An AI Piano Tutor

Supervisor:
Dr. C. Wu
Department of Computer Science
The University of Hong Kong

Team Members:
Lin Yuyang (3035446899)
Ma Zhiying (3035334709)
Wang Ximing (3035448158)

2021.1.24
Abstract

AI is more and more widely used in the education industry. Many people are learning piano now and the need for free and professional practicing tutor increases largely. Some AI piano tutor apps appear and become popular. In this paper, some problems of these existing apps’ real-time strategy are presented like distracting and high error tolerance in rhythm. Our app uses a non-real-time strategy of doing overall comparison between a user’s piano playing audio and that of a teacher’s, and in the end returning a marked report on note accuracy and rhythm accuracy. Also, the system structure of our app is reported with Flutter in frontend and Python in backend. The music processing library, Essentia is chosen to be used in our music processing algorithm because it is cross-platform, robust and well-established. Madmom may also be used as a supplement for note-by-note detection. The current progress is in the algorithm implementation phase where general research on market and algorithms has been completed. Three kinds of algorithms in Essentia have been tried out, including music similarity, rhythm detection and pitch detection. The results are presented in the report, illustrating satisfying performance. However, difficulties still exist like how to implement one-by-one note detection and how to transform the value results into user-friendly reports. This report also puts forward some possible solutions such as using beat detection to split up subparts and combine music similarity algorithms, using Madmom as a supplement, etc. Testing out which way is the best to solve the problems as well as testing out how to generate our expected user-friendly report are the focuses in the next step of our project.
# Table of Contents

1. Introduction 1
   1.1 Background 1
      1.1.1 Education in AI 1
      1.1.2 Needs for Free Piano Tutors 1
      1.1.3 Problems embedded in existing solutions 2
         1.1.3.1 Dependence on Hardware 2
         1.1.3.2 Problematic Real-time Strategy 2
            1.1.3.2.1 High Error Tolerance in Rhythm 2
            1.1.3.2.2 Distraction from Playing 3
   1.2 Project Objectives 3
   1.3 Expected Deliverables 4

2. Methodology 4
   2.1 The Client-Server System Structure 4
   2.2 The Music Processing Algorithm 5
      2.2.1 Essentia as the Main Library In Use 5
      2.2.2 Supplementing Essentia with Madmom 6

3. Project Status 6
   3.1 Overview of the Development Cycle and Project Schedule 6
   3.2 The Algorithm Implementation Phase 7
      3.2.1 Music Similarity 7
         3.2.1.1 Technical Details 8
         3.2.1.2 Testing with a Real-Life Example 9
         3.2.1.3 General Comments 9
      3.2.2 Rhythm Detection 9
         3.2.2.1 Beat Detection 10
         3.2.2.2 Onset Detection 10
      3.2.3 Pitch Detection 11
   3.3 Project Difficulties 11
      3.3.1 One-by-one Note Detection 12
      3.3.2 User Friendly Report 12
   3.4 Possible Mitigation 12

4. Conclusion 13

5. References 14
List of Figures

Figure 1: *Leyi app user interface* 3
Figure 2: *An illustration of the client-server system structure* 4
Figure 3: *A diagram showing the HPCP features* 8
Figure 4: *Audio waveform of a sample music piece and its estimated beat positions* 10
Figure 5: *Audio waveform of a sample music piece and its estimated onset positions* 11
Figure 6: *A diagram showing the pitch contour* 11

List of Tables

Table 1  *An Overview of the Project Schedule* 7

List of Abbreviations

API  Application Programming Interface
BPM  Beats Per Minute
CNN  Convolutional Neural Network
1. Introduction

AI Piano Tutor is an application of the education industry in AI technology. The global trend and the market indicate that this project is well worth implementing. There are already some existing similar apps, however, every of them has some drawbacks. Our project has our own consideration and innovation and tries to accomplish a relatively perfect app finally.

1.1 Background

There are two situations that will be mentioned below. One is the situation of the education industry in artificial intelligence, another is the piano tutors problem. The situations show that our project is workable and has promising market potential.

1.1.1 Education in AI

Nowadays the education industry is becoming more and more popular in the AI-related search area, and there are substantial applications using AI technology to assist or even replace human beings to teach users. According to a report from Prescient & Strategic Intelligence Private Limited, the application of artificial intelligence (AI) in the education sector is rising. In 2019, the global AI in education market reached $1.1 billion and in 2030, it is predicted to generate $25.7 billion, advancing at a 32.9% CAGR during the forecast period (2020–2030)[1].

Therefore, the project that we focus on, making an e-learning platform for piano learners is workable and following the global trend.

1.1.2 Needs for Free Piano Tutors

Piano tutor is an application among AI technology applications. Learning piano is already known as a technique that needs quite a lot of time to repeatedly practice and real-time feedback. The learning process always needs supervisors. For beginners, they need the tutor to tell them which key on the keyboard to play according to the pages. As for median learners, they might not notice their performance while playing and it is really helpful if some audiences can give feedback immediately. Therefore, a one-to-one tutor for piano learners is very helpful and even necessary.

AI Piano tutor is more advanced than human beings tutor because of three reasons. First, obviously, AI piano tutor is much cheaper than private tutorials. Second, the practice time would be more flexible and fully depends on the learners. Last but not least, AI piano tutor is more accurate than human begins and it will not be less focused after a long time at work.

Due to the benefits and market potential, there are already several piano tutor apps emerging and different apps have different target users and use different methods to implement the tutor task.
1.1.3 Problems embedded in existing solutions

Several popular AI piano tutor apps are selected for analysis. Each of them has its advantages and disadvantages. Among those apps, three apps are selected, namely One Pianist, Leyi and SimplyPiano, which are similar popular piano tutor apps. The three apps all focus a lot on teaching beginners or children, so they provide many detailed tutorials of good quality and make the learning process like a game. However, by trying the apps or looking at user comments, it is found out though they achieve the tutoring functionality to some extent, there are still some problems embedded, namely dependence on hardware, problematic and distracting real-time strategy with high error tolerance in rhythm. Details are as follows.

1.1.3.1 Dependence on Hardware

One Pianist is a special app because it has to work with its matching The ONE smart piano with some hardware installed. When a note is to be played, the corresponding key lights up to show the learner where the next note is. And when the key is pressed down, the hardware reports the action back to the app. So the app mainly depends on hardware support to achieve great performance in judging note accuracy. However, the drawback lies in the fact that it depends on specific hardware and cannot adapt to different pianos, which increases the expense of users to buy a piano for using the app, thus decreases the possibility of the widespread use of the app.

1.1.3.2 Problematic Real-time Strategy

Though SimplyPiano and Leyi work well with different pianos, they apply a strategy of real-time judgement. SimplyPiano is the most popular app in Google Play store, and it uses a strategy of listening to the user's piano playing and giving real-time judgement on the note played. After the song is over, an overall report is given on note accuracy and timing accuracy. Similar design is applied for Leyi, and Leyi does more by annotating the notes that are wrongly played and the parts that need improvement directly on the music sheet, which gives clearer instruction on where users need to practice more. However, the problem of these two applications lies in the real-time strategy, mainly because it is distracting and the error tolerance in rhythm is too high.

1.1.3.2.1 High Error Tolerance in Rhythm

As shown in figure 1, this is the Leyi app user interface example. When the user is playing the music piece, a highlighting bar as shown in the picture moves on the music sheet to show which note is to be played and examined. If the note is played correctly and timely, it is marked green and otherwise red.

In fact, this real-time strategy only tests one note at a specific time, so if the user plays the target note for several times around that time, which is wrong in rhythm, the app will still judge it to be correct. And also, the design of testing one note at a time tolerates errors in rhythms because it
does not care about how long a certain note is played. However, these are definitely harmful and misleading from the perspective of the user's learning process.

Figure 1: Leyi app user interface, a highlight bar moves to show the next note to be played and examined. If the note is played correctly and timely, it is marked green and otherwise red.

1.1.3.2.2 Distraction from Playing

It is very distracting for users to pay close attention to whether the last note has been judged to be correct while playing music pieces, especially for non-beginners. In addition, the real-time strategy requires high network speed. Otherwise, it is very likely that players are interrupted when playing if the last notes played are not judged because of network delay. Then they have to play the notes again until the real-time judgement is resumed.

1.2 Project Objectives

In view of the problems mentioned above, what this project does is to build an AI piano tutor app that is different from those existing ones. Instead of checking each note on by one in real time, the app uses a non-real-time strategy, which means that the app automatically generates an overall report after the user plays the whole piece. This enables users to focus more on the playing of the musical piece itself without the need to check how the app is reporting for each note. In addition, the design reduces the requirement of real-time network speed and reduces the chance of misjudging and interruption of user’s playing because of network delay.

A music processing system would be set up. The report is generated on a cloud platform through the system by comparing the master’s/teacher’s playing and the user’s playing of the selected musical piece. To better improve the user’s performance, the judgement not only focuses on the accuracy of notes, but also focuses on the rhythm. The report is presented by giving marks on each aspect as well as circling notes that are wrongly played and highlighting paragraphs that need improvement on the music score. Moreover, the aim is to make the mobile interface user-friendly and achieve high accuracy on the assessment of the user's play.
1.3 Expected Deliverables

The final deliverable will be a user-friendly AI piano tutor mobile app using a non-real-time strategy, with a music processing system set on a cloud platform. After listening to user’s complete piece of playing, the app can return an overall score report to the user on note accuracy and rhythm accuracy, and also make annotations on music sheets to show which notes are played wrong and which paragraphs have bad rhythm.

2. Methodology

As a mobile application, the system structure is essential to our project. Our system is designed to be composed of two parts, respectively the client side and the server side. In addition to the overall structure, the algorithm design also plays an important role, because music processing lies in the core of the functionality of our app. Therefore, the following sections introduce the methodology of our projects in these two aspects.

2.1 The Client-Server System Structure

The AI Piano Tutor of our project consists of a client-side app and a server-side app. Figure 2 below illustrates the overall structure of the system.

![Client-Server System Structure Diagram]

**Figure 2**: An illustration of the client-server system structure

As illustrated above, the client-side app implements the user interface, whereas the server side handles the algorithms for music processing. The client-side calls the APIs provided by the server-side so that the calculation could be carried out at the backend and the results would be returned to the frontend.

The client-side uses Flutter, a cross-platform toolkit that can be used to build native apps both for Android and iOS from a single codebase. Traditionally, developers use Android and Swift to implement two sets of codes for the two major mobile platforms, namely Android and iOS, but Flutter relieves developers from this redundant work by allowing for one set of codes for both platforms. In addition, Dart, the programming language it uses, has been proven to be fast on all the platforms it supports. Thus, in order to maximize efficiency, Flutter is chosen for our project.
The server-side, on the other hand, uses Django as its web framework. This framework uses Python as its programming language, and Python is largely preferred in our project because of its preeminent signal processing power. The existing audio processing packages such as librosa and PyAudio are all implemented for usage in Python only. Moreover, the majority of open-source libraries for music analysis, such as Essentia [3] or madmom [5], are either directly implemented in Python or available for usage in it. This project will rely completely on existing libraries to realize the music processing functions, and since the majority of the libraries are available in Python, it is reasonable to choose it as our backend language. In addition to the language compatibility, Django is also rapid in development and pragmatic in its design, which all accounts for our commitment in it.

The server is deployed to a cloud service platform. At this moment, Azure is elected to be the platform for our usage during the developmental cycle, because the lowest price tier in Azure is actually free of charge. Although this price tier only provides limited storage space and calculation power, it is sufficient for testing purposes. Therefore, with financial consideration, choosing Azure is a rather practical decision.

2.2 The Music Processing Algorithm

In order to generate a report based on the user’s performance, the app needs algorithms to process and analyze the audio inputs. In particular, we need to detect the note sequences, the beat and the onset timing of each note, and the overall similarity in the audio inputs. In order to achieve these goals, existing open-source libraries are to be imported into our backend application, and the related functions in these libraries will serve to process the audio and generate the results. In this section, the selected music processing libraries, namely Essentia [3] and madmom[5] will be introduced in details.

In general, Essentia is cross-platform, robust and well-established, but the algorithms it provides are mainly for high-level analysis.

2.2.1 Essentia as the Main Library In Use

Essentia is an open-source library that is implemented in C++, but it also comes with Python and Javascript bindings which makes it available in multiple settings including our Python-implemented backend application. As a collection of algorithms for audio analysis and music information retrieval, this library stands out for several reasons. First, Essentia is designed to be cross-platform. The various systems that it covers include iOS, Android, and web applications, and the languages it supports include Python and Javascript. With this outstanding compatibility, it is reasonable to assume that it should be adaptable to our project. What’s more, the library is robust in terms of its computational speed and memory usage. Implemented in C++, it is natural that Essentia computes faster than others that are implemented in high-level
programming languages such as Python. The memory usage is also optimized in a way that impose the least burden on the computer. Last but not least, Essentia is well-established. The abundance of documentations, samples, and tutorials altogether provide a good way for beginners to learn how to use it. More importantly, a lot of companies and projects have already used Essentia in their development, which showcased Essentia’s potential to run stably and effectively in real-life applications [3].

Essentia provides a wide range of algorithms including beat tracking, onset detection, and cover song identification. In the algorithm implementation stage so far, the algorithms in Essentia that appear relevant to our project have been tested, and the results will be introduced in more details in later sections. In general, the algorithms in Essentia focus more on the high-level analysis, which should be helpful in rhythm evaluation and overall music similarity, but it is still unclear whether it is possible to do note-by-note detection using Essentia.

2.2.2 Supplementing Essentia with Madmom

Madmom is another open-source library written in Python and focuses mainly on musical information retrieval. Although it is not as powerful and well-established as Essentia, it provides functions that return lower-level details retrieved from audio inputs such as an array of notes detected [5]. This may be able to fulfil our needs to do note-by-note detection, and the testing of it will therefore be our focus in the next steps.

3. Project Status

The development cycle of our project is divided into several stages, from research, algorithm implementation, system development to testing and refinement. Currently, we are at the algorithm implementation stage, where the music processing algorithms are investigated and tested. The following sections provide an overview of the development cycle and proposed schedule. Our current progress is also introduced in detail.

3.1 Overview of the Development Cycle and Project Schedule

Our project mainly consists of 4 phases, with a respective focus in research, algorithm implementation, client-side development, and testing and refinement. The first phase, namely the research phase, aims to gain practical insights into existing algorithms related to music processing. Then, with sufficient knowledge in related works, our project moves into the second phase, the algorithm implementation, in which some elected open-source libraries will be examined and subsequently incorporated into our project. The end product of this phase should ideally be functional in identifying notes, evaluating rhythm correctness, and generating an overall performance report from audio inputs. After that, the user interface and the backend server will be designed and developed. Since the aesthetics is not of our major considerations, our project only aims to create a functional interface. Last but not least, representatives of
potential users will be invited to test the app. Based on their feedback, adjustment will be proposed and, when time permitting, implemented.

The project will be completed in April 2021 and exhibited in early May. Based on these deadlines, a general schedule is proposed (see Table 1).

<table>
<thead>
<tr>
<th>Steps</th>
<th>Dates</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Plan</td>
<td>Oct 4, 2020</td>
</tr>
<tr>
<td>Phase 1: Research</td>
<td>Oct 5 - Nov 30, 2020</td>
</tr>
<tr>
<td>Phase 2: Algorithm Implementation</td>
<td>Nov 30, 2020 – Feb 28, 2021</td>
</tr>
<tr>
<td>Phase 3: System Development</td>
<td>Mar 1, 2021 - March 31, 2021</td>
</tr>
<tr>
<td>Phase 4: General Testing and Adjustment</td>
<td>Apr 1, 2021 - Apr 17, 2021</td>
</tr>
<tr>
<td>Finalized Tested Implementation</td>
<td>Apr 18, 2021</td>
</tr>
<tr>
<td>Project Exhibition</td>
<td>May 4, 2021</td>
</tr>
</tbody>
</table>

Table 1: An Overview of the Project Schedule

The research phase has ended before December. From December to the end of February, the focus is on the algorithm implementation. Then, the project will proceed to phase 3, where the user interface of the client side will be developed. At the beginning of April, the last phase of general testing will be carried out and adjustment will be made accordingly. After that, our project should ideally be ready for the final presentation.

3.2 The Algorithm Implementation Phase

So far, we have selected three different algorithms in Essentia that may be related to our project, namely music similarity, rhythm detection and pitch detection algorithms. Each of these are tested separately. The following sections will illustrate the results of the testings.

3.2.1 Music Similarity

This algorithm is designed to take two audio inputs in MP3 format, and return a similarity distance in numerics. The bigger the similarity distance, the less similar the two pieces are. In our project, the user may either upload a customized music exemplar, or they may choose from
our built-in database as the exemplar. Then, they may play the piano and upload the recording of their practice. After that, the back-end will take the exemplar and the practice recording, and the music similarity algorithm will calculate the similarity distance before returning the value back to the front end.

In the following sections, the technical details of this music similarity algorithm, including its judgment criteria and the calculation steps, will first be explained. Then, it will proceed to illustrate the results of the testing.

3.2.1.1 Technical Details

The major criteria that this algorithm uses to decide the similarity is the tonal sequence. Since this algorithm is originally designed for the purpose of identifying cover songs, this factor itself is already sufficient.

There are mainly three steps to compute the similarity. The first step is pre-processing, where the HPCP features of the two audio inputs are extracted separately by time-frequency analysis. HPCP stands for harmonic pitch class profile. It is a collection of vectors that measure the relative intensity of the 12 pitch classes in music. From the HPCP features, it is possible to estimate the tonal sequence of the audio input. Figure 3 shows an example of the HPCP features plotted.

![Figure 3: A diagram showing the HPCP features](image)

As shown above, the x-axis of the diagram represents the time and y-axis stands for the 12 pitch classes. The segments in light yellow means that the corresponding pitch class (y-axis) has high intensity at a particular time (x-axis) in the audio. The HPCP features are robust to noise, and are independent of factors such as timbre, instrumentation, loudness and dynamics [9]. It is therefore
reliable in its results, but it also means that if we want to evaluate the loudness or dynamics as part of the performance report in our project, these features will not be helpful.

After the HPCP features of the two audio inputs are both computed, the binary chroma cross similarity matrix is then computed based on the pair of HPCP features from the previous step. Finally, based on that matrix, the similarity distance is computed.

3.2.1.2 Testing with a Real-Life Example

In order to see how this algorithm performs with real-life piano inputs, we run it with two different pairs. We downloaded two different versions of For Elise (referred to as Sample 1 and Sample 2), both played correctly and completely, and we made a recording (referred to as Sample 3) of one of the teammates, who had no previous knowledge of piano, playing this exact same composition. Based on the functionality of this algorithm, we expect to yield a small similarity distance between Sample 1 and Sample 2, and a much larger distance between Sample 1 and Sample 3. The results were approximately 0.02 for the first pair, and 0.40 for the second pair, which met with our expectation. As a result, we may preliminarily assume that this algorithm works well to tell the general similarity in terms of the tonal sequence between two audio inputs.

3.2.1.3 General Comments

In summary, this algorithm may be useful in our project by producing an overall similarity score against an exemplar. It is flexible in the sense that we may allow users to upload their own exemplars as they want, and thus eliminating the need for a large built-in database of music exemplars. However, it only takes into consideration the tonal sequence, thus neglecting all the other factors such as rhythm and dynamics. Besides, it is unable to identify individual errors, and only a general score is returned. Therefore, this algorithm cannot replace note-by-note detection. As a matter of fact, it can only serve as a complement to the one-by-one note detection, since our project focuses more on informing the users of their specific errors. In this sense, we may conclude that this algorithm does not meet the core requirements of our project.

3.2.2 Rhythm Detection

To test the rhythm accuracy of the user, there are mainly two goals. The first goal is to get the overall tempo of a music piece, and to detect the beat positions in order to calculate the beat steadiness, which is important for examining the overall rhythm. The second goal is to examine the onset accuracy of each note, so that we know whether an individual note is played on its expected beat or not. The first goal can be achieved by doing beat detection using RhythmExtractor2013 in Essentia, and the second goal can be achieved by using the onset detection algorithm in Essentia library. Details are as follows.
3.2.2.1 Beat Detection

Essentia library provides a very powerful rhythm extractor. This extractor receives an input of a piece of audio and it can detect and output the beat positions, BPM and the beat estimation confidence, etc. BPM is short for Beats Per Minute, and it is a measurement of tempo, the speed of piano playing. The calculated BPM as well as the BPM histogram can be made use of to calculate the overall beat steadiness as well as the tempo difference between user’s BPM and target BPM. The beat positions are detected accurately in our try out, and figure 4 shows the audio waveform of our sample music piece and its estimated beat positions. The red lines represent the estimated beat positions, and the blue one is the waveform standing for magnitude with respect to frame. The sample music piece is well-played, and it can be seen that the rhythm is rather steady, with relatively even beat distribution. Also, with audio visualization by marking detected beat positions with beep sound, it can be heard that the beat positions are detected accurately, showing that the rhythm extractor performs well. With these beat positions, it is possible for us to split up the bars.

![Audio waveform and the estimated beat positions](image)

**Figure 4:** Audio waveform of a sample music piece and its estimated beat positions

3.2.2.2 Onset Detection

The onset detection algorithm by Essentia can detect the onset position of each individual note. The algorithm has two phases. First, it calculates an onset detection function for an audio frame. Various onset detection functions are provided, and in our project, the complex onset detection function is used because it has better performance after our comparison. Second, it decides the onset locations in the audio based on the functions. In our try out, the algorithm performs well for our sample music piece. As shown in figure 5, red lines represent the detected onset positions and the blue one is the waveform. It can be seen that the red lines lie close to the peaks in the waveform which mean the actual onset positions. Also, the satisfying performance can be shown in the audio visualization with beep sound marked at estimated onset positions. The beep sounds coincide with the notes played, which is exactly what we want. With these onset positions as
well as the splitted bars, it is possible for us to decide the rhythm accuracy of each individual note.

![Audio waveform and the estimated onset positions (complex onset detection function)](image)

**Figure 5:** Audio waveform of a sample music piece and its estimated onset positions of notes

### 3.2.3 Pitch Detection

PredominatpitchMelodia in Essentia is an algorithm designed to extract pitch value, in Herz, of a melody.

The input is the audio file in MP3 format, and then the output is a series Herz values of the melody. The algorithm uses the output to plot a graph that is shown as pitch contour of the melody, like in figure 6. The x-axis is the time, and the y-axis is the pitch value, in Herz.

![estimated pitch [Hz]](image)

**Figure 6:** A diagram showing the pitch contour

### 3.3 Project Difficulties

Although we have already explored several useful algorithms in Essentia library, there are still some problems that we need to solve immediately. We need to find a method to detect one single note frequency and determine whether it’s correct or not and also we should think about how to generate our report in the app properly using the output we get.
3.3.1 One-by-one Note Detection

According to our current progress, in Part 3.2.3, we are able to get the pitch contour of the melody, however, we don't get the exact pitch value for one single note. It's highly possible that the algorithm, PredominatpitchMelodia has that value in progress, however, until now our superficial research, it is like black box for us, and we cannot get the intermediate output. Although it's possible to use the integrated pitch contour comparison to get the similarity, it would be useful if our app could tell the user exactly which note is wrong or not.

3.3.2 User Friendly Report

According to Part 3.2.1, we’ve implemented a method to generate a value to evaluate the similarity. However, the problem is how to change the value to a user friendly report, which is our final deliverable. We mentioned scores like 0.02 and 0.40 in Part 3.2.1.2, and obviously that the users cannot understand whether he or she plays well or not if we just show the value to them.

3.4 Possible Mitigation

According to the problems mentioned in Part 3.3, we've planned several solutions and will implement them in our next steps.

Instead of detecting one single note, we could cut the melody into subparts. According to Part 3.2.2.1, we’re able to detect the beat for melody and it’s possible to reasonably cut the melody into several parts depending on the beat. Therefore, we could show the overall similarity of the input melody and also more detailed similarity for subparts. It's also possible that we plot a diagram for users, which x-axis is the time and y-axis is the similarity value.

Another alternative solution is using another library, madmom. In madmom, there is a class called madmom.features.notes.CNNPianoNoteProcessor. The output of the algorithm is a series of 3-dimensional arrays, representing [seconds, pitches, duration][10]. This class can be used to detect the onsets and intermediate note features exploiting CNN. The features of notes in this library class are expressed by the note start time (seconds), a value to represent the pitch, and duration time. These parameters are sufficient to distinguish the different notes and we can possibly tell whether a single note is correct or not. Further research is needed for this library.

In terms of how to generate a user friendly report, it will need much more experiments. We should transfer the similarity scores generated by algorithms to adjectives or scores in particular range. We might use a perfect example to get the best similarity value generated computer and use a bad practice to get the lowest similarity. The transformation can be convincing only when we have an abundant amount of tests.
4. Conclusion

AI Piano Tutor is a promising application according to the background survey and we would like to utilize our own consideration and innovation to implement a better app than the existing similar products. Our finished product will give the user a report with good visualization and detailed analysis for the user’s performance.

The system architecture design is very common as many other apps. The client side framework will be Flutter and the server side will be put in the cloud. The music algorithm we import in this project is Essentia, which is very powerful and suits our features. It has an algorithm to get the similarity of two input melodies, an algorithm to get the beat, and another algorithm to plot the pitch contour.

The pitch contour is significant to visualize the melody in a non-hearing way, which could be understood by computer because it gets the pitch numerical value. However, we would further want to get a single note value and to get whether a signal note is correct or not. Since the algorithm is a kind of black box, we can use another library or cut the melody into subparts. Much more study about the Essentia library and also other libraries, like madmon is needed.

Furthermore, how to generate the final report for the user and what to show in the report is one of the most important problems in our next stage. We’ve now got some information about the input melody, but how to transfer it into the information that users can easily understand is a big problem. More experiments, data sets and surveys might be needed in our next stage.
5. References


