configuration.

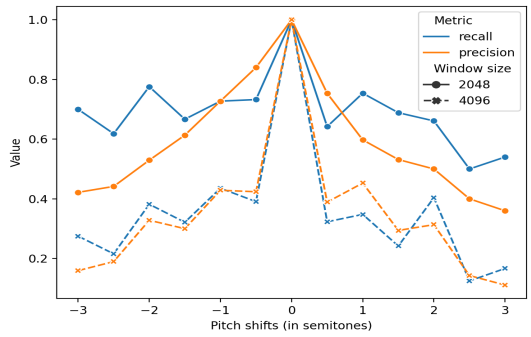



Figure 4: Overview of PS metrics with FFT window size of 2048

However, with regards to other categories, there is a reason to expect that this configuration may hinder the performance. Paradoxically, this is due to the fact that frequency bins are larger and therefore provide less resolution. In practice, this could translate into similar songs being matched due to the close proximity of their frequency peaks and the consequent similarity of their fingerprints. However, the current configuration may provide sufficient granularity. To examine the effect of this configuration on identification in other categories, the benchmark was rerun against the entire dataset.

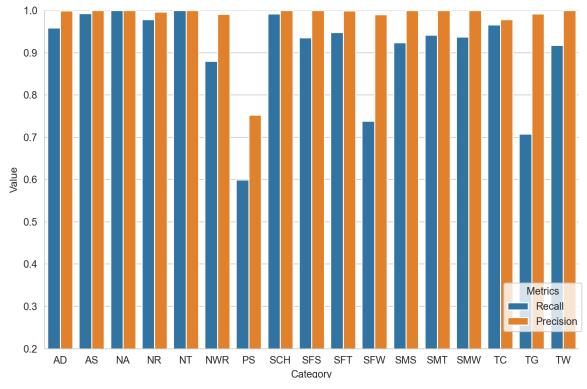



Figure 5: Benchmark metrics with FFT window size of 2048

Despite the potential of the new configuration presenting inaccuracies in fingerprinting, the benchmark has improved relative to the default configuration which employed FFT with a greater resolution. The metrics of recall and precision increased, averaging at 0.9 and 0.98 respectively.

With this configuration being superior to the default one in both recall and precision, and thus proving to be both more robust and more reliable. By our assumption of equal weights of these criteria, the optimal configuration out of those tested within this research.

## 5.6 Validation with real data

Despite the expectations, testing Dejavu in the identification of real movie data with the new configuration showed no improvement from preliminary testing in terms of true positives. However, in some cases, the song was misidentified, but the match corresponded to a different one from the same soundtrack. This was the case especially when the songs in the soundtrack belonged to the same genre. This indicated some positive improvement, as there were no such cases present in the preliminary testing.

## 5.7 Search speed and scalability

The results of search speed and scalability testing based on the configurations defined in [6] are summarised in Section 5.7. For this evaluation, three categories, namely SMT, SMW and TW were omitted due to the time constraints of this research. There is a visible trend of average query time growing with the database size, which is justified, as in larger databases, more comparisons have to be made on average.

| Total tracks | Average query time (in seconds) |
|---|---|
| 10 | 0.965 |
| 98 | 0.991 |
| 196 | 1.357 |
| 980 | 1.487 |

Table 2: Search speed and scalability results

# 6 Discussion

Upon further analysis of incorrect matches, it has been discovered that some tracks in the data set were duplicated. This circumstance gave rise to several invalid false positives for results that were classified correctly. Due to time constraints, the data set could not be fully examined and corrected. However, given the awareness of this circumstance, it is likely that most reliability coefficients are higher in reality than the study shows. In terms of using the selected optimal configuration, fingerprinted confidence for thresholding could turn into a limitation. This confidence is determined by the proportion of the result matched to the song in the database. In a realistic scenario, any excerpt of the song of any length can be input, which directly influences the fingerprinted confidence. Therefore, this confidence may generally not be a valid or a reliable indicator of match correctness. Furthermore, the usage of a smaller FFT window also implies less granularity in captured frequencies. The decrease in resolution proved to work well in the setup of this research, however, in a real-world application where many tracks in the database feature similar dominant frequencies, identification may become more inaccurate with this configuration. Finally, it is important to emphasise that the results obtained in this research do

not accurately represent the performance in a real-world application. The main purpose of this performance assessment is to examine the framework in greater detail and establish its score in terms of the benchmark metrics. This is done to allow for a fair comparison of Dejavu with other frameworks with regards to the context of music identification in movies.

## 7 Responsible Research

One of the most important aspects of research validity is the reproducibility of results. In our research, the data used were synthesised, and for its generation, both the base tracks and noise samples were necessary. The base tracks were obtained from a data set of soundtracks provided by Muziekweb [5]. However, this data is protected by copyright, and therefore could not be published. To mitigate this issue, the list of movies used to extract the soundtracks from was uploaded to GitLab [2].

The noise samples used were extracted from a website called freesound.org [3]. To ensure that the noises from the samples that were overlapped with the base tracks did not contain any silent parts, the samples were further processed to minimise the silence. The exact editing of these samples depended on the time-wise position of silent parts, and therefore, the precise format of samples would be difficult to reproduce. To mitigate this, the edited noise samples were provided and are available for use in attempts to reproduce this research.

Another ethical issue regarding the usage of these samples is their licensing. While most samples extracted were licensed under Creative Commons 0, and thus allowing for use without any attribution, few samples were licensed with the Attribution license.

Both the collection of edited noise samples and overview of attributions can be found 'Movie Labeling' document on GitLab [2].

For fully synthesised data, specifically pitch shifting and tempo changing, a Python script was written and used and is also available on GitLab [2].

Finally, for reproducibility of evaluation of the framework, the evaluation scripts were made available on GitLab [4].

## 8 Conclusions and Future Work

In this paper, we analysed the performance of an open-source audio fingerprinting framework, Dejavu, in music identification in movies with data synthesised to simulate the appropriate conditions. In evaluation against the collective benchmark for movie music identification with audio fingerprinting, Dejavu measured up to the expectations derived from its implementation and prior testing performance. In addition, its configuration was further optimised to target its weaknesses, namely its performance in pitch shifting. This yielded a configuration superior to the default one in terms of the benchmark.

To advance even further in the investigation of Dejavu's optimal configuration, the confidence thresholds should be examined and tested with precision higher than two decimal places, as this accuracy may not provide enough detail to determine the match correctness. Furthermore, the time offset of identification relative to the input should be analysed in order to pinpoint the exact parts of the noise that are easier or more difficult to identify. In addition, query time appears to have no correlation with the other metrics. To better understand the inner workings of the matching algorithm, it is recommended to be examined in more detail.

In terms of further development of the benchmark, more categories of noise degradation potentially occurring in movies due to postprocessing can be examined. To examine categories such as Speech in detail, speech could be synthesised with specific frequencies to observe their effect on Dejavu's performance in the category. In addition, combinations of categories can be further investigated to better understand the effect of overlapping several categories in a given query track.

## 9 Acknowledgements

I would like to thank Dr Cynthia Liem and Dr Jaehun Kim for their supervision and guidance during this research. Next, I would like to thank Casper Hildebrand, Tim Huisman, Ruben Nair and Cas Weaver for collaboration and sharing their enthusiasm throughout the project.

## References

[1] P. Cano and E. Batlle, "A review of audio fingerprinting," *Journal of VLSI Signal Processing*, vol. 41, pp. 271–284, 11 2005.

[2] R. Typke, F. Wiering, and R. Veltkamp, "A survey of music information retrieval systems," p. 153–160, Jan 2005.

[3] A. Wang, "An industrial strength audio search algorithm.," Jan 2003.

[4] J. Haitsma and T. Kalker, "A highly robust audio fingerprinting system with an efficient search strategy," *Journal of New Music Research*, vol. 32, p. 211–221, Jun 2003.

[5] "Muziekweb - the music library of the netherlands."

[6] C. Hildebrand, T. Huisman, R. Nair, N. Struharová, and C. Wever, "Benchmarking audio fingerprinting implementations for music identification in movies," 2021.

[7] J. Guzman, C. Feregrino, A. Menendez-Ortiz, and J. J. Garcia-Hernandez, *A Robust Audio Fingerprinting Method Using Spectrograms Saliency Maps*. Dec 2014. journalAbbreviation: 2014 9th International Conference for Internet Technology and Secured Transactions, ICITST 2014.

[8] Z. Rafii, B. Coover, and J. Han, "An audio fingerprinting system for live version identification using image processing techniques," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, p. 644–648, May 2014.

[2] https://gitlab.ewi.tudelft.nl/cse3000/2020-2021/rp-group-5/rp-group-5-common

[3] https://freesound.org

[4] https://gitlab.ewi.tudelft.nl/cse3000/2020-2021/rp-group-5/rp-group-5-nstruharova

[9] K. Chaudhary, "Understanding audio data, fourier transform, fft, spectrogram and speech recognition," Jun 2021.

[10] W. Drevo, "Audio fingerprinting with python and numpy," Nov 2013.

[11] Worldveil, "worldveil/dejavu."

[12] B. Zhu, W. Li, Z. Wang, and X. Xue, "A novel audio fingerprinting method robust to time scale modification and pitch shifting," in *Proceedings of the 18th ACM international conference on Multimedia*, MM '10, p. 987–990, Association for Computing Machinery, Oct 2010.

[13] S. Fenet, G. Richard, and Y. Grenier, "A scalable audio fingerprint method with robustness to pitch-shifting," *Poster Session*, p. 6, 2011.

[14] J. Burg and J. Romney, "Digital sound and music."

# A Appendix

| Name | Description | Effect of low values | Effect of high values |
|---|---|---|---|
| FINGERPRINT REDUCTION | The number of bits of the hash that will be used for the fingerprint calculation | More collisions in matches due to less unique data in fingerprints; Less storage necessary | More precise matching due to storing more unique data; More storage necessary |
| PEAK SORT | Determines whether peaks are sorted temporally or not | False: Less fingerprints generated Can negatively affect performance | True: Better performance More fingerprints generated |
| DEFAULT OVERLAP RATIO | Determines to what extent do the FFT windows overlap | More precise matching More storage necessary | More precise matching More storage necessary |
| DEFAULT FAN VALUE | The number of close peaks used to make up the fingerprint | Low accuracy expected Less storage necessary | High accuracy expected More storage necessary |
| DEFAULT AMP MIN | The minimum amplitude in order to be considered a peak in the spectrogram | More fingerprints generated Captures more data and promotes accuracy | Less fingerprints generated Can negatively affect accuracy |
| PEAK NEIGHBORHOOD SIZE | The number of cells around an amplitude peak in order to be considered a spectral peak | More fingerprints generated Positively affect accuracy | Less fingerprints generated Can negatively affect accuracy |
| CONNECTIVITY MASK | Determines the morphology of the mask used when searching for maximum peaks | Value 1: diamond morphology, diagonal elements are not neighbours | Value 2: square mask, all elements are considered neighbours |
| DEFAULT FS | The default sampling rate | Lower quality of audio when sampling Worse performance | Higher quality of audio when sampling Better performance |
| DEFAULT WINDOW SIZE | The size of the FFT window | Lower granularity of frequencies Can negatively affect accuracy | Higher granularity of frequencies Positively affect accuracy |
| MIN/MAX HASH TIME DELTA | Determine how close or far can fingerprints be to be paired time-wise | Small range may reduce number of matches found May hinder performance of DEFAULT FAN VALUE | Higher range will likely increase the number of matches, bu itt does not indicate better performance |

Table 3: Configurable parameters in Dejavu

| Category | Subcategory | Code |
|---|---|---|
| Speech | Female Talking | SFT |
| | Male talking | SMT |
| | Female whispering | SFW |
| | Male whispering | SMW |
| | Female shouting | SFS |
| | Male shouting | SMS |
| | Cheering | SCH |
| Ambient | Street noise | AS |
| | Dining noise | AD |
| Nature | Rain | NR |
| | Running water | NW |
| | Thunder | NR |
| | Wind (Air) | NA |
| Terrain | Gravel | TG |
| | Wood creaking | TW |

Table 4: Noise categories and their acronyms