SCHOOL OF SCIENCE AND ENGINEERING

AI MUSIC COMPOSER
Capstone Design

Louai El Achkar
Supervised by Dr. Violetta Cavalli-Sforza

April 2020
Capstone Report  Student Statement:
I, Louai El Achkar, hereby affirm that I have applied ethics to the design process and in the selection of the final proposed design. I have also held the safety of the public to be paramount and have addressed this in the presented design wherever applicable.

_____________________________________________________
Louai El Achkar

Approved by the Supervisor

_____________________________________________________
Dr. Violetta Cavalli-Sforza
ACKNOWLEDGEMENTS

Working on this project was an interesting and fruitful experience that made me discover a new interesting field and expanded my knowledge. Therefore, I would first like to express my gratitude to my supervisor, Dr. Violetta Cavalli-Sforza, for agreeing to supervise this project and for giving me clear and concise feedback on how to improve my ideas.

I am very grateful to my parents and my sister for their continuous support not only for this project, but throughout my entire academic career. Without their support and motivation, I would not have been able to reach this far academically. I would also like to thank the friends, colleagues, professors, and staff that made my journey in AUI an enjoyable experience.

Finally, I would like to thank the different online communities for providing good quality documentation, tutorials, and examples that helped me progress in this project and allowed me to learn about different tools and methodologies.
# CONTENTS

1 Introduction 1

2 Background and Motivation 1
   2.1 Music Technology 1
   2.2 AI and Music 4

3 Feasibility Study 5
   3.1 Technical Feasibility 5
   3.2 Economic Feasibility 5
   3.3 Timeline 6

4 STEEPLE Analysis 6
   4.1 Social 6
   4.2 Technical 6
   4.3 Economic 7
   4.4 Environmental 7
   4.5 Political 7
   4.6 Legal 7
   4.7 Ethical 7

5 Requirements and Specifications 7
   5.1 Functional Requirements 7
   5.2 Non-functional Requirements 8

6 Technology Enablers 9
   6.1 JavaScript and Node.js 9
   6.2 TensorFlow and TensorFlow.js 9
   6.3 Magenta and Magenta.js 9
   6.4 Electron 9
   6.5 LitElement 10
   6.6 Ableton Live 10
   6.7 Max and Max for live 10

7 Design and Implementation 10
   7.1 Phase 1: Deep Learning Algorithm 10
      7.1.1 Autoencoders and MusicVae 11
      7.1.2 Dataset and Training 13
   7.2 Phase 2: Desktop Application 15
7.2.1 Back-end
7.2.2 Front-end
7.3 Phase 3: Plugin Implementation
7.4 Variations
7.5 Diagrams
7.5.1 Use Case Diagram
7.5.2 Software Architecture Diagram
7.5.3 Class Diagram
7.5.4 Sequence Diagram

8 Results
9 Limitations and Future Work
10 Conclusions
11 References
LIST OF FIGURES

Figure 2.1.1  MIDI Controller. From: Amazon.com................................................................. 2
Figure 2.1.2  Ableton Live (DAW) Dashboard. From: Ableton.com................................. 3
Figure 7.1.1.1 Encoder and decoder model. z is the latent factor. from: magenta.tensorflow.org/music-vae... 11
Figure 7.1.1.2 Concise diagram of an autoencoder model. from: magenta.tensorflow.org/music-vae........... 12
Figure 7.1.1.3 Decode variable................................................................................................... 13
Figure 7.1.2.1 Dataset i used........................................................................................................ 14
Figure 7.1.2.2 Convert the MIDI files into "Notesquence".......................................................... 15
Figure 7.2.1.2 Calling the model from electron .......................................................................... 16
Figure 7.2.2.1 User interface ....................................................................................................... 17
Figure 7.3.1  Adapting the generated notes to ableton live......................................................... 18
Figure 7.5.1.1 Use case diagram ................................................................................................ 20
Figure 7.5.2.1 Software architecture diagram ............................................................................ 21
Figure 7.5.3.1 Class diagram ....................................................................................................... 21
Figure 7.5.4.1 Sequence diagram ................................................................................................ 22
Figure 8.1  Generated files ........................................................................................................... 23
Figure 8.2  Generated files in ableton.......................................................................................... 24
LIST OF TABLES

Figure 3.3.1 Schedule of Work ........................................................................................................................................... 6
ABSTRACT

In this report, I document the main ideas and processes behind my Capstone project named AI Music Composer. This project aims to make a software program that uses Artificial Intelligence to generate coherent and novel music.

In this project I focused mainly in the development of a Deep Learning model using an Autoencoder, a type of Neural Network that is currently very popular for music generation. I also used Magenta; an open-source library powered by TensorFlow that provides tools for art generation using Deep Learning.

The Deep Learning model is implemented in a software program that can be used as both a standalone desktop application as well as a Virtual Studio Technology plugin. For this implementation, I used Electron, a framework that makes possible to use JavaScript to make cross platform desktop applications. To make my software program available as an audio plugin for Ableton Live, the most popular Digital Audio Workstation, I used Max, an environment that is used for creating audio and music software.
1 INTRODUCTION
Computational creativity is an extensive field that intersects computer science with the arts as well as other disciplines. The goal of computational creativity is to develop systems that can simulate, enhance, or understand human creativity [1]. Music is an art that is open to the different applications in computational creativity in many ways. In this project, I decided to develop a program that generates music in order to help or inspire the user to come up with different melodies and rhythms.

AI Music composer is a Deep Learning powered software program that generates music MIDI files. The software can either save the outputs generated or open them directly in a Digital Audio Workstation, where the user can modify the music piece. The main emphasis of the project is to develop a software program that makes use of the new state-of-the-art Deep Learning techniques of generating music in a real time manner and to incorporate them in a piece of software that is easy to use for users that do not necessarily have a background in anything but studio technology.

Music Software has been gaining much popularity among professional musicians and hobbyists. Using trendy technologies such as Artificial Intelligence (AI) makes this product interesting for the Music Software market.

2 BACKGROUND AND MOTIVATION
This project addresses a very specific topic, which is music generation, and, for that, some background knowledge is necessary to understand the scope of this project. Therefore, I will address two main factors: music technology and the use of AI in music.

2.1 MUSIC TECHNOLOGY
Nowadays we have many types of uses for technology when it comes to music, ranging from music recording to online music streaming services. In this project I will be working with MIDI files, which will be generated with the music generator and manipulated using a Digital Audio Workstation (DAW). The program can also be launched as a Virtual Studio Technology (VST) plugin in the DAW. Here are the definitions of these concepts.
MIDI

MIDI Stands for Musical Instruments Digital Interface and it is a format that was introduced in 1982. It is still very popular nowadays because it allows a clear representation of music notes in a digital format. It also provides the possibility to generate music notes using a MIDI controller. Most MIDI controllers adopt the shape of a piano as seen in figure 2.1.1 but they can be in any other form [2]. Other popular MIDI controllers are launchpads, samplers, and electronic drums, among others.

![MIDI Controller](https://via.placeholder.com/150)

**Figure 2.1.1  MIDI Controller. From: Amazon.com**

A MIDI file represents music notes with the pitch and the time of the notes, as well as the silence between them, as the main parameters. Thanks to the simplicity of the representation, other software programs can be used to perform digital signal processing on the MIDI files such as modulation [2]. This way, using the appropriate software, we can have any type of effects on the music track as well as assigning it any virtual musical instrument.

The reason why I chose to work with MIDI files instead of audio files is because I found it easier to use as data. An audio file is a representation of an analogic signal that has many properties such as noise. These properties need proper handling, and that can be a complicated process. It is also difficult to separate specific notes, especially in audio files that have many instruments playing at once. A MIDI file represents a sequence of notes that can be used as data for a software program without taking into consideration any other factor.
DAW

A Digital Audio Workstation is a software program used for recording and editing audio files. It can be used for all audio related tasks, and music is one of them. There are many options available that go from open-source, such as Audacity, to commercially available DAWs, such as FL Studio, Pro Tools, and Ableton Live [3]. For this project I used Ableton Live 10.

VST

Virtual Studio Technology, or VST, are software audio plugins that are run on top of a DAW and that provide audio manipulation options needed for studio audio production such as audio effects, virtual instruments, mastering, and others [4].

This technology is owned by Steinberg and it was launched in 1996. It is an open standard technology, and most audio software companies make some sort of VST product [4].

![Ableton Live (DAW) Dashboard. From: Ableton.com](image)

Figure 2.1.2 Ableton Live (DAW) Dashboard. From: Ableton.com

In figure 2.1.2 we can see the dashboard of the DAW Ableton Live 10 reading a MIDI sequence. The MIDI sequence is represented as piano notes.
2.2 AI AND MUSIC

The bridge between music and computer science was built a long time ago. The first ever documented notion of algorithmic composition was a German game called "Musikalische Würfelspiele" played in the 18th century, in which music is generated depending on the output of dice [5].

Markov chains were also used to generate music automatically. A Markov chain is a mathematical model used in many areas such as weather prediction, speech recognition, and many others. This model calculates the probabilities of node traversal and therefore generates data that is subsequent to other existing data. In 1958, the first use of Markov chains for music was documented when Iannis Xenakis released his work called "Analogique a et b" that used this model to generate music [6].

Up to this point it was only possible to generate music from previously inputted sub sequences. Later on, with the introduction of Neural Networks, it was possible to generate novel music from scratch by training a model with music data. In 1989, an attempt to make music in the style of J.S. Bach was made by using Recurrent Neural Networks, a type of Neural Networks. This approach was limited since RNN can only learn from a small section of a song because of a problem called the “vanishing gradient problem” [7].

The use of LSTM, or Long Short Term Memory, Neural Networks was introduced by Douglas Eck, a researcher at Google who later would lead the Magenta project, a Google project that develops tools for art creativity using AI. For my project I will use some of the tools and research of Magenta, including the use of Autoencoders, another type of Neural Networks [7].
3 FEASIBILITY STUDY

3.1 TECHNICAL FEASIBILITY

The technical feasibility study shall consider both the Machine Learning and software application aspects. For the Machine Learning part, the technologies I considered using were Python and its corresponding Machine Learning Libraries. For the software application part, C++ and a framework for audio software, JUCE, were considered.

However, after progressing on the report, I found out that some JavaScript frameworks were more suitable for this project. These frameworks include: TensorFlow.js, Magenta.js, and Electron. The reason why I ended up switching to JavaScript is because I found it difficult to implement Python Deep Learning models into a C++ program. Using a single language for both the desktop application and the Deep Learning model was more convenient for me. Also, I personally wanted to experiment with Tensorflow.js and Magenta.js since they are relatively recent tools.

The DAW used was Ableton Live, as initially specified.

3.2 ECONOMIC FEASIBILITY

For the economic feasibility, all the tools were free to use and open source, therefore, no financial cost was necessary to implement the project. For that reason, we can say that this project is economically feasible.
3.3 TIMELINE

A proposed schedule that includes the order and time required for the tasks was necessary to make sure that the project will be fully implemented before the capstone deadline. Overall, the project implementation focused first on the Machine Learning models to be used and then on the software application and its user interface. The main tasks that were assigned for my project as well as the personal deadlines I set for each are shown in Table 3.3.1.

![Table 1.3.1 Schedule of Work](image)

<table>
<thead>
<tr>
<th>Task</th>
<th>Personal Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements and Specifications/ Installation and preparation of needed tools</td>
<td>February 14th, 2020</td>
</tr>
<tr>
<td>Machine Learning Model Implementation</td>
<td>March 13th, 2020</td>
</tr>
<tr>
<td>Desktop Application Implementation/ Debugging</td>
<td>April 10th, 2020</td>
</tr>
<tr>
<td>Testing</td>
<td>April 16th, 2020</td>
</tr>
</tbody>
</table>

Due to unforeseen circumstances, including the COVID-19 outbreak, this schedule was impossible to follow. All the deadlines starting April, ended up being postponed.

4 STEEPLE Analysis

4.1 SOCIAL

From a social perspective, this product’s aim is to help music creators get a lead in their music creation, and therefore directly release the music or use it in other products such as videos, websites, games, etc.

4.2 TECHNICAL

State-of-the-art Deep Learning technologies were used as the basis of the project. Also, innovative music software frameworks were used.
4.3 ECONOMIC
For the scope of the capstone project, the product required no financial investment for its construction. However, this product might be made commercially available in the future at a competitive price.

4.4 ENVIRONMENTAL
There is no direct link between this project and any environmental factor. Except to the extent that the use of computers requires electricity, which itself has an impact on the environment.

4.5 POLITICAL
There is no direct link between this project and any political aspect of society.

4.6 LEGAL
The product will generate music that is not copyrighted; therefore, the user can legally own the music he produces using the product. Also, tools used to develop the project are available for free and therefore, legal to use.

4.7 ETHICAL
The music generated by the product will be copyright free, so it is ethically correct for the user to own it. There will be no chances of generating music that sounds like the music created and owned by someone else. Also, the product will ensure that the user’s privacy is protected by not requiring the entry of personal data from the user.

5 REQUIREMENTS AND SPECIFICATIONS
5.1 Functional Requirements

- The user should be able to get generated music from a music dataset.
  \(\text{Status: Fulfilled.}\)

- The user should be able to get the generated music as MIDI files.
  \(\text{Status: Fulfilled.}\)
• The user should be able to get the MIDI files opened in a Digital Audio Workstation.
  
  Status: Fulfilled
  
  The VST plugin version of the program was designed for Ableton Live.

• The user should be able to choose some of the parameters that he or she would like in the generated music such as key and tempo.
  
  Status: Unfulfilled
  
  For the prototype version, the parameters of the generated music are hardcoded into the program.

5.2 Non-functional Requirements

Security: The program should have a regulated access control that only lets authorized users (owners) access the software.
  
  Status: Fulfilled.

Performance: The application should be efficient in time and use of computer resources.
  
  Status: Fulfilled.

Portability/Compatibility: The program should work in all computers that run Windows (From Windows 7 onwards).
  
  Status: Fulfilled.
  
  The program can be built for Windows, MacOS, and Linux. This is possible thanks to the Electron packager.

Robustness: The product should avoid crashes and handle errors properly.
  
  Status: Partially fulfilled.
  
  The program crashes from time to time as a VST plugin in Ableton. This is the case for many audio plugins commercially available when used in this DAW, therefore it was inevitable.

Usability: The product should have a user-friendly interface and come with appropriate documentation.
  
  Status: Fulfilled.
6 TECHNOLOGY ENABLERS

6.1 JavaScript and Node.js

JavaScript (JS) is the main programming language that was used in this project. It is usually used for web applications but with Node.js and the framework Electron it was possible to build a desktop application [13].

Node.js, or Node, is an open-source cross-platform JavaScript environment. JavaScript used to be read only by browsers but with the introduction of Node.js, it became possible to run JavaScript in a specified server (a node server). Node allows to have asynchronous programming in which a piece of code can run before another one, even if it is written after [13].

6.2 TensorFlow and TensorFlow.js

TensorFlow is an open-source library that is used for Machine Learning applications, especially Deep Learning. It was developed by Google Brain and it was released in 2015. The library is optimized for Python but in 2018, a JavaScript version called TensorFlow.js was released. This version is still more limited than the Python version, however for the scope of this project, it was enough [9].

6.3 Magenta and Magenta.js

Magenta is an open-source library powered by TensorFlow. It is used mainly for manipulating music and images, and to use these data to train Deep Learning models. It has a more limited version called Magenta.js that is powered by TensorFlow.js. This library was created and is supported by the Magenta project, a Google Brain research team led by Douglas Eck [9]. The Magenta website and documentation provided me with extensive knowledge on how to use Deep Learning technologies to generate music.

6.4 Electron

Electron is an open-source framework. It allows the user to build desktop applications using web technologies such as HTML, CSS, and JavaScript. It uses the Chromium rendering engine and the Node.js runtime. It was created and is supported by GitHub [14].
6.5 LitElement

LitElement is a base class that is used to build web components that can be used in the front-end of a program. It was developed by Polymer [15].

6.6 Ableton Live

Ableton Live is a DAW developed by Ableton. It is available for Windows and MacOS. Like all DAWs, it can be used for recording, mixing, mastering, among other audio related activities. It is the most popular DAW available and the current version is Ableton Live 10 [3].

6.7 Max/Max4live

Max is a visual programming language that is used to create music and multimedia applications. The current version is Max 8 and it comes with a Node.js API that allows it to run JavaScript code. Max4Live is a version of Max that allows the creation of audio plugins for Ableton Live. It was created by Cycling’74 [12].

7 DESIGN AND IMPLEMENTATION

For the implementation of this idea, I had to work on three different phases: build the Deep Learning model, create the desktop application, and make the application being able to be opened through a DAW as an audio plugin. I will describe in detail how I performed each of these phases.

7.1 PHASE 1: DEEP LEARNING ALGORITHM

To generate music that is novel and coherent, I had to choose the best Machine Learning model that could do so. At first, I started by experimenting with TensorFlow for Python but then I had to switch to the JavaScript version. I had to do this change because, after doing research, I found that implementing a Python TensorFlow model and embed it in a C++ application would be very challenging [9]. Therefore, I chose to go for the option of building the Deep Learning model in JavaScript and implement it in a JavaScript based application. This way I could use one programming language instead of two, which was more convenient for me.
To build the Deep Learning model, I used the Magenta library. Magenta has many functions that allow the implementation of Deep Learning algorithms for generating and processing images, drawings, and music. Magenta happens to also be available for JavaScript under the name of Magenta.js which is very practical for me [9]. This library is open-source; therefore, I was able to modify the functions in order to make them fit into my program.

7.1.1 Autoencoders and MusicVAE

One function that I found interesting, and therefore used exclusively from the Magenta library is MusicVae. This function allows the use of Autoencoders to create music by interpolating music data. Autoencoders are different from Recurrent Neural Networks since they use an encoder and a decoder to learn from the data. In the case of music, a sequence of music notes is encoded into a latent factor and then decoded into a sequence again. When inputted in the latent factor, the qualities of the musical sequence are learned by the algorithm, and therefore we can apply operations that will give us a new musical sequence [10], as shown in figure 7.1.1.1.

![Encoder and decoder model](https://magenta.tensorflow.org/music-vae)

**Figure 7.1.1.1 Encoder and decoder model; z is the latent factor.**

From: [https://magenta.tensorflow.org/music-vae](https://magenta.tensorflow.org/music-vae)

A latent factor, or latent space, is a type of model that allows high dimensional input data. In this case I am using music sequences, which are high dimensional, meaning they have many variations such as silences, the time a note is pressed, etc. A latent factor allowed me to not enumerate such variations and take into consideration just the musical notes. This allows for more realistic and smooth interpretation of the music sequence, and eventually, smooth and realistic output can be generated [10].
The autoencoder encompasses the input data in a vector (the latent factor) and then decodes it into a generation of the same data. Through the process, the common qualities of the input, in this case, of the music sequence, will be retained in the latent factor. Therefore, when the input is a dataset, the model can generate an output that shares the common qualities of the dataset items [10]. In figure 7.1.1.2, we can see a representation of how an autoencoder would work for a musical sequence.

![Figure 7.1.1.2 Concise diagram of an autoencoder model. From: https://magenta.tensorflow.org/music-vae](https://magenta.tensorflow.org/music-vae)

The most challenging part was to modify the function by making it read the dataset I specified, and train the model in the most optimal way. Since I decided to be inputting music of just some specific styles, I allowed some overfitting so it can output logical, nice-sounding musical melodies that listeners can relate to a music style.

One of the key parameters to check is the degree of overfitting. It is represented as “temperature” in the program (See figure 7.1.1.3). The lower the temperature the higher the overfitting degree, and vice versa. If the temperature is high, the program will generate music that is very chaotic and not similar at all to any of the songs that the model used for training. If it is low, the outputted melody will be similar to one of the melodies in the dataset. I set the temperature so I can get a 50% degree of overfitting. This way we can get melody sequences
that sound according to the musical style of the songs in the training dataset, yet that are still original to the generation.

```
abstract decode(
  z: tf.Tensor2D, length: number, initialInput?: tf.Tensor2D,
  temperature?: 1.0, controls?: tf.Tensor2D): tf.Tensor3D;
```

**Figure 7.1.3** Decode variable.

### 7.1.2 Dataset and Training

Training a Deep Learning model needs a lot of processing power and the larger the data, the more time it takes the model to complete the training. TensorFlow.js is much slower than the Python version when it comes to training the data since it uses WebGL to accelerate the TensorFlow training. WebGL is a JavaScript API for rendering graphics. In TensorFlow for Python, the computer’s GPU is used for the model training, allowing the model to train faster [9].

To avoid having to wait for days for the training to complete, I had to train the model with a dataset that is not very large. However, the dataset that provided me with a source of MIDI music sequences needed to be extensive enough to generate coherent and nice sounding music. For that reason, I decided to train my first model with just a specific style of music. Since classical piano music is the most available music genre in MIDI format, I decided to use some datasets of this style to train my model. There is a website called “Classical Piano Midi Page” that provides MIDI files of various piano songs of classical music artists such as Mozart, Bach, and Beethoven. I used a small subset of the files provided to train my model as shown in figure 7.1.2.1.
14

Figure 7.1.2.1 Dataset I used

To train the model, I had to convert my collection of MIDI files into note sequences that can be read by the specialized function in the model. To do so, I specified the parameters of the songs such as ticks per quarter, and a duration. This allows the file of the dataset to be inputted to the model directly from its path. Therefore, the model could start the training and recognize the common qualities of the dataset. Then the program outputs MIDI sequences with the same qualities as the dataset and with the parameters I specified. As seen in figure 7.1.2.2, all the generated music will be in 4/4 time signature and in 60 bpm.
7.2 PHASE 2: DESKTOP APPLICATION

7.2.1 Back-end

For the Desktop Application, I had initially planned to work with JUCE, the most popular framework for audio applications. However, after switching to JavaScript exclusively, I had to find a way of implementing my TensorFlow.js model into a desktop application built with JavaScript. After research, I found that Electron, a JavaScript framework, is a good option since...
it allows the implementation of TensorFlow.js. Electron is used to develop desktop cross-platform applications [14].

```javascript
import { MusicVAE, sequences } from '@magenta/music'
const { quantizeNoteSequence, unquantizeSequence, clone } = seq

export class Model {
  constructor()
  
  const composeSeq = resolve(modelPath, 'DeepModel')
  this.model = new MusicVAE(composeSeq)

  async load(){
    try {
      await this.model.initialize()
    } catch (e){
      const dialog = document.createElement('Create')
      document.body.appendChild(dialog)
    }
  }

  async sample(i, z=1){
    let generated = await this.model.sample(i, z)
    generated = generated.map(time => unquantizeSequence(time, 60))
    generated.forEach(seq => seq.notes.forEach(n => n.velocity = 100))
    return generated
  }
}
```

**Figure 7.2.1.2** Calling the model from electron

In figure 7.2.1.2, I show how I called the Deep Learning model from Electron using Node.js. The model is packed with the program and has a specific path. I use the “generated” variable to hold the created music to be outputted.
7.2.2 Front-end

The front-end of the program consists of a window that prompts the user for the folder to output the files in, and a button to generate the files (see figure 7.2.2.1). I used Polymer LitElement to create the elements that compose the front-end layers of the program, called components, such as the generate button.

![Image](AIComposer.png)

Figure 7.2.2.1 User interface

7.3 PHASE 3: PLUGIN IMPLEMENTATION

The last part of my project is to allow the program to be opened in a DAW. DAWs read external programs as plugins if they are in some type of VST format. The best way to do so for a JavaScript program is to use a framework called Max. Max allows to build JavaScript audio plugins since it includes a Node.js API [13].

With Max, I used the same code for my standalone application to build an audio plugin version. I had to make sure that the output MIDI would be opened directly in the DAW. To build it for the most popular DAW, Ableton Live, I had to use a Max version called Max4Live. This way, my program could be adapted to the Ableton Live interface and I was able to perform basic tasks such as to automatically create audio tracks in the DAW.
One of the key tasks was to make the MIDI files be read and outputted into the DAW. Max provided features to achieve this goal as seen in figure 7.3.1.

```javascript
function decodeMidi(note, pitch, time, duration, velocity=100, muted=0) {
    return {
        pitch, time,
        startTime: time,
        endTime: time + duration,
        duration, velocity, muted
    }
}
```

```javascript
function decode(abletonNotes) {
    if (abletonNotes.shift() !== 'notes') {
        throw new Error('not read!')
    }
    const reNotes = []
    const count = abletonNotes.shift()
    const selected = 0
    for (let i = 0; i < count; i++) {
        const reversedNotes = abletonNotes.slice(i, selected + 1)
        reNotes.push(decodeMidi(reversedNotes))
    }
    reNotes.sort((a, b) => a.time - b.time)
    return reNotes
}
```

```javascript
function encodeNote(note) {
    const [pitch, startTime, endTime, velocity=100, muted=0] = note
    return [pitch, startTime, endTime, velocity, muted]
}
```

```javascript
function encode(notesArray) {
    let ret = []
    notesArray.forEach((note) => ret.push(encodeNote(note)))
    return ret
}
```

**Figure 7.3.1  Adapting the generated notes to Ableton Live**

### 7.4 VARIATIONS

Depending on how large a dataset is and on the computer’s processing power, the Neural Networks model takes several days and even weeks, to complete. The end user, obviously, should not have to wait that amount of time to use the product.

To overcome this limitation, I decided to adopt a technique in which software makers release variations of the same software by just changing some parameters. This way they get software variations that may include different pre-sets in what is essentially the same program. An example would be how Native Instruments and NeuralDSP sell different variations of one amplifier simulator. These variations consist of tweaking the parameters of the program to get different sounds, yet it is still the same base program.
In my case I decided to use different datasets for the training of the neural network model. This way I could end up with music generators for different styles that would come with pre-trained models implemented within them. Some possible variations of the software, among others, could be:

- AI Music Composer: Classical
- AI Music Composer: Techno
- AI Music Composer: Rock

Another interesting type of variation would be having pre-trained models that combine more than one style of music. This way the outcome would be innovative music sequences and could inspire many interesting music ideas. Also, we could have pre-trained models trained with the most popular melodies of a specific style of music. This way we could get melodies with the properties of the songs that were popular at some point of time, which is also an interesting idea. The different possibilities of pre-trained models are endless.

Users would be able to download one or more versions of the program. Since the program is not very heavy and the installation is simple, the users will be able to download as much AI Music Composer variations as they want.

7.5 DIAGRAMS

7.5.1 Use Case Diagram

The use case diagram represents the actions the user takes with the software. Since the software application has only one task, to generate music, this use case represents the interaction with the DAW and the generated music from the program. Figure 7.5.1.1 shows the use case diagram of the program.
7.5.2 Software Architecture Diagram

The software architecture represents the different components used in the program and the relationship between them. This program consists of the Deep Learning model that is built separately and comes packed within the software. The model works as the backend for the program and it interacts with the front end in the Electron framework. Node.js fetches the output of the model and stores it in a file.

To use the software as an audio plugin for Ableton Live, a different version can be built using Max. The software Max will run the Electron app code through a Node.js server and will make the program available as an audio plugin that can be opened directly in Ableton Live.

In figure 7.5.2.1 we can see the comprehensive software architecture diagram for the program.
7.5.3 Class Diagram

The class diagram represents the attributes of the classes and the relationship between them. The program ended up having many classes, mainly because of all the frameworks I used. I deemed it unnecessary to include all of them. Instead, as seen in figure 7.5.3.1, I made a simple class diagram to represent the main classes behind this project.

![Class diagram](image)

**Figure 7.5.3.1 Class diagram**

7.5.4 Sequence Diagram

The sequence diagram shows the interaction between the user and the components of the program. It specifies the order of each action and the involvement of each component. In this program the user only acts by requesting the generation of a musical sequence. The dashboard of the program will take the input and send it to the Deep Learning model through Node.js. The
Model will take some time to generate the MIDI files and then ressend it to the Node.js backend to output it in a specified folder. For the audio plugin the sequence is the same. First the user needs to initialize the DAW, and the generated MIDI files will be outputted and opened directly in the DAW. In figure 7.5.4.1 we can see the sequence diagram for the program.

Figure 7.5.4.1 Sequence diagram
8 RESULTS

The results of this project are pretty satisfying, considering how new and unstable are the technologies I am using for music generation. The program crashes many times especially when used as a plugin. This might be because of imperfections in my code, but it is probably due to the recency and instability of the technologies I am using.

The program outputs eight files in a folder that the user specifies (see figure 8.1). So far, I did not include the possibility to choose the number of files to be outputted and it is set by default. If it is used as a plugin, it outputs eight MIDI sequences in the DAW as shown in figure 8.2.

The music generated is quite simple yet feels natural and coherent. I believe it is good quality music generation. Even though simple, the melodies generated can be used as a base for any musical project. The user can generate as much music as needed to find the melodies that work best for any purpose.

Figure 8.1 Generated files
9 LIMITATIONS AND FUTURE WORK

This program has limitations of two types. The first are the actual limitations related to the state-of-the-art Deep Learning technologies and how it is not possible, or it is very difficult, to implement certain tasks.

In my case, and through research, I found that it is difficult to adapt the neural network model to train with songs that are not in the chromatic scale of western music. The chromatic scale of western music has twelve notes (semitones) in an octave. For this instance, microtonal notes like quarter notes, will not work for the program. This is due to the fact that most DAWs and other music software require special programs and handling to read microtonal notes from MIDI files. This complication extends to the technologies I used for the Deep Learning model.

An example of this limitation would be traditional Arabic, Turkish, or any variation of microtonal music that uses scales with quarter tones. This limitation is not only related to the scope of this project, but to the entire music manipulation field, and it is an interesting topic for future research.
Also, it will not be able to generate music that has odd time signatures that is usually found in progressive music. To implement this, the model needs to learn how to properly insert time signature changes into the music sequences. This could be done in the future.

The second type of limitations are the limitations related to the technology choices I made. An interesting improvement to the program would be the possibility to develop it for the JUCE framework. I tried doing so before switching to JavaScript, however I found out that developing Deep Learning models for a C++ audio application was a hard task. However, it is still an interesting option considering how JUCE is very popular for audio and music applications.

10 CONCLUSIONS

The outcome of the project, even though quite simple, fulfilled most of the intended requirements. Music creators can download a version of the program and have it generate musical melodies for them. Then they can use these melodies for inspiration, help, or just for fun.

As I went through the research and implementation for this project, I realised how broad and interesting the field of Music computation is. It makes use of many fields and disciplines such as computer science, music theory, psychology, etc. It was a challenging experience yet a very fruitful one, since it made me discover new state-of-the-art technologies and methods.

This project made me realize that music creation using Computer Science has many opportunities that I could work on for my career, as both Computer Science and music are my main interests.
11 REFERENCES


