An Intelligent System to Assist Piano Composition and Chords Generation using AI and Machine Learning

Jiaxuan Zhang[1] and Yu Sun[2]

[1]Arcadia High School, 180 Campus Dr, Arcadia, CA 91006
[2]California State Polytechnic University, Pomona, CA, 91768

Abstract

As a musician and producer, I've always struggled with finding chords when I first started writing music [5]. It sometimes goes to the extent of me forgetting my melody because I take so long trying to figure out the chords. So I came up with an idea for this app, that will help amateur and beginner musicians save time and provide chord suggestions to them as a booster to start writing songs [6]. It features a recording or a midi input feature, then the app will carefully analyze the given melody and give a selection of the best chord progressions using intelligent AI. As an output, it is able to present it as guitar chords, piano chords, and ukulele chords, enabling more different musicians to use this app.

Keywords

Music, Chords, Melody, Chord Generation.

1. Introduction

There are many beginner musicians all around the globe, and many of them are aspiring songwriters and producers. With a proper tool, many musicians will have the benefit of saving time and getting a jump start on their passion. By using our app, the musician can simply just record or upload a file into the app, and it will automatically present a set of the best chord progression that matches their melody perfectly. After the app outputs the chord progressions, it will also provide a chord chart for three instruments, piano, guitar, and ukulele, enabling for the musicians to learn the chords along the way as well.

There isn't a tool that fits so perfectly into the needs of the musicians. There are apps that transcribe existing music into chord progressions, but there has not been an app that transcribes a brand new melody into a chord progression. The existing apps are very simple since an existing song already includes a chord progression within, so it is very easy to identify. But using a melody to determine its chords, then it is a way harder thing to achieve. Luckily, with knowledge in songwriting and coding, we are able to create a very intelligent app that will do just that, enabling musicians to easily create chord progression inspirations.

Our goal with Melodyfi is to create an algorithm that thoroughly examines the user's melody, taking the first note of each measure and matching it to a chord that perfectly fits the melody. Then the many chords will create a perfect chord progression that fits the melody and also stays within a good key. The app includes a feature that enables musicians to record their melody idea, then the algorithm takes in the melody to produce the final chord progression suggestions. Other features that are coming with future updates, are a bank of guitar, piano, and ukulele chords that lets the user learn any chords on the instrument of their choice.

In two application scenarios, we demonstrate how the above features are of use. In experiment 1, firstly, we show the usefulness of our approach by testing the accuracy of the melody and chord progression. Second, we will compare the chord output to a real human approach in finding the chords to the same exact set of melody. In experiment 2, we create a survey and send it to a group of the users to collect the review score.

The rest of the paper is organized as follows: Section 2 gives the details on the challenges that we met during the experiment and designing the sample; Section 3 focuses on the details of our solutions corresponding to the challenges that we mentioned in Section 2; Section 4 presents the relevant details about the experiment we did, following by presenting the related work in Section 5. Finally, Section 6 gives the conclusion remarks, as well as pointing out the future work of this project.

2. CHALLENGES

In order to build the tracking system, a few challenges have been identified as follows.

2.1. Choosing chords from a melody

Whether because they do not have sufficient knowledge in music theory or if they just don't know how to compose chords, beginner musicians all around the world suffer from unable to identify chords to songs to a melody they've come up with [7]. In order to quickly come up with chords, a musician must have sufficient knowledge in at least the basics of music theory, which a lot of beginner musicians lack.

2.2. Forgetting melodies

As a Beginner musician, they forget their catchy melodies easily while in the process of songwriting due to the lack of experience. A lot of time goes into figuring out the technical side of the song instead of them focusing on the important creating part, so a lot of the time goes wasted on figuring out chord progressions or other technical music processes.

2.3. Wrong chord usage

Beginner musicians tend to create chord progressions that match the melody's notes but it is not correct by musical definition due to the lack of experience. They would write chords that "fits" technically, but does not necessarily sound good and/or is not musically correct, whether if it doesn't fit the key or many other reasons [15].

3. Solution

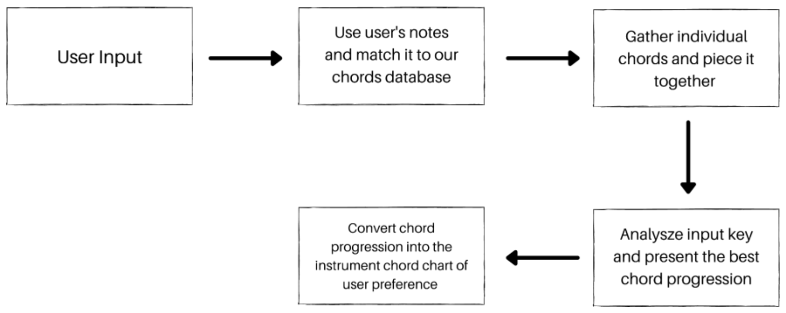

User Input
Use user's notes and match it to our chords database
Gather individual chords and piece it together
Analysze input key and present the best chord progression
Convert chord progression into the instrument chord chart of user preference

Figure 1. Overview of the system

Figure 1 shows a high-level overview of the system. The system is implemented as a web service application as the backend, and a frontend mobile UI for users to interact [8]. All the requests are coming from the mobile component, and those requests will be sent to the backend server for generating the recommended chords. The backend server runs in the cloud and it uses algorithms to process the input melody file and generate the recommended chords based on that melody. The result will be sent back to the mobile devices and the mobile screen will render the results in a user-friendly way [1]. Figure 2 shows the basic mobile design and the user experience of the app. More details about each functionality and the implementation will be discussed in sections 3.1.

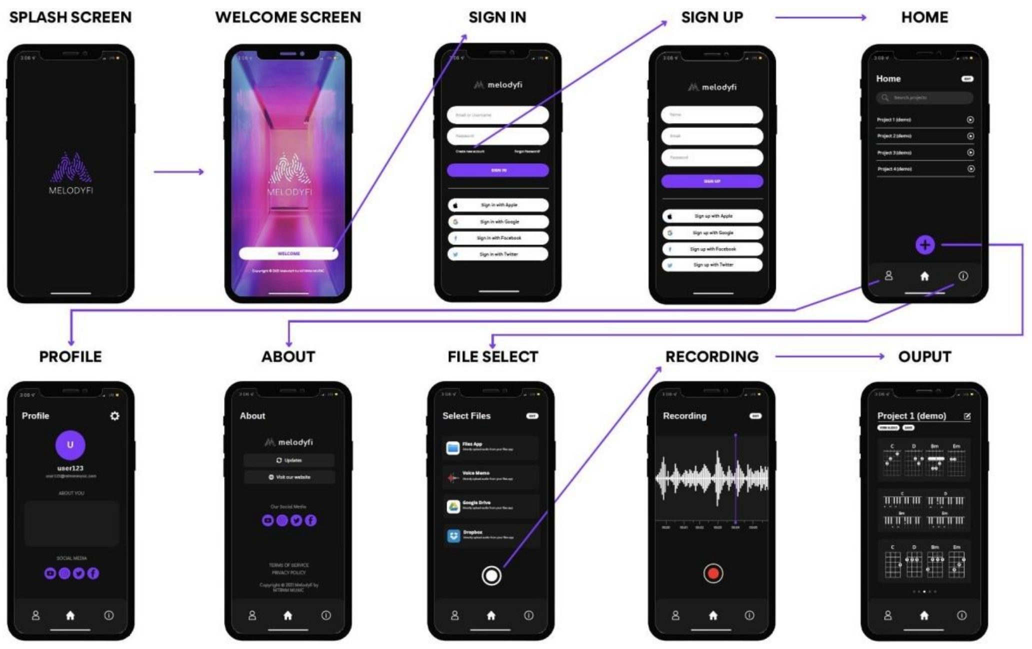
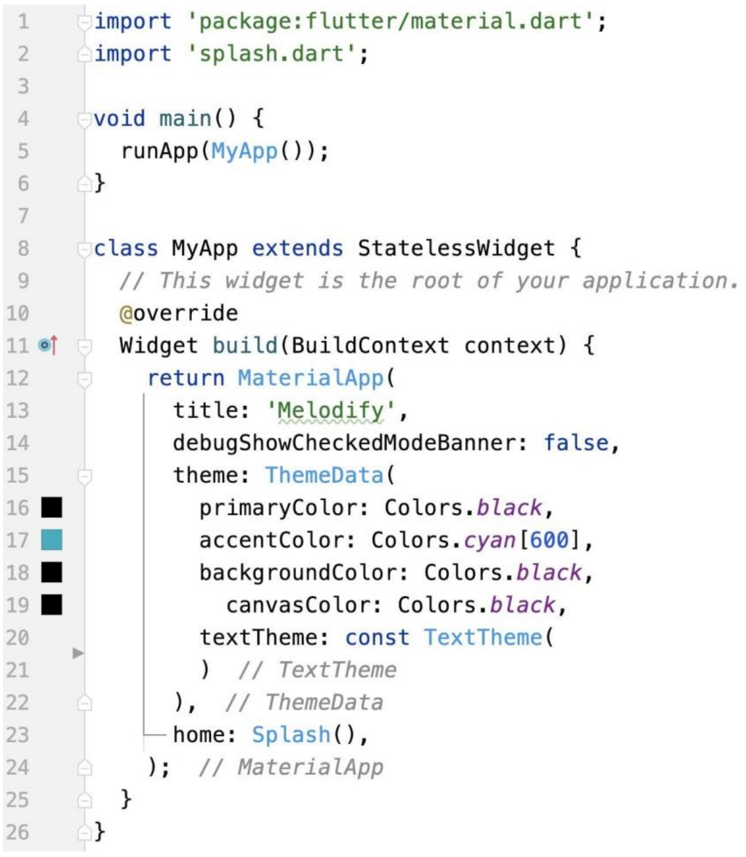


Figure 2. The UI design of the application

3.1. Mobile Development

We have decided to use Flutter to develop this mobile application. Flutter is a popular and advanced mobile development framework. It supports both Android and iOS systems, so that you only need to write the code once and the app will be generated for both platforms. Flutter uses the language Dart which is very similar to Java. Dart has a lot of new features that simplify the logic and algorithms, including a strong support from the 3rd party libraries.



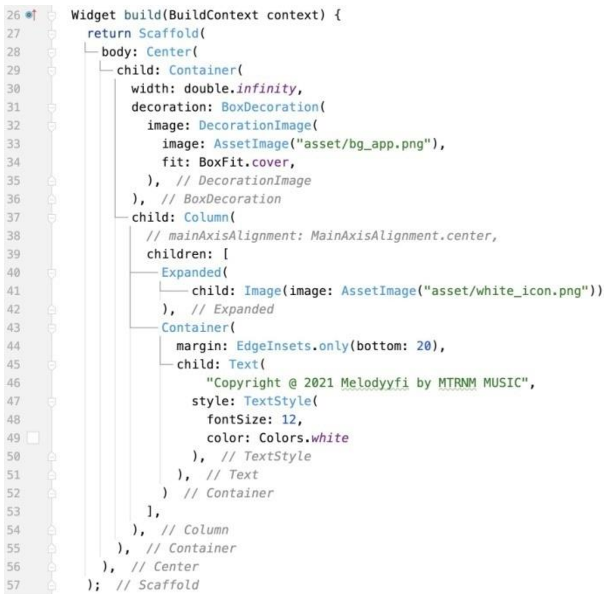
```dart
import 'package:flutter/material.dart';
import 'splash.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Melodify',
      debugShowCheckedModeBanner: false,
      theme: ThemeData(
        primaryColor: Colors.black,
        accentColor: Colors.cyan[600],
        backgroundColor: Colors.black,
          canvasColor: Colors.black,
        textTheme: const TextTheme(
        )  // TextTheme
      ),  // ThemeData
      home: Splash(),
    );  // MaterialApp
  }
}
```

Figure 3. The code excerpt of the app starter main method

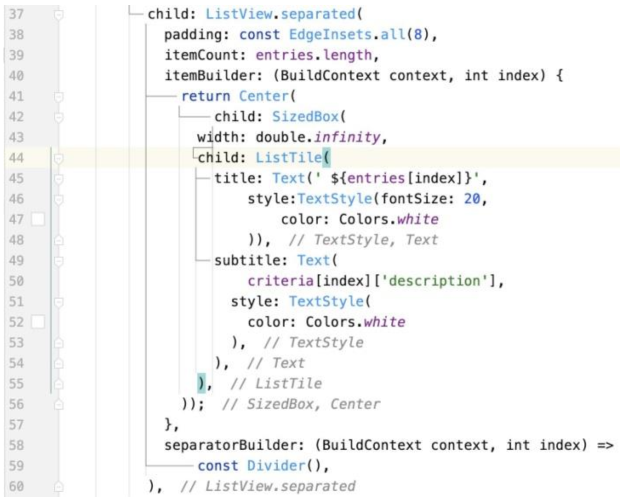
The entrance to the app is specified in the MyApp class. Our app uses material design. As we can see from the code above, we specify the color theme used for the app. The color theme includes the primary color, accent color, background color, canvas color and text theme. Flutter uses a lot of Json style specifications for different kinds of configurations [9]. This code also specifies the very first screen to start with when people run the app.



```
Widget build(BuildContext context) {
  return Scaffold(
    body: Center(
      child: Container(
        width: double.infinity,
        decoration: BoxDecoration(
          image: DecorationImage(
            image: AssetImage("asset/bg_app.png"),
            fit: BoxFit.cover,
          ), // DecorationImage
        ), // BoxDecoration
        child: Column(
          // mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Expanded(
              child: Image(image: AssetImage("asset/white_icon.png"))
            ), // Expanded
            Container(
              margin: EdgeInsets.only(bottom: 20),
              child: Text(
                  "Copyright @ 2021 Melodyyfi by MTRNM MUSIC",
                style: TextStyle(
                  fontSize: 12,
                  color: Colors.white
                ), // TextStyle
              ), // Text
            ) // Container
          ],
        ), // Column
      ), // Container
    ), // Center
  ); // Scaffold
```

Figure 4. The code excerpt of the app splash screen

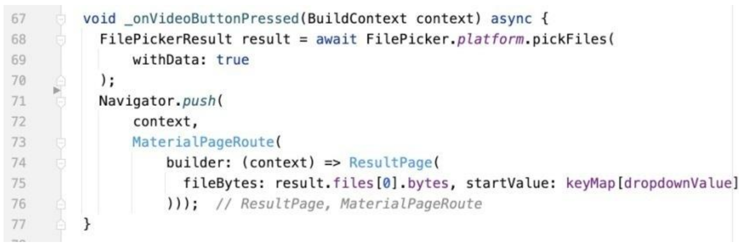
The code above shows how we implemented the splash screen [10]. All of the UI layout and components are specified in the build method. In this case, a center layout is used that includes a container with the image inside. A line of text is displayed at the bottom of the screen for the copyright information. All the images used in the app must be specified in the yaml configuration file first before being referred to in the UI code such as AssetImage [11].



```
child: ListView.separated(
  padding: const EdgeInsets.all(8),
  itemCount: entries.length,
  itemBuilder: (BuildContext context, int index) {
    return Center(
        child: SizedBox(
      width: double.infinity,
      child: ListTile(
        title: Text(' ${entries[index]}',
            style:TextStyle(fontSize: 20,
                color: Colors.white
            )), // TextStyle, Text
        subtitle: Text(
            criteria[index]['description'],
          style: TextStyle(
            color: Colors.white
          ), // TextStyle
        ), // Text
      ), // ListTile
    )); // SizedBox, Center
  },
  separatorBuilder: (BuildContext context, int index) =>
      const Divider(),
), // ListView.separated
```

Figure 5. The code excerpt of the about screen

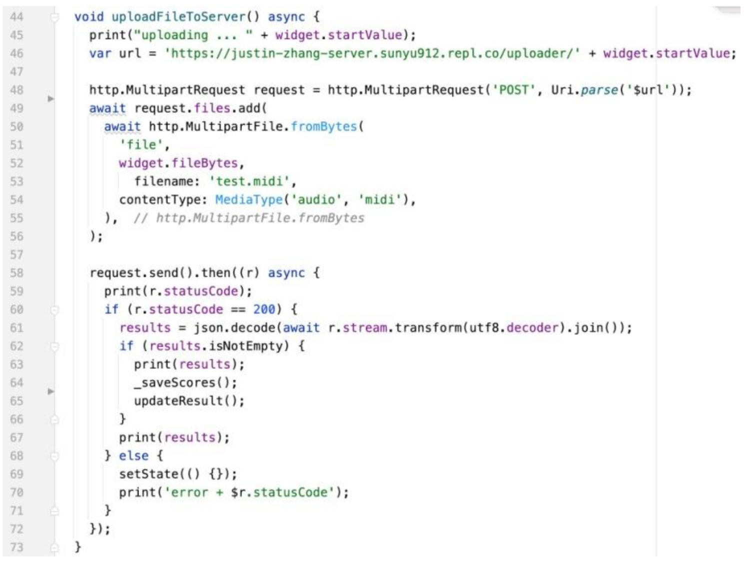
The app uses a bottom navigation bar to navigate between different screens. The first item in the bottom navigation bar is called "learn". This table shows all the information about the app including how to use the app, the purpose of the app, and the user tips. The screen contains most of the text information. In order to make the screen more extensible and customizable, we have stored all the text information in the list of strings. As we can see from the code above, the UI of this screen uses a ListView to display the list of strings. All the text information is not hard-coded, so that whenever we need to make a change, the only change to make is the list of strings.



```dart
void _onVideoButtonPressed(BuildContext context) async {
  FilePickerResult result = await FilePicker.platform.pickFiles(
      withData: true
  );
  Navigator.push(
      context,
      MaterialPageRoute(
          builder: (context) => ResultPage(
            fileBytes: result.files[0].bytes, startValue: keyMap[dropdownValue]
          ))); // ResultPage, MaterialPageRoute
}
```

Figure 6. The code excerpt of the file upload library

The middle tab is the major functionality of the app. Users choose the base key and upload a file of the melody, so that all these inputs will be sent to the backend server for processing. This UI is implemented with a couple of UI input widgets including the drop-down menu, and the file upload button. File upload on a mobile device requires interaction with the operating system [12]. In our app, we have applied a 3rd party library to facilitate the file uploading process. This library automatically interacts with both Android and iOS file systems so that whenever a user presses the button, it triggers the external file picker used by the operating system which enables users to easily pick the files from different folders and different categories. The code shown above shows how the file picker library works. It is a very simple and straightforward API to call that will return all the file information being selected. The selected default information will be sent to the next crane where the results green will process all the requests and render the result.



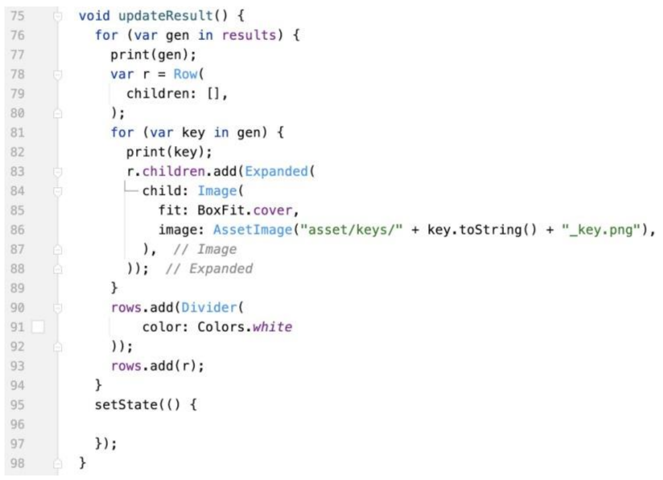
```dart
void uploadFileToServer() async {
  print("uploading ... " + widget.startValue);
  var url = 'https://justin-zhang-server.sunyu912.repl.co/uploader/' + widget.startValue;

  http.MultipartRequest request = http.MultipartRequest('POST', Uri.parse('$url'));
  await request.files.add(
    await http.MultipartFile.fromBytes(
      'file',
      widget.fileBytes,
        filename: 'test.midi',
      contentType: MediaType('audio', 'midi'),
    ),  // http.MultipartFile.fromBytes
  );

  request.send().then((r) async {
    print(r.statusCode);
    if (r.statusCode == 200) {
      results = json.decode(await r.stream.transform(utf8.decoder).join());
      if (results.isNotEmpty) {
        print(results);
        _saveScores();
        updateResult();
      }
      print(results);
    } else {
      setState(() {});
      print('error + $r.statusCode');
    }
  });
}
```

Figure 7. The code excerpt of sending the midi file to the backend for processing

When the next screen receives the selected file, it will send all the chosen file together with the chosen base key to the back-end server. We have applied the HTTP library in Dart to send all the requests. The back-end server has been hosted in replit.com with the HTTPS configured. Sending the HTTP from Dart in Flutter is very straightforward by specifying the base URL, the input parameters, and the file input. The library takes care of converting the chosen file on the mobile device to an array of bytes which will be sent together in the HTTP request. A callback is used here to check the response. If the status code is 200, and if the content is returned correctly, we will save the results on the local device with the local database, followed by rendering all the generated results on the screen.

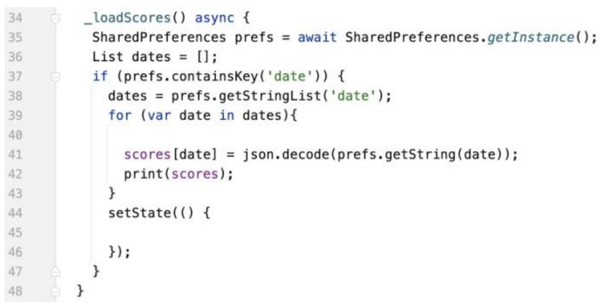


```
void updateResult() {
  for (var gen in results) {
    print(gen);
    var r = Row(
      children: [],
    );
    for (var key in gen) {
      print(key);
      r.children.add(Expanded(
        child: Image(
          fit: BoxFit.cover,
          image: AssetImage("asset/keys/" + key.toString() + "_key.png"),
        ), // Image
      )); // Expanded
    }
    rows.add(Divider(
        color: Colors.white
    ));
    rows.add(r);
  }
  setState(() {

  });
}
```

Figure 8. The code excerpt of converting the generated chord result to images

The updateResult method handles rendering the final result on the screen. The generated chord is saved in the list as integers. The updateResult method reads the list and chooses the corresponding images to use in the display. In order to improve the performance of the rendering, we have stored all the images on the device since there are a certain number of keys being used in the generation process. Loading these images directly on a device is a lot faster compared with loading those from servers. We simply use a file name schema to automatically map the key number to the image.



```
_loadScores() async {
  SharedPreferences prefs = await SharedPreferences.getInstance();
  List dates = [];
  if (prefs.containsKey('date')) {
    dates = prefs.getStringList('date');
    for (var date in dates){

      scores[date] = json.decode(prefs.getString(date));
      print(scores);
    }
    setState(() {

    });
  }
}
```

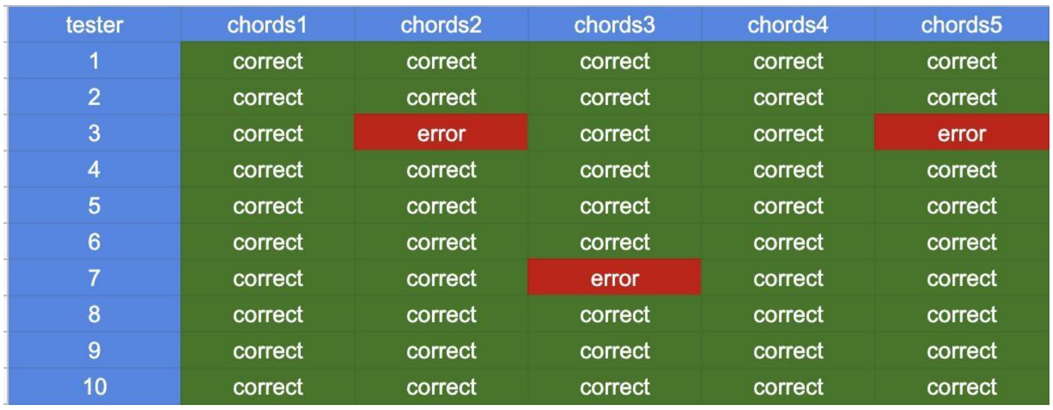
Figure 9. The code excerpt of loading the persisted generated results

In order for the users to view the past generated chords, we have implemented a local database to store and persist all the generated results. Every time a generation request was handled successfully, the result will be saved as a Json string in the SharedPreferences. SharedPreferences is a simple mechanism supported by most mobile operating systems to store the information as key-value pairs [13]. Instead of setting a professional relational or non-relational database, the SharedPreferences provides a very simple and rapid way to handle the local information storage efficiently.

4. Experiment

4.1. Experiment 1

For experiment 1, we check for the AI's accuracy on the chord progression output. I asked 10 participants to try inputting 5 different files with different keys into the apps. The results they received are all very accurate since the AI targets the specific notes and produces the chords based on the notes. The data table shows below:



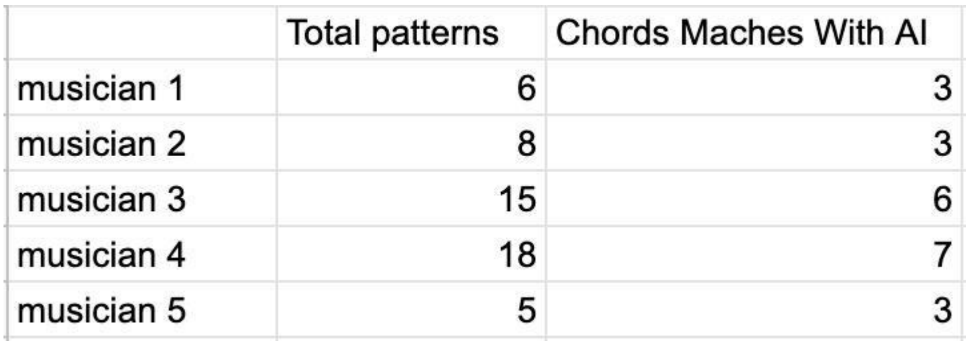
tester	chords1	chords2	chords3	chords4	chords5
1	correct	correct	correct	correct	correct
2	correct	correct	correct	correct	correct
3	correct	error	correct	correct	error
4	correct	correct	correct	correct	correct
5	correct	correct	correct	correct	correct
6	correct	correct	correct	correct	correct
7	correct	correct	error	correct	correct
8	correct	correct	correct	correct	correct
9	correct	correct	correct	correct	correct
10	correct	correct	correct	correct	correct

Figure 10. Table of chords

In the table we can see that the AI's accuracy on the chord progression output is 94%. Based on the test cases and performance we can consider that the AI has a high accuracy rate.

4.2. Experiment 2

For experiment 2, we compare the AI's output to humanistic output ß if the musician produces the chords with his own thinking. We ask 5 musician produce to do the test and the result turns like this:



	Total patterns	Chords Maches With AI
musician 1	6	3
musician 2	8	3
musician 3	15	6
musician 4	18	7
musician 5	5	3

Figure 11. Table of result

After comparing some results, we can conclude that the list of chords contains at least one of the chord patterns produced by a human.

4.3. Experiment 3

For experiment 3, we create a review survey provided to users. We collected all the data from the 50 different users to check the review score, the result shows below:

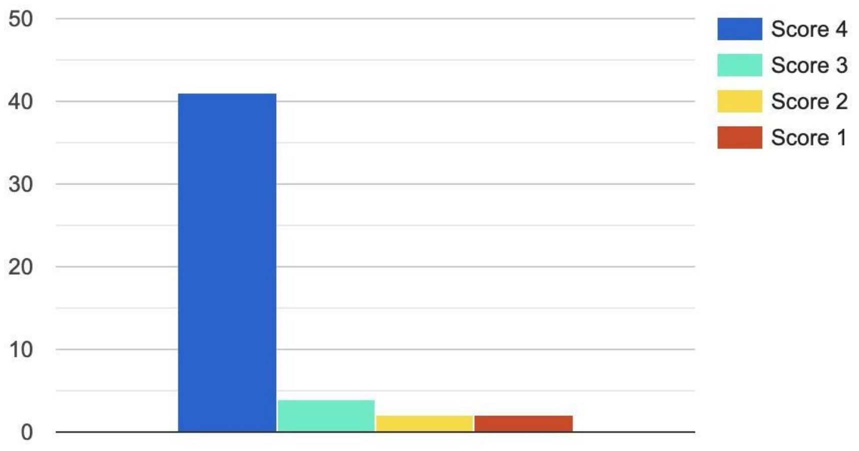


Figure 12. Result of Experiment 3

We can tell most of the users have a score as 4, which can prove the app works effectively.

To summarize, in the first experiment, we checked AI’s accuracy rate asking 10 participants to try inputting 5 different files. The results they received with an accurate rate 94%. For the second experiment,

We compared the AI’s outputs to humanistic outputs, which turns out at least one of the chord pattern matches for every tester. For the third experiment, we create a review survey to collect the review scores from users. 92% of the users score as the best score.

5. RELATED WORK

This research is very similar to mine, but this specially focuses on the input of a produced vocal, and it produces a superficial accompaniment [2]. Whereas my research and app inputs using any recorded melody line as well as any midi file, and it produces a unique chord progression that has not been superficially produced beforehand as a default output.

This research is based on using the user's humming to produce a ringtone [3]. But again, this research is based on pre-produced and superficial made output. It might give the same output to many different users since the ringtone has been pre-made from before. Whereas my research and app is based on a unique input and unique chord progression output system.

This research is based on the user inputting an existing song as “inspiration” and the system will produce a set of chord progressions that shares a similar vibe or characteristics as the chords inthe existing song [4]. This app is also made to aid beginner musicians but all it does is to give inspiration for the musicians to write a similar song to the ones they’ve input into the system. Whereas my research and system does a similar thing, but it is designed for the musician to have their own freedom and allows them to express music the way they want to; by setting the input into the user’s melodic idea and producing a set of unique chords that would perfectly match the melody.

6. CONCLUSIONS

To summarize, the app Melodyfi, is a new innovative way to the future of songwriting [14]. It is efficient yet accurate in generating chord progression. As we have seen from the experiments, the app is very accurate and can be worked closely with the musicians.

The current limitations to the algorithm is that we currently only have the basic major and minor chords. We have yet to implement more advanced chords such as the 7th chords, augmented, diminished chords, inversions, and etc.

We will continue to advance and update our app with the more advanced chords. In the next update we will for sure implement a better UI with more functions such as profile, chord bank, and more options for users to input and record audio.

REFERENCES

[1] De Mantaras, Ramon Lopez, and Josep Lluis Arcos. "AI and music: From composition to expressive performance." AI magazine 23.3 (2002): 43-43.

[2] Simon, Ian, Dan Morris, and Sumit Basu. "MySong: automatic accompaniment generation for vocal melodies." Proceedings of the SIGCHI conference on human factors in computing systems. 2008.

[3] Lee, Hong-Ru, and J-SR Jang. "i-Ring: A system for humming transcription and chord generation." 2004 IEEE International Conference on Multimedia and Expo (ICME)(IEEE Cat. No. 04TH8763). Vol. 2. IEEE, 2004.

[4] You, Shingchern D., and Po-Sheng Liu. "Automatic chord generation system using basic music theory and genetic algorithm." 2016 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW). IEEE, 2016.

[5] Tymoczko, Dmitri. "The geometry of musical chords." Science 313.5783 (2006): 72-74.

[6] Deren, Maya. "Amateur versus professional." Film Culture 39.1 (1965): 43-44.

[7] Krumhansl, Carol L. "Music psychology and music theory: Problems and prospects." Music Theory Spectrum 17.1 (1995): 53-80.

[8] Al-Masri, Eyhab, and Qusay H. Mahmoud. "Discovering the best web service." Proceedings of the 16th international conference on World Wide Web. 2007.

[9] Miller, Danny. "Configurations revisited." Strategic management journal 17.7 (1996): 505-512.
[10] DiMarzio, J. F. "The Splash Screen." Android Game Recipes. Apress, Berkeley, CA, 2013. 41-50.
[11] Suchman, Lucy. "Configuration." Inventive methods. Routledge, 2012. 62-74.
[12] Hansen, Per Brinch. Operating system principles. Prentice-Hall, Inc., 1973.
[13] Machamer, Peter, Lindley Darden, and Carl F. Craver. "Thinking about mechanisms." Philosophy of science 67.1 (2000): 1-25.
[14] Baker, Felicity, and Tony Wigram. Songwriting: Methods, techniques and clinical applications for musictherapy clinicians, educators and students. Jessica Kingsley Publishers, 2005.
[15] Bharucha, Jamshed J., and Keiko Stoeckig. "Reaction time and musical expectancy: priming of chords."Journal of Experimental Psychology: Human Perception and Performance 12.4 (1986): 403.