



National University of Computer and Emerging Sciences



TuneGen: Artificially Intelligent Music Generation & Composition Engine

FYP Team

Muhammad Maarij	19L-2347
Hadiya Kashif	19L-1120
Riva Nouman Khan	19L-1117

Supervised by

Dr. Mubasher Baig

FAST School of Computing

National University of Computer and Emerging Sciences

Lahore, Pakistan

March 2023

TuneGen: Artificially Intelligent Music Generation & Composition Engine

This is to declare that the above publication produced under the:

Title: TuneGen: Artificially Intelligent Music Generation & Composition Engine

is the sole contribution of the authors and no part hereof has been reproduced on **as it is** basis (cut and paste) which can be considered as **Plagiarism**. All referenced parts have been used to argue the idea and have been cited properly. We will be responsible and liable for any consequence if violation of this declaration is determined.

Date: 12th October 2022

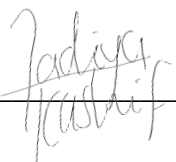
Student 1

Name: Muhammad Maarij

Signature: 


Student 2

Name: Hadiya Kashif

Signature: 

Student 3

Name: Riva Nouman Khan

Signature: 

Authors' Declaration

The authors listed above declare that the work presented in this report is their own and has not been submitted/presented previously to any other institution or organization.

Abstract

In our project, we aim to create TuneGen, an Artificially Intelligent system that can generate novel music based on the user's initial input. However, unlike other AI-based composition systems currently developed, the input given by the user will be in the form of humming a tune or uploading a short MP3 file that they want the system to work off. This will be used as the initial input for our Machine Learning algorithm. Our project's goal is to enable music producers and artists to be able to compose audio tracks for their songs. The output from this engine will be a completely unique music composition.

Executive Summary

With a focus on both the consumer and corporate industries, TuneGen intends to address the issue of how time-consuming music production is. Based on the brief melody that the user sings or hums into their microphone, it will generate context-based music. After the initial data entry, no more human input will be made; instead, all music will be produced by the software on its own. This article intends to showcase the development processes for the artificially intelligent system TuneGen, which can generate creative music based on the user's first input. The user can upload a small MP3 audio or hum a tune to provide the system with input.

TuneGen would make it possible for musicians and music producers to create audio tracks for their tunes. A tune may be hummed into the system by the user, and the engine will utilize it as a starting point to create a full composition, or a tune can be uploaded as an MP3 file. The SDG-9 (Industry, Innovation, and Infrastructure) targets the development of a strong infrastructure, the promotion of just and sustainable industrialization, and the promotion of innovation. To manage upcoming difficulties and reduce our carbon footprint, our infrastructure will be located in the cloud rather than on separate servers. Users use 84% less power, 77% fewer servers, and 88% less carbon emissions when they use the cloud.

While being more difficult and decision-based, the process of making music is comparable to that of composing written language. Harmony, form, melody, and instrumentation are considered to be the four main essential components of music. Every composition a composer writes has a basic melodic or harmonic idea. Both short-term and long-term relationships make up these conceptions.

The topic of algorithmic composition has three main branches: automata-based, statistical models, and machine learning models.

First, there are automata-based models, which are used to express formal languages in finite form. According to formal language theory, grammar describes how to create alphabetic strings that are appropriate for the language's syntax. L-systems, ATNs, and Grammatical Evolution are the three different categories of automata-based approaches. At every stage of the derivation process, L-systems, a type of parallel rewriting formal grammar, apply all rewriting rules concurrently. Compared to other forms of grammars, they are more often utilized in algorithmic composition. Genetic algorithms are used by Grammatical Evolution to change and apply grammatical rules. A sort of finite state automaton or graph structure called augmented transition networks may parse very complicated languages.

Second, we have statistical models, which include hidden Markov models and Markov chains. The idea of state and transitions is used by the context-based approach known as Markov Chains. The values of the states are mapped onto musical objects, and the transition value represents the probability needed to go from one state to the next. The hidden parameters are discovered using a hidden Markov model (which are states; musical objects in case of musical composition). With great success, HMMs are employed in the creation of music.

Last but not the least, there are machine learning models, mostly utilized for algorithmic music synthesis. These models include RNN-LSTM, Genetic, CNN, GAN, and VAE.

Recurrent neural networks (RNNs) are a subset of neural networks that process sequential input and data across a structure resembling a differential graph. When the neural network has to receive and retain data for a considerable amount of time, LSTMs are helpful. Information may

readily pass along the chain unaltered because there aren't many interactions over the whole length of it. Then we have Genetic Algorithms, named so because they mimic real-world genetics and population dynamics. The majority of GAs essentially reproduce in cycles of evaluations, selection, and mutation. A fitness function and heuristic technique are used to evaluate each person's level of fitness (which can be empirical or quantitative). Machine learning also includes convolutional neural networks. It is a particular form of artificial neural network that is utilized for tasks like processing pixel data and image analysis. Convolutional/hidden layers make up a CNN; these levels receive input and transmit output to the layer below them. Then we have GANs which are made up of two neural networks: a generator and a discriminator. The generator's objective is to generate fake images that look like real data but do not contain the required data. Discriminator is trained using both actual domain images and false images in order to identify the best image. Lastly there are VAEs which consist of an encoder and a decoder. An encoder is a collection of layers that compresses and maps the input to a distribution to reduce its dimension. A sample from the distribution is then utilized to rebuild the input during decoding.

Table of Contents

Table of Contents	i
List of Tables	iii
List of Figures	iv
Chapter 1: Introduction	1
1.1 Purpose of this Document	1
1.2 Intended Audience	2
1.3 Definitions, Acronyms, and Abbreviations	2
Chapter 2: Project Vision	3
2.1 Problem Domain Overview	3
2.2 Problem Statement	3
2.3 Problem Elaboration	3
2.4 Goals and Objectives	3
2.5 Project Scope	3
2.6 Sustainable Development Goal (SDG)	4
Chapter 3: Literature Review / Related Work	5
3.1 Definitions, Acronyms, and Abbreviations	5
3.2 Detailed Literature Review	5
3.2.1 An Overview of Music Composition	6
3.2.2 Early Work on Algorithmic Composition	6
3.2.3 Machine Learning Based Methods	10
3.2.4 Shortlisted ML Method for this Paper: RNN-LSTM	13
3.3 Literature Review Summary Table	14
3.4 Conclusion	16
Chapter 4: Software Requirement Specifications	17
4.1 List of Features	17
4.2 Functional Requirements	17
4.3 Quality Attributes / Non-Functional Requirements	17
4.4 Assumptions	18
4.5 Hardware and Software Requirements	18
4.5.1 Hardware Requirements	18
4.5.2 Software Requirements	18
4.6 Use Cases	18
4.6.1 Upload File	18
4.6.2 Record Audio File	19
4.6.3 Create Music	19
4.6.4 User Download	20
4.7 Proposed Graphical User Interface	21
4.8 Risk Analysis	21
Chapter 5: Proposed Approach and Methodology	22
5.1 Phase 0: Collecting the Dataset	22
5.2 Phase 1: Training the Model	22
5.2.1 Step 1: Parsing the input files	22
5.2.2 Step 2: Creating the training data	22
5.2.3 Step 3: Training	22
5.3 Phase 2: Getting Predictions – Generating Music	23
5.3.1 Step 1: MP3 Input	23
5.3.2 Step 2: Run Input Through LSTM	23
5.3.3 Creating Final Output	23

Chapter 6: High-Level and Low-Level Design	24
6.1 System Overview	24
6.2 Design Considerations	24
6.2.1 Assumptions and Dependencies	24
6.2.2 General Constraints.....	24
6.2.3 Goals and Guidelines	24
6.2.4 Development Methods	25
6.3 System Architecture.....	25
6.4 Architectural Strategies.....	25
6.4.1 Python Language	25
6.4.2 Extensible.....	25
6.4.3 Metaphor UI Paradigm	25
6.4.4 Client - Server Model.....	26
6.5 Domain Model/Class Diagram	26
6.6 Sequence Diagrams.....	26
6.7 Policies and Tactics.....	27
6.7.1 Using Python (with Django) to implement the Backend	27
6.7.2 Using ReactJS for the Frontend	27
6.7.3 Evolutionary Prototyping Design Paradigm	27
6.7.4 Empirical Testing.....	27
6.7.5 Single Static Webpage	27
Chapter 7: Implementation and Test Cases	28
7.1 Implementation	28
7.1.1 Dataset Selection.....	28
7.1.2 Data Loading.....	28
7.1.3 Preprocessing	28
7.1.4 Model Architecture	29
7.1.5 Training.....	29
7.1.6 Evaluation	29
7.1.7 Deployment.....	29
7.2 Test Case Design and Description	30
7.2.1 MIDI File Scraping Test Case	30
7.2.2 Dataset Loading and Preprocessing Test Case	31
7.2.3 Model Training Test Case.....	31
7.2.4 Music Composition Test Case	32
7.2.5 Sheet Music Output Test Case.....	33
7.2.6 Upload, Record and Play Music Test Case.....	34
7.2.7 Download Generated Music Test Case.....	35
7.3 Test Metrics	36
7.3.1 Test Case Metrics.....	36
7.4 Conclusion	36
Chapter 8: User Manual	38
8.1 Step 1: Upload or Record Audio.....	38
8.2 Step 2: Select Options	38
8.3 Step 3: Download ZIP.....	38
Chapter 9: Experimental Results and Discussion	39
Chapter 10: Conclusion and Future Work	40
References.....	41

List of Tables

Table 1: Evolution of Music Generation Algorithms	14
---	----

List of Figures

Figure 1: L-Systems Drawing Plants	8
Figure 2: Proposed Graphical User Interface.....	21
Figure 3: Algorithm for Generation of Chords	23
Figure 4: System Architecture	25
Figure 5: Class Diagram	26
Figure 6: Sequence Diagram.....	26
Figure 7: Final User Interface	38

Chapter 1: Introduction

A source of inspiration and a blank canvas for many, music is a fascinating topic that surrounds us continually. It is a measure to the creativity and competence required, that for some people, making music is a subject worth dedicating their entire lives to. The notion of developing a systematic method to create music has existed for millennia, even though composition is a complex kind of art that necessitates a profound grasp of the human experience.

In its simplest form, algorithmic composition — also known as "automated composition", describes "the process of using some formal process to make music with minimal human intervention" [1]. Music has been using such "formal procedures" since the dawn of time. However, the label itself is rather recent, with the word "algorithm" coming from the disciplines of computer science and information science at about the midpoint of the 20th century [2]. Composers now have more chances to automate the creative process thanks to computers. In addition, as we will see there are several distinct approaches that have emerged in the last 30 years or more.

Many autonomous music generating algorithms based on deep learning have also recently been proposed. Long music pieces with distinctive musical features are still a difficult challenge to produce since they strongly rely on musical patterns. By utilizing the fundamental ideas of artificial intelligence and generative machine learning models, TuneGen seeks to address the issue of the time taking task of music creation and will be aimed at the consumer and business markets. With this application, we can provide users the option of letting technology create a personalized and unique music composition based on their initial tune as an input, which will save them the resources, time, and effort in the long run.

Because it will be a cutting-edge model and will produce context-based music based on the brief tune that the user delivers or hums into their microphone, TuneGen is being referred to as a "tune-based AI music composer" in contrast to other available AI music generators. There will be no more human involvement after this first input of data; instead, the software will create all the music on its own. As a result, TuneGen will be a genuine music composer rather than a music classification or recommendation engine.

This report will consist of 3 chapters. This first introductory chapter being where we define the purpose for our project and it's intended audience, along with all the definitions, acronyms and abbreviations that will be used throughout this report. The second chapter will take readers through the vision for our project. It will define and elaborate our problem statement and the domain which it belongs to. We will also define our goals and objectives including sustainable development goals. The third and last chapter in this report will focus on our literature review and provide an overview of all the research papers that we have studied.

1.1 Purpose of this Document

This document aims to highlight the methodologies used to develop an artificially intelligent system called TuneGen that can produce original music depending on the user's first input. The user will provide the system with input via humming a song or uploading a brief MP3 recording, unlike other AI-based composing systems that are presently under development, our machine learning system will utilize this as its initial input. The purpose of our project is to make it possible for musicians to create audio tracks for their compositions. This engine will produce an entirely original musical composition.

1.2 Intended Audience

The intended audience of this project are people in the music industry such as artists, composers, and music producers. TuneGen can prove to be a huge help to new singers who are not familiar with the process of music composition. It can be a source of inspiration for composers who might be going through a creative block.

1.3 Definitions, Acronyms, and Abbreviations

SDG: Sustainable Development Goal

AI: Artificial Intelligence

ML: Machine Learning

MIDI: Musical Instrument Digital Interface - is a technical standard that describes a communications protocol which connects a variety of electronic musical instruments, computers, and other related audio devices.

Sheet Music: Sheet music is a type of musical notation that is handwritten or printed and includes musical symbols to describe the pitches, rhythms, or chords of a song.

Chapter 2: Project Vision

This section specifies the overall goals and objectives of the project along with its scope and sustainable development targets to be achieved.

2.1 Problem Domain Overview

TuneGen will take a user's first input and utilize it to generate creative music. The user's input will instead be in the form of humming a tune or uploading a small MP3 file that they want the system to get ideas from and this will be the initial input for our machine learning system. Our endeavor intends to make it easier for musicians and music producers to construct audio tracks for their songs. This engine will produce an entirely unique musical composition as its final product.

2.2 Problem Statement

Since the beginning of time singers have had to hire a music composer to produce a beat for their songs. TuneGen will provide these services without costing the singer a huge amount of money. Singers don't need any prior composition knowledge to use this software.

2.3 Problem Elaboration

Firstly, this paper will start off by looking at how music is composed by human composers and musicians and what empirical criteria they use to evaluate their work. Next part of the research will be working on selecting a machine learning model to focus on and then looking for an appropriate dataset, which will ideally be a corpus of various types of music in MIDI format. Next up, this paper will work on implementing the selected machine learning algorithm and training it with the dataset downloaded. Lastly, we will polish the final product by creating a user interface that can receive inputs from the user in the form of a short audio clip that will be the seed for the ML model. This audio clip will first be pitch corrected and then converted into MIDI format to be fed into the model. The result that gets returned will be played back to the user as an MP3 or MIDI. Alternatively, the user would be able to opt for receiving the output as a piece of sheet music in PDF format.

2.4 Goals and Objectives

We hope to achieve the following milestones in our research and project:

- Explore different generative machine learning models such as:
 - Recurrent Neural Networks
 - Long Short-Term Memory
 - Genetic Algorithms
 - Convolutional Neural Nets
 - Generative Adversarial Networks
 - Variational Autoencoders
- The user can either hum a tune to the system and the engine will use that as a starting point to generate a complete composition or they can upload an MP3 file containing a tune to get the same result.
- The output generated will be in the form of an MP3 or MIDI file.
- The user can also choose to receive this final composition in the form of sheet music.

2.5 Project Scope

Our project is divided into two major phases that must be addressed effectively:

- Exploratory / Research Phase:
 - Explore preexisting algorithms on music compositions, and investigate the techniques used. We will also evaluate their success based on how the results they produce.
 - Identify features that music producers currently use to create cohesive music compositions.
- Implementation Phase:
 - Create a feature set relevant to how music is composed by humans.
 - Gather data and create the most complimentary machine learning model for the problem at hand.
 - Get user input in the form of a tune.
 - Before feeding the provided tune to the neural network we will preprocess by using a pitch correction algorithm on the data provided.
 - Create an audio output for the user and generate corresponding sheet music.

2.6 Sustainable Development Goal (SDG)

In this project we will be targeting SDG-9 (Industry, Innovation, and Infrastructure) which aims to construct robust infrastructure, advance fair, and sustainable industrialization, and encourage innovation. Every prosperous community is built on a solid infrastructure that is both functional and durable. Our infrastructure will be housed in the cloud rather than on individual servers to handle future challenges and lessen our carbon impact. In many respects, the cloud is transforming the IT sector. By using the cloud, users utilize 84% less electricity, 77% fewer servers, and produce 88% less carbon emissions [3]. We will support cutting-edge green technology and guarantee that everyone has access to our system. This will guarantee that we create a successful product that can be utilized all around the world.

Chapter 3: Literature Review / Related Work

3.1 Definitions, Acronyms, and Abbreviations

AI: Artificial Intelligence

VAE: Variational Autoencoder

GAN: Generative adversarial network

SVM: Support Vector Machine

LSTM: Long short-term memory

HMM: Hidden Markov Model

CNN: Convolutional Neural Network

GUI: Graphical User Interface

ATN: Augmented Transition Network

GA: Genetic Algorithm

NLP: Natural Language Processing

RNN: Recurrent Neural Network

L-System: Lindenmayer system

Temporal: Relating to time

Quantization error: Round-off error

Neural network: A computer model that simulates the working of human brain

Genre: A classification based on the style of music

Corpus: A huge collection of documents

Viterbi algorithm: Tells the maximum probability of most likely sequence

Normal distribution: It is a bell-shaped curve that is used to represent random variables whose distribution are unknown. The width of the curve represents the standard deviation and there is a vertical symmetric line which represents the mean value

Automated: A process happening without human intervention

Stochastic: Something that is categorized to be random.

Chords: multiple notes played together

Arpeggios: broken chord played in reverse order

Turtle Graphics: Turtle graphics are vector images that use the "turtle" as a pencil to draw on a Cartesian plane (x and y axis).

Chord Progression: the sequence of chords in a song or other piece of music

3.2 Detailed Literature Review

This section will include a detailed literature review of the problem at hand categorized by different types of work done in the past including early methods of algorithmic music generation and modern machine learning based methods. In addition, this section will provide a summary table that describes each paper reviewed, along with references to them.

3.2.1 An Overview of Music Composition

Similar to how written language is created, the process of creating music is intricate and highly decision dependent. The method used varies depending on the music type with which we are dealing. For instance, it is rather typical in classical music to begin with a brief motif i.e., a unit of one or two bars - and build upon that to create a melody or musical phrase, but it is more typical in jazz music to take a chord progression and improvise a melody ahead of it [18]. No matter what musical genre we are writing in, every piece of music a composer starts has a fundamental melodic or harmonic notion. Having said that, we may consider music to be a sophisticated language model made up of both short-term and long-term associations [5]. These connections span across two dimensions: the temporal dimension, which is connected to the structure of music, and the harmonic dimension, which is connected to the notes/pitches and the chords/harmony [5].

From the perspective of music creation, the four key fundamental music elements, based on Walton's concepts [19] as follows:

1. Harmony - the process of joining or composing distinct sounds into larger units or compositions.
2. Musical Form - describes the organization of a musical composition or performance. Some of the factors that influence musical form [20] are:
 - The arrangement of musical units of rhythm, melody, and harmony.
 - The way the instruments are arranged
 - How a composition is orchestrated
3. Melody/Tune - is a series of musical notes that are heard as a single unit by the listener.
4. Instrumentation - the specific arrangement of musical instruments used in a piece, as well as the characteristics of each instrument separately. Instrumentation is also sometimes referred to as Orchestration.

3.2.2 Early Work on Algorithmic Composition

Throughout history there has been a significant interest in the composition of music using automated means so that humans can enjoy music without having to work through the tedious and time-consuming process of actually creating it from scratch.

Automated composition of music refers to the creation of music with bare minimum human intervention [12]. This method of creating music is also referred to as the, algorithmic composition. Creating art using a limited number of predefined sets of rules helps the composers come up with new compositions in a very short period of time. This helps singers by giving them more time to work on their vocals and lyrics and worry less about the composition.

In this section this paper will discuss the early techniques employed by mathematicians, scientists and musicians in order to aid them in creating music.

3.2.2.1 Pre-Computer Era Methods

When we talk about algorithmic music composition our first instinct is usually that this methodology would be computer based, but as a shock to many this started off as a non-computer-based practice [12].

Famous philosophers like Pythagoras, Ptolemy and Plato, all believed that there was a strong association between music and mathematical theory. They believed that an extensive knowledge of mathematics and numbers would be the key to understanding all the workings of the universe and beyond. [13] The algorithms i.e., the step-by-step procedures were all implemented through mathematics. Even though these were considered groundbreaking deductions at the time, these remained strictly theoretical and there is no actual proof of it ever being implemented. This definitely facilitated the artists in terms of how they operated and practiced, but humans were never completely removed from the process. [12]

Later, another idea was proposed which involved the imitation of the melody at a specific time after a short delay. This was considered a level up from what the Greeks discovered considering there was no interference by the composer in the majority part of the process. The composer makes one melody which is then used by the singers to make an entire song. [14]

Mozart used a method that included putting together a number of minor musical pieces and merging them at random, creating a new composition from sections that were selected at random. Similarly, John Cage used the sounds of the chess board when there was a game going on, to get a new beat every time, another example involving randomness. Another example of this could be Karlheinz Stockhausen's Klaveirstucke XI, where the pianist performs some musical piece in random order. [12] All these approaches are considered to be stochastic approaches.

In efforts to completely make the composition process abstract, movements like the twelve-tone method and serialism were founded. During this time all the decisions were made by randomly defined numeric arrays, these decisions included: notes and rhythms. These movements were founded in attempts to automate the entire process [12].

3.2.2.2 Automata Based

The study of abstract machines and the computing issues that they may be used to address is known as automata. Formal language theory and automata theory are closely linked disciplines. Automata are employed as finite representations of formal languages. Furthermore, grammar, according to formal language theory, explains how to construct strings from an alphabet that are acceptable in accordance with the syntax of the language. This is done so by recursively applying grammar rules on a word base to form complete and coherent sentences. Because the rules are applied recursively, grammars are very well suited to express hierarchical systems [10]. Hence, given that most musical forms exhibit hierarchical patterns, it is not unexpected that formal grammar theory has long been used to study and create music [21].

There are three key concepts from automata that have historically been used in the research field of algorithmic music composition.

3.2.2.2.1 L-Systems

Lindenmayer Systems, or simply L-systems, are a sort of parallel rewriting formal grammar that employs all rewriting rules simultaneously at each step of the derivation process. Because they can describe hierarchical self-similarity and are easier to understand and use than traditional formal grammars, L-systems are more frequently used in algorithmic composition than other types of grammars [10]. Furthermore, L-Systems can be used in visually stunning ways to create patterns using turtle graphics and have been used in the creation of realistic looking 2D and 3D plant and tree renderings (see figure XYZ below). It makes sense that the first time L-systems were applied to algorithmic composition, turtle graphics were employed

to create an image that was then translated/interpreted into a musical composition [22]. Furthermore, L-systems may be utilized to construct tools that just solve a component of the composing process, such as simply supplying the composition's base chord progression.



Figure 1: L-Systems Drawing Plants

*L-Systems can be used to draw turtle graphics of plants
And trees – these images can be used to then compose
Interpretive music*

3.2.2.2.2 Grammatical Evolution

Grammatical evolution is a genetic programming technique in which grammars have been used in conjunction with evolutionary approaches where the genomes are grammars that are specified via a GUI, and the fitness function is interactive i.e. the fitness of the grammars is determined by the user. The most typical technique is to represent music as the output of the grammar, which might be highly general or quite specialized to a specified music style. Like formal grammars in general, L-systems have also been built using evolutionary algorithms. The bulk of cases, however, make use of an empirical fitness function in which a person determines fitness [10]. An alternative approach is where the composer must also program a "critic" function, which listens to the numerous automatically produced outputs at various stages of the processing to decide which are "fit" or suitable for final output [12]. This latter approach however was not widely used as music tends to be highly subjective and cannot be quantitatively evaluated.

3.2.2.2.3 ATNs

Augmented Transition Networks are a type of graph structure or a finite state automata machine which is able to parse relatively complex languages [10]. An example of the usage of ATNs for music generation can be found in Experiments in Musical Intelligence (EMI), developed by David Cope in 1992, which is a piece of software that can examine a collection of musical compositions written in a particular style. The software uses pattern-matching algorithms to locate signatures (short musical sequences that are typical of the style of the dataset being analyzed). These inferences are used to drive the ATN [23]. Following this analysis, the synthesis phase develops new music compositions that comply with the criteria stored in the inferred ATN, yielding an impressive outcome. However, it is crucial to note that Cope's EMI and his research methodology are heavily scrutinized with Wiggins [24] stating that Cope does not justify the argument; he only states it, and that the work "piles error upon error".

3.2.2.2.4 Rule Learning

Rule based learning as the name suggests is a model where a set of procedures or rules are followed through which the functioning of the program is carried out. These levels are often designed so that the result of the prior level determines its trajectory into the upcoming level. Contrary to the stochastic approach which was sheerly dependent on randomness, rule-based learning follows a set of laws or formally referred to as the “grammar”, to formulate the composition. This approach is similar to the one proposed by Lejaren Hiller and Robert Baker they made a software called MUSICOMP which was a database of compositional subroutines. [15] Artists could make a composition using these subroutines or they can integrate one or two subroutines into their already existing composition. Like that system these include a collection of already existing or newly created compositional work. Examples of these rule learning systems could be William Shottstaedt's automatic species counterpoint program which automates music based on seventy-five rules and CHORAL by Kemal Ebcioglu based on 350 rules. [12] A software MUSE developed by Schwanauer [26] is also an implementation of rule learning, where the system follows the predefined rules determined by the developers and the system alters the priority of each rule based on some variables. New rules are also derived based on the patterns identified in the results. Rule learning can also be seen in hybridized models with evolutionary algorithms where they introduced fitness functions. Rather than conforming to general methods which use a binary mechanism it instead transitioned into a graded scale based on the fitness scores.

3.2.2.3 Statistical Models

A piece of music can be regarded as a sequence of notes also known as musical objects. A statistical model helps identify the constancy in a class of music based on the genre or style. This model is then used by assigning a higher probability to the music pieces that belong in the class and a lower probability otherwise. The parameters of this model are set by a corpus of musical pieces within a class and during the testing phase models of the same class are then evaluated on an unseen test data set [9].

3.2.2.3.1 Markov Chains

Markov Chains is a context based statistical technique [9] which uses the concept of state and transitions. The states are mapped onto musical objects and the value of transition denotes the probability required to transition from a current state to the next one. These probabilities are stored in a probability matrix which can be produced from either an existing composition corpus or can be generated on our own. In the basic Markov Chain model, the next state depends on the current state only but it can be extended to an nth order Markov Chain which then uses n number of previous states rather than just the current state. Among the many notable early implementations of Markov Chains is of Olson, who used the existing music compositions to produce a new composition. The limitation of this model depends on the number of states (n) used to estimate the probability of a new state. If the value of n is less, the music produced is usually scattered and has a strange composition but if the value of n chosen is high then the composition produced is very similar to the existing fed composition and training of such a model is very expensive [10].

3.2.2.3.2 Hidden Markov Model

A hidden Markov model is used to find the hidden parameters (which are states; musical objects in case of musical composition) from a set of observable parameters (dependent on the hidden parameters). An output matrix is to be created along with the transition probability matrix which has probabilities of every output from all the possible states. As the output sequence can be produced by many state sequences, the most likely sequence (which has the maximum

probability) can be found by the Viterbi dynamic programming algorithm. [10] HMM's are used in music generation with quite success. An example of usage of HMM can be found by Allan in 2002 where Viterbi algorithm was used to find the most reasonable sequence of the melody [9].

3.2.3 Machine Learning Based Methods

Machine learning is a topic of research that focuses on the study of techniques that can make computers “learn” and recognize patterns, or techniques that utilize data to enhance performance on a certain set of tasks. It is gaining interest as a tool for music synthesis in addition to conventional tasks like prediction, classification, and translation, as demonstrated by recent research groups like Magenta at Google and the Creator Technology Research Lab at Spotify [8]. By exploiting the ability of deep learning architectures and training methods to automatically learn musical composition from any dataset, the goal is to produce samples that are novel and appeal to humans. This section discusses a study of current approaches that use various machine learning and deep learning methods to improve music generation.

3.2.3.1 RNN + LSTM

A recurrent neural network is a specific kind of neural network that uses sequential input and information. Recurrent weights are used to apply the same set of weights repeatedly over a differential graph-like structure. RNNs have enormous potential for use in natural language processing [25] and therefore, by extension, can also be applied to the field of algorithmic music generation. Furthermore, to prevent problems with reliance over time, a particular type of recurrent neural network, LSTMs are employed [7]. LSTMs are very useful when the neural network has to gather and store information for a long time and have shown promise in the field of music generation due to the fact that music, like a sophisticated language model, is made up of both short-term and long-term associations [5] over a temporal domain. The LSTM's fundamental component is the cell state, which performs like a conveyor belt. Due to the fact that there are very few interactions along the whole chain, information may easily travel through it unmodified. The LSTM model is capable of deleting or adding information to the cell state by carefully manipulating components referred to as gates.

The network suggested by Agrawal et al. [7] is intended to learn chord-note correlations in MIDI files. The network that was built is then utilized to produce music completely autonomously, allowing humans to generate a wide range of compositions. The work was completed with the use of the Music21 library and the Python-based Keras framework and creates music object files for input into the architecture. The authors state that the results, while not perfect, are extremely good, showing that the neural network can be used to make music and has the ability to develop more complicated musical extracts. The LSTM model is capable of accurately representing long temporal relationships [7].

Mariano Schmidt [4] proposes their method, known as MelodyRNN, and is built on an LSTM. Through the use of training techniques or constructing neural network topologies, MelodyRNN enables the user to modify the pitch range of a MIDI file. A series of commands in the application, which was created on top of Google's Magenta, can be used to construct a dataset from a MIDI file and extract the melodies from each track, which can then be used to train the model. The software used jazz melodies and children's songs to train the models from scratch.

3.2.3.2 Genetic Algorithms

Genetic Algorithms are called so because they model genetics and population evolution as seen in real life. The bulk of GAs basically follow this pattern - there is a continually changing population of individuals that goes through a cycle of evaluation, selection, and variation in reproduction. As the first phase, the initial populations must be generated in a more or less random way or from user specified examples. Each candidate's quality is then assessed using a fitness function and a heuristic method (which can either be empirical or quantitative). The following stage is selection: A new population is formed by reproducing potential solutions from the preceding population. The number of times each candidate is copied is proportional to its fitness. This stage reduces the diversity of the population, which is subsequently boosted by applying some operators designed to increase variation (such as mutation) on a small number of candidate solutions. These processes are repeated iteratively; as a result, average and best fitness tend to improve with time.

In regards to using basic GA techniques take Marques et al. [27] as an example. They created brief tunes using a basic fitness function and chromosomal representation. The outcomes were deemed to be satisfactory. In order to create brief melodies, Johnson et al. [28] employed an GA with a fitness function which was the weighted sum of a number of the melody's most crucial local properties. Similar to this, Birchfield [29] used a hierarchical EA with a fitness function that was a huge weighted sum of multiple characteristics. Each individual in a population was composed of members of lower populations, and there were several levels of population. From the output of the GA, he assembled a long composition for 10 instruments.

On the more advanced side Han [6] suggested a genetic algorithm-based sampling technique to enhance character representation in sheet music. The traditional algorithm for algorithmic music composition can directly extract features from the pitch and duration of music and encode them in a heat-independent form, but it is unable to characterize the rhythm of pitch and duration, which prevents the composition network from learning music style more effectively. The method proposed attempts to fix this problem. To generate rhythm, pitch and time value are mixed in accordance with a preset rule. The beat of the song has an impact on its general style. The association between pitch and time value coding helps to better retain the rhythmic style of music, and the composition network can more quickly pick up on the traits of different musical styles.

3.2.3.3 CNNs

Convolutional neural networks lie under the umbrella of machine learning. It is a specific kind of artificial neural network that is used for tasks like image analysis and processing pixel data. A CNN has some type of specialization which helps it find patterns and make sense of it, which is the reason why it is so popular. A CNN consists of convolutional/hidden layers, which receives input and sends output which works as an input to the successor layer. [31]

The convolution layer is the first layer and a fully connected layer is the last layer, there can be multiple convolution layers too. Most calculations are performed on the convolution layer. A pooling layer is also similar to a convolution layer where a filter is applied to the entire layer and the dot product is taken with the original data. All the computations and calculations are performed on matrices as the data and the filter are both in matrix form. The advantage of a pooling layer is that the number of parameters is reduced in it compared to the convolution layer. In the last layer images are classified depending upon their characteristics. Everything in a CNN is not connected as if that was the case the cost would boost drastically. [31]

Each layer successively detects and extracts features, applies an activation function to them. Each layer subsequently identifies more complex features partially. This is why the object has to convolve through so many layers in order to completely get recognized. [31]

Generating music using deep learning has attracted a lot of attention in recent years but there are a lot of areas of concern when it comes to this. Music has a lot of variables that have to be kept into consideration e.g., music is classified into layer has dependencies which have to be kept in account, it also contains multiple instruments and different combinations of chords, melodies and arpeggios. [11]

In order to bring the idea of algorithmic music generation to real life a model was proposed which was a hybrid of CNN and LSTM. WaveNet is a model based on CNN is a Generative Model [33] which is developed by Google Mind. Here, this model's purpose is to guess the melody (like an NLP language model) and LSTM is a type of RNN which takes care of dependencies. This model was trained on a dataset containing Turkish pop music and the clean version of Nottingham. In order to be sure that this was the best implementation the people conducting the experiment made three models: one only using CNN, one only using LSTM and a hybridized model. Based on the survey results, customers seemed to enjoy the melodies created by the hybrid model more than any other, hence proving it to be relatively successful. [33]

3.2.3.4 GANs

The GAN model consists of two neural networks, one of which is known as a generator- it is used to create fake images and the other one is known as a discriminator- its purpose is to distinguish if the produced image by the generator is fake or belongs to the real domain. Discriminator is trained by both the real domain images and other fake images so it can be able to identify the best image. In terms of music the images are later converted to music files. [16] The purpose of the generator producing fake images is to eliminate the factor of misinterpretation of images which are similar to real data but not the data required and this purpose is fulfilled by the discriminator. [30]

In music generation it is important to store data of the previously generated note to create the new one so the system approach of GAN proposed by Rajat [16] uses two LSTM models to serve this purpose. One model act as a generator which produces noise and the discriminator LSTM finds the error between the original sample and the noise produced and instructs the generator to update its weights accordingly. Until the discriminator can no longer find any error this process is carried out and later when GAN is able to produce new music files, they are sent to the SVM (Support Vector Machine) Classifier to identify the best image which can be converted to music file.

3.2.3.4.1 Conditional GANs

Just like the basic GAN model the conditional GAN model also consists of two neural networks; a generator and a discriminator, along with some new information which is conditioned with both the neural networks. This makes sure that the added information and the image produced are trained together for output. [17]

An implementation of the conditional GAN is used in lyrics-conditioned melody generation [17] where the generator and discriminator are conditioned on lyrics. A lyrics-melody dataset was generated and with the use of trained skip-gram model lyrics vectors were extracted. Later a conditional GAN model was used to establish a relationship between the melody and lyrics.

After performing this proposed model, a quantization error leads the generated note to destroy the rhythm because of its unsuitable time duration but by using LSTM the probability of this error can be minimized.

3.2.3.5 Variational Autoencoders

A variational autoencoder has 2 basic components; an encoder and a decoder. An encoder is a bunch of layers which reduces the dimension of the input by compressing it and mapping it to a distribution having 2 vectors; mean vector and a standard deviation vector. To decode, a sample from the distribution is then used to reconstruct the input. The latent space therefore needs to have the properties of continuity and completeness. The VAE is then trained by using a loss function which defines the error between the original and reconstructed image and a term which ensures that the latent distribution is close to standard normal. [32]

3.2.4 Shortlisted ML Method for this Paper: RNN-LSTM

3.2.4.1 Summary of the research item on selected method

Mariano Schmidt [4] introduces their MelodyRNN technique, which is based on an LSTM. MelodyRNN allows the user to change the pitch range of a MIDI file by using training techniques or building neural network topologies. A sequence of instructions in the software, built on Google's Magenta, can be used to generate a dataset from a MIDI file and extract the melodies from each track, which can then be used to train the model. To train the models from scratch, the programme employed jazz tunes and children's songs.

3.2.4.2 Critical analysis of the research item

One of the strengths of using an RNN when compared to other ML models, is that RNN-based approaches typically need less time to train. MelodyRNN, for example, completes training in a matter of hours. Furthermore, it can create basic melodies and is able to output decent results that have cohesion and harmony. Additionally, RNNs are able to handle inputs of any length and when an RNN-LSTM model is used, it can easily retain time-step information. The model size of an RNN does not grow even when the input sizes are increased which makes this ideal for processing and generating music since it can be of different lengths.

Looking at some of the disadvantages of RNN-LSTM approaches, the network's computation process is quite slow and may take some time to return an output. Moreover, when using the tanh activation function, the model becomes sluggish, and it becomes tedious to process longer samples. Lastly, one of the biggest weaknesses of an RNN is that it suffers from problems such as Vanishing Gradient and Gradient Explosion.

3.2.4.3 Relationship to the proposed research work

An RNN is a type of neural network that employs sequential input, and music can be thought of as sequential data that spans time. RNNs have immense promise for usage in natural language processing, and hence may be applied to the field of algorithmic music production as well. Furthermore, to avoid difficulties with dependency over time, LSTMs, a sort of RNN, are used. LSTMs are particularly useful when the neural network must gather and store information for an extended period of time, and they have shown promise in the field of music generation because music, like a sophisticated language model, is composed of both short-term and long-term associations across a temporal domain.

3.3 Literature Review Summary Table

Table 1: Evolution of Music Generation Algorithms

The summary of various methods employed to generate music composition- including pre-computer era, early methods and machine learning methods is presented here.

No.	Name, reference	Technique Name	Description
1.	Music Composition with Deep Learning: A Review, [5] <i>Basic Forms in Music</i> , [19] <i>Worlds of music</i> , [20] <i>Connections between the Musical and Life Experiences of Young Composers and Their Compositions</i> , [18]	An Overview of Music Composition	These papers discuss the key of fundamental elements of music composition.
2.	A Brief History of Algorithmic Composition, [12] The History and Philosophy of Algorithmic Composition, [13] Canon music, [14]	Pre-Computer Era Methods	These papers discuss methods opted by famous philosophers to generate music using randomness.
3.	Score generation with L-systems, [22] AI Methods in Algorithmic Composition: A Comprehensive Survey, [10]	L-System	These papers discuss how parallelism is used to apply grammar rules.
4.	AI Methods in Algorithmic Composition: A Comprehensive Survey, [10] A Brief History of Algorithmic Composition, [12]	Grammatical Evolution	Grammatical Evolution is a genetic programming method where the grammar is changed in every generation and the fitness for each generation is determined empirically by the user.
5.	AI Methods in Algorithmic Composition: A Comprehensive Survey, [10] Computer Modeling of Musical Intelligence in EMI, [23] Computer Models of Musical Creativity, [24]	ATN	ATN is a type of graph structure which can process complex languages and therefore music too. Completely new music compositions can be created by using the stored criteria in the inferred ATN.
6.	<i>Machine models of music</i> , [26] Discovering rule-based learning systems for the purpose of music analysis, [15]	Rule Learning	In these papers it is discovered that a pre-defined set of rules is followed

	A Brief History of Algorithmic Composition, [12]		known as subroutines. These subroutines can be used to create a new composition or can be integrated to existing compositions.
7.	AI Methods in Algorithmic Composition: A Comprehensive Survey, [10] Music Generation from Statistical Models, [9]	Markov Chains	It is a context-based model which uses current state (note) to predict the next note and the transition between states shows the probability.
8.	AI Methods in Algorithmic Composition: A Comprehensive Survey, [10] Music Generation from Statistical Models, [9]	Hidden Markov Model	Viterbi dynamic algorithm is used to find the most probable sequence of notes.
9.	How to Generate Music with AI, [4] Music Composition with Deep Learning: A Review, [5] Automated Music Generation using LSTM, [7] Parsing natural scenes and natural language with recursive neural networks, [25]	RNN+LSTM	LSTM helps overcome the problem of RNN by storing the information for a long time to generate long outputs for music generation.
10.	Music composition using genetic evolutionary algorithms, [27] Evolutionary Computation Applied to Melody Generation, [28] Evolving intelligent musical materials, [29] Research on the Interaction of Genetic Algorithm in Assisted Composition, [6]	GA	By using fitness functions dependent on crucial local properties or dependent on multiple characteristics melodies can be generated.
11.	Deep Learning for Music Generation, [11] What are convolutional neural networks? [31] Style-Specific Turkish Pop Music Composition with CNN and LSTM Network, [33]	CNN	A CNN is a model that helps identify trends in the provided input data. A dataset is previously used to train this model in order for it to identify these patterns
12.	Generative Adversarial Nets, [30] IJERT-Survey on Deep Learning in Music using GAN, [16] Conditional LSTM-GAN for Melody Generation from Lyrics, [17]	GAN	Basic GAN and conditional GAN models both can be used for composition of music with the help of a generator and discriminator.
13.	Generating Music Using Deep Learning, [32]	VAE	VAE compress the input data which is later reconstructed by a sample from the distribution using a decoder.

			This is one of the deep learning methods which can be used to generate music.
--	--	--	---

3.4 Conclusion

In this section we discussed the evolution of algorithmic music composition, starting off with pre computerized methods leading up to GANs, LSTM, VAE, GA and RNNs. We discussed each of these techniques in detail with examples of already existing implementations and their drawbacks. After reading through a lot of research papers we came to the conclusion that RNN and LSTM will be the most suitable for the kind of system we foresee ourselves creating.

Chapter 4: Software Requirement Specifications

This section will describe all modules of system requirements and design along with the necessary diagram and figures. It will describe functional requirements, design constraints, and other factors necessary to provide a complete and comprehensive description of the software.

4.1 List of Features

Our system will have the following list of features:

- User can upload their own MP3 file or record one in the browser.
- User can select an instrument they want the output to be synthesized as
- User will also have the option to download the output audio as sheet music
- User can select if they want their audio to be pitch corrected before it is passed to the music generation function.

4.2 Functional Requirements

- **MP3 File Upload and Recording:** The user should be able to upload their own MP3 file or record one directly in the browser. The system should be able to accept MP3 files with a maximum size limit.
- **Instrument Selection:** The user should be able to select the instrument they want the output to be synthesized as. The system should provide a list of instruments from which the user can select.
- **Audio Download:** The user should be able to download the output audio in MP3 or MIDI format. The user should also have the option to download the output audio as sheet music in PDF format.
- **Pitch Correction:** The user should have the option to select if they want their audio to be pitch corrected before it is passed to the music generation function. The system should be able to perform pitch correction on the input audio.
- **Music Generation:** The system should be able to generate an entirely unique musical composition based on the user's input in the form of a short audio clip. The system should be able to convert the audio clip into MIDI format and feed it into the machine learning algorithm.
- **User Interface:** The system should have a user-friendly interface that can receive inputs from the user in the form of an audio clip and provide the output as a musical composition. The user interface should also provide options for instrument selection, pitch correction, and audio download.

4.3 Quality Attributes / Non-Functional Requirements

- **Usability:** The system should have a user-friendly interface that makes it easy for users to input their music, select the desired output format, and access the generated music.
- **Performance:** The system should be able to process the input music and generate the output in a timely manner. The performance should not be affected by the number of users accessing the system at the same time.
- **Reliability:** The system should produce consistent and accurate results, without any errors or crashes.
- **Security:** The system should be secure and protect user data from unauthorized access.
- **Compatibility:** The system should be compatible with different web browsers and operating systems.

- **Maintenance:** The system should be designed in a way that makes it easy to maintain and update in the future.
- **Documentation:** The system should come with clear and comprehensive documentation, including instructions on how to use it, installation, and maintenance.

4.4 Assumptions

1. Client connects from a suitable web browser.
2. Client has adequate internet speed as to not hinder connectivity.
3. The dataset used provides adequate results.

4.5 Hardware and Software Requirements

Here is a list of our Hardware and Software Requirements needed to deploy this project.

4.5.1 Hardware Requirements

1. Sufficient GPU for training the model.
2. Server with adequate computing and networking capacity

4.5.2 Software Requirements

1. Windows 10
2. Latest version of Python3 with additional
 - a. Music Processing Libraries
 - b. Sci-Kit Learn for Machine Learning Models
 - c. MIDI Processing Libraries
 - d. Other libraries such as Matplotlib, NumPy and Pandas

4.6 Use Cases

This section lists use cases or scenarios from the use-case model if they represent some significant, central functionality of the final system, or if they have a large architectural coverage.

4.6.1 Upload File

Name		Upload file	
Actors		User	
Summary		Allows the user to upload a file to the system	
Pre-Conditions		The file must be in MP3 format and not longer than 20 seconds.	
Post-Conditions		The file has been uploaded to the system and is ready to be sent to the next phase.	
Special Requirements		None	
Basic Flow			
Actor Action		System Response	
1	The user clicks on the file upload button	2	System opens the file explorer to allow the user to upload the file there

3	The user selects an MP3 file from the device and uploads it	4	System shows a message saying, “file uploaded successfully”
Alternative Flow			
3A	The user does not select an MP3 file	4A	System shows an error message “Please select a valid MP3”

4.6.2 Record Audio File

Name	Record audio		
Actors	User		
Summary	Allows the user to hum an audio clip to the system		
Pre-Conditions	The recording must not be longer than 20 seconds.		
Post-Conditions	The file has been uploaded to the system and is ready to be sent to the next phase.		
Special Requirements	Microphone Permission Enabled in the Browser		
Basic Flow			
Actor Action		System Response	
1	The user clicks on the record audio button	2	System asks for microphone access
3	The user grants access to their microphone	4	System starts recording the audio.
5	The user hums a tune into the mic for 20 seconds	6	System shows a message “recording successful”
Alternative Flow			
3A	The user does not grant microphone access	4A	System shows an error message “Please grant permission to access microphone before continuing”

4.6.3 Create Music

Name		Create Music	
Actors		User	
Summary		The user will send the short mp3 clip to the music composition generator	
Pre-Conditions		The user must upload an mp3 file for the create music button to be enabled.	
Post-Conditions		The user successfully receives a music composition file.	
Special Requirements		None	

Basic Flow			
Actor Action		System Response	
1	The user checks the boxes that they want their audio to have	2	The system returns the output file that is ready to be downloaded
Alternative Flow			
3	The user checks no boxes	4-A	The system responds with an error message saying, “select an option”

4.6.4 User Download

Name	User Download		
Actors	User		
Summary	The user will make the music composition available locally.		
Pre-Conditions	There must be a file returned after the create music functionality is called.		
Post-Conditions	The user’s session is successfully downloaded and directed to the generated audio file.		
Special Requirements	None		
Basic Flow			
Actor Action		System Response	
1	The user clicks the download file button	2	The system prepares to download the file to the device.
3	The user selects the place where the file should be downloaded	4	The system returns the file generated and downloads it to the location selected.
Alternative Flow			
3	The user clicks the multiple files option	4-A	The system returns the files generated and downloads them all.

4.7 Proposed Graphical User Interface

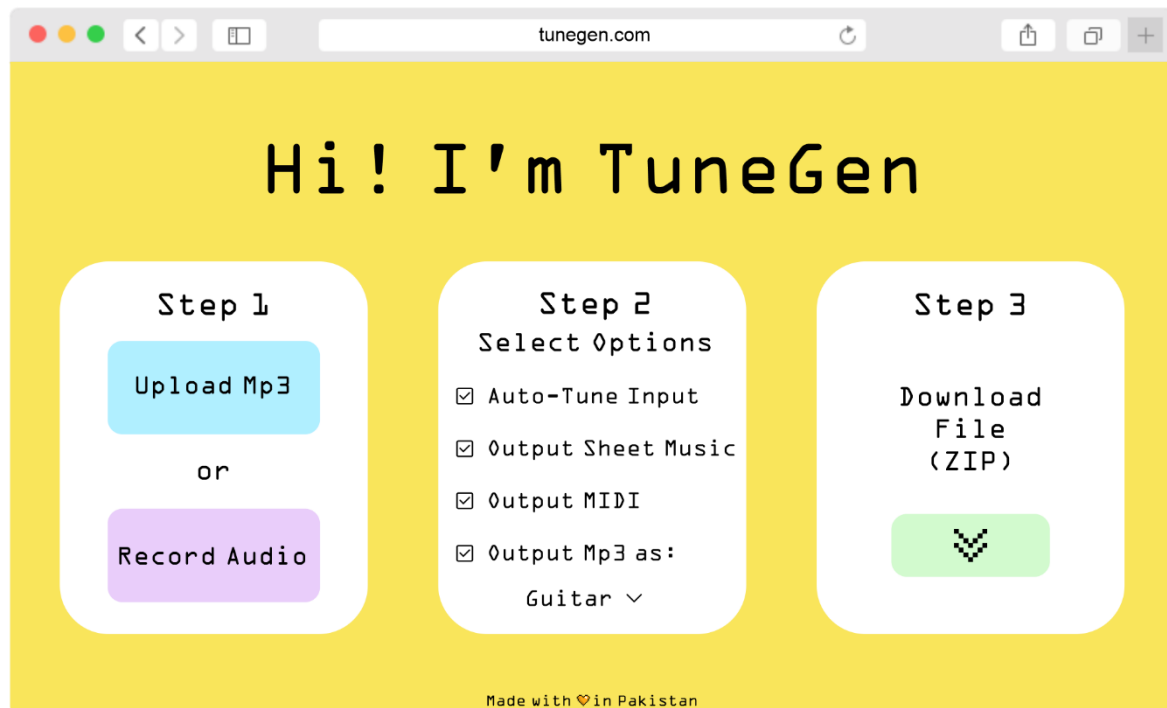


Figure 2: Graphical User Interface

The GUI of the application implemented as a basic web page with all user functionality selectable via check-boxes. Also supports being able to select an instrument from the output MP3 dropdown menu to synthesize the music.

4.8 Risk Analysis

1. Code Issues
 - a. Inefficient code
 - b. Logical errors
2. Schedule Risk
 - a. Inaccurate time estimation
 - b. Project scope expansion
 - c. Failure in completion
3. Unmet Expectations
 - a. Imprecise budgeting
 - b. Inaccurate outputs
 - c. Incorrect estimation of deadlines
4. Budget Issues
 - a. Overruns in costs
 - b. Unpredicted growth of project scope
 - c. Inadequate budget estimation
5. Technical Risks
 - a. Use of outdated technology which has no scope in future
 - b. Extremely complex implementations
 - c. Module integration issues

Chapter 5: Proposed Approach and Methodology

This chapter will outline the proposed algorithm pertaining to how our system will generate music at the backend. This will form the crux of this entire project. The algorithm chosen to tackle the problem is an LSTM model.

5.1 Phase 0: Collecting the Dataset

We will be collecting the dataset from various sources and combining into one large dataset. It will compose mainly of MIDI files (or MP3 files converted to MIDI). We will be focusing on one instrument mainly (The Piano) as the input to the model. This is because pianos can be found in a variety of genres.

5.2 Phase 1: Training the Model

The following steps highlight the training algorithm being used in this research.

5.2.1 Step 1: Parsing the input files

We will read the .MID files in the input folder and convert them into an appropriate format. Each file will be stored as a string of chords and notes. Next this string will be encoded as numbers, each chord in the music vocabulary being replaced by one integer (these integers can also then be one hot encoded). This is done because neural networks perform much faster on numerical data than string data.

5.2.2 Step 2: Creating the training data

Next the algorithm will go through the pre-read dataset and separate out input and proposed outputs for the model to use. This can be done by using the number of chords. From the sequence we can use the first 50 notes as input and choose the next 70 chords as output. The input will be used in training of the model and the output chords will be compared against what the model generates. This evaluation step will make the model better at predictions over time.

5.2.3 Step 3: Training

The LSTM model will be created using Keras TensorFlow and will contain the following layers

1. LSTM layers – RNN layers that take a sequence as input and can return either sequences or a matrix.
2. Dropout layers – regularization approach that involves changing part of the inputs during training to prevent overfitting. The fraction that is changed is determined by a variable factor.
3. The Fully Connected Layer / Dense Layer – is a completely linked neural network layer in which each input node is coupled to each output node. This part of the model will help in diffusing the output from the LSTM.
4. The Activation Layer – decides the activation function the model will use to generate an output from the output vocabulary.

The input data and output data will be used to fit the model with loss being calculated at every epoch.

5.3 Phase 2: Getting Predictions – Generating Music

5.3.1 Step 1: MP3 Input

The user will input an MP3 file by either uploading it or recording the tune in the browser. This input will be capped at 20 seconds. This input will be fed into the program. If the auto-tune input parameter is selected, the file will also be run through a pitch correction library. The resultant object will be converted into a MIDI object and then a string of chords and notes, exactly like in 5.1.1. The string will also further be encoded into integers (and also one hot encoded later).

5.3.2 Step 2: Run Input Through LSTM

We will load the model trained during phase one and then run the input through it. We will choose a sequence size and then send a sequence of chords/notes to the network for each note that we wish to create. The first sequence we provide is the note sequence at the beginning. As shown in Figure 3, for each succeeding sequence that we utilize as input, we will delete the first note of the sequence and insert the output of the previous iteration at the end of the sequence.

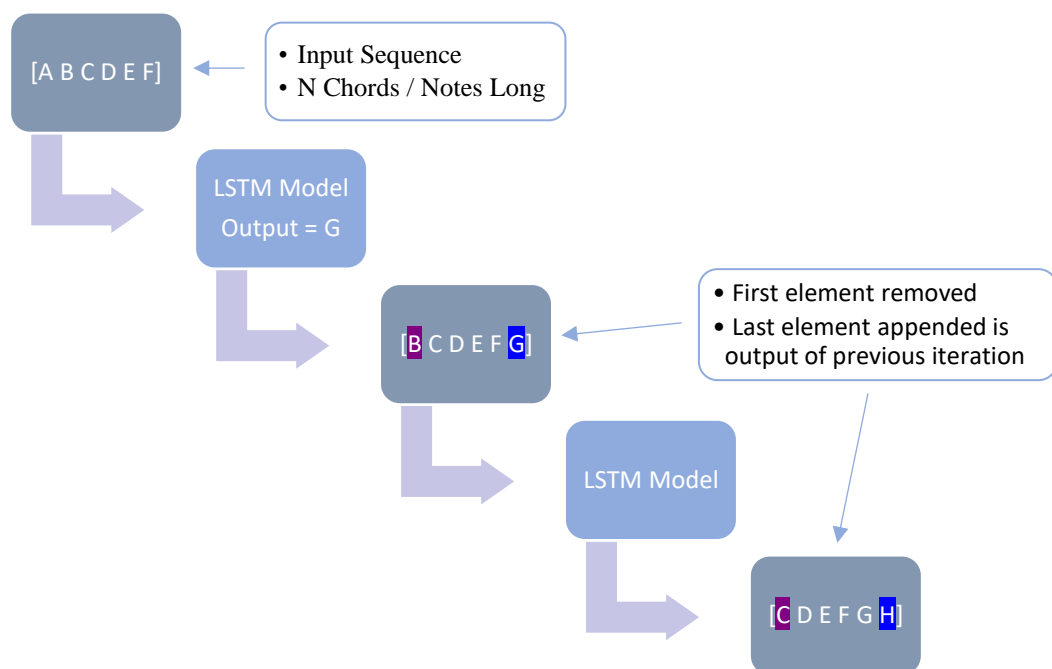


Figure 3: Algorithm for Generation of Chords
The algorithm shows how each subsequent chord in a sequence is synthesized

5.3.3 Creating Final Output

We will be using the output from the LSTM to create a MIDI file. This MIDI file will also then be synthesized in the style of the music instrument selected by the user at the frontend. The output of this instrument synthesis will be an MP3. Lastly, if the user requested the output in the form of sheet music, we will also convert the MIDI file to a PDF/Image with the composed piece drawn out as sheet music. All the outputs requested by the user will be compressed into one final ZIP file and returned to the client side.

Chapter 6: High-Level and Low-Level Design

The different aspects of our system are described in this chapter after analyzing various design solutions.

6.1 System Overview

Music is composed of musical objects known as notes having a relationship between each other to create a melody. There is an overall co-relation between the notes when a melody is created. Our webpage will be a simple interface created using ReactJS as frontend and python with Django to implement the backend.

Once the user uploads the mp3 file or recording, our frontend using an API, will provide the starting composition of at most 20 seconds which will then be completed and a composition of around 2 minutes will be returned.

6.2 Design Considerations

This section describes many of the issues which need to be addressed or resolved before attempting to devise a complete design solution.

6.2.1 Assumptions and Dependencies

Assumptions or dependencies regarding the software and its use, such as:

- Client connects from a suitable web browser.
- Client has adequate internet speed as to not hinder connectivity.
- The dataset used provides adequate results.

6.2.2 General Constraints

Following are the constraints that limit our system's working.

6.2.2.1 Network Constraint

A steady internet is required to be able to upload the mp3 or recording for processing and generation of a new composition.

6.2.2.2 Testing Constraint

Empirical evaluation will be done so evaluations will be time consuming. No definite function can be used for music evaluation.

6.2.2.3 File Format

Currently our system is only capable of dealing with mp3 file formats. Other formats are not supported as input method.

6.2.3 Goals and Guidelines

Goals, guidelines, principles, or priorities which dominate or embody the design of the system's software:

- The KISS principle ("Keep it simple stupid!") because we do not want to overcomplicate things that need to be simple.

- Trading off speed for memory use because we want the user to get a better result even if it takes some time to compute.

6.2.4 Development Methods

The development method for our project is evolutionary prototyping. We aim to develop a prototype which later after feedback will be refined and incorporated in our final prototype.

6.3 System Architecture

The system will be following a client – server architecture as shown in the following diagram.

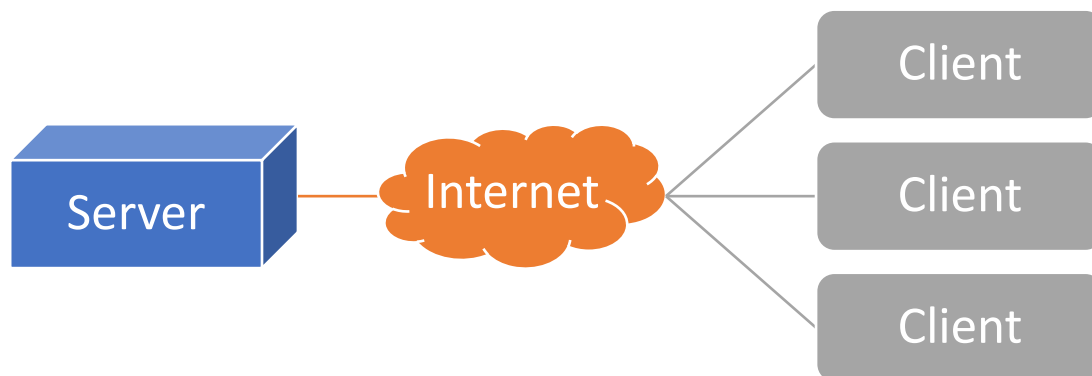


Figure 4: System Architecture
Client-Server Architecture to be followed in this project

6.4 Architectural Strategies

We will be focusing on the following architectural strategies to take our project to fruition.

- Python Language
- Extensible
- Metaphor UI Paradigm
- Client - Server Model

6.4.1 Python Language

We will be using Python because it makes Machine Learning Tasks easier to code and process.

6.4.2 Extensible

We plan to keep the system easily extensible in the future if we want to add the ability to process various kinds of genre.

6.4.3 Metaphor UI Paradigm

The selected UI Paradigm will be the Metaphor Paradigm which focuses on the developer designing the UI based on the assumption that an intuitive user will easily be able to figure out how to use it themselves

6.4.4 Client - Server Model

We will be using a Client – Server Model with the backend and frontend communicating via API requests. Backend will be Implemented in Python Django and Frontend will make use of ReactJS.

6.5 Domain Model/Class Diagram

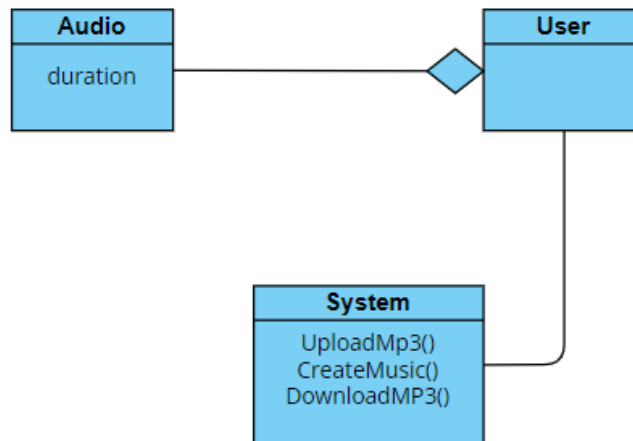


Figure 5: Class Diagram

*There will be 3 classes in the system
Audio, User and the System*

6.6 Sequence Diagrams

The entire working of our system can be visualized through a single sequence diagram.

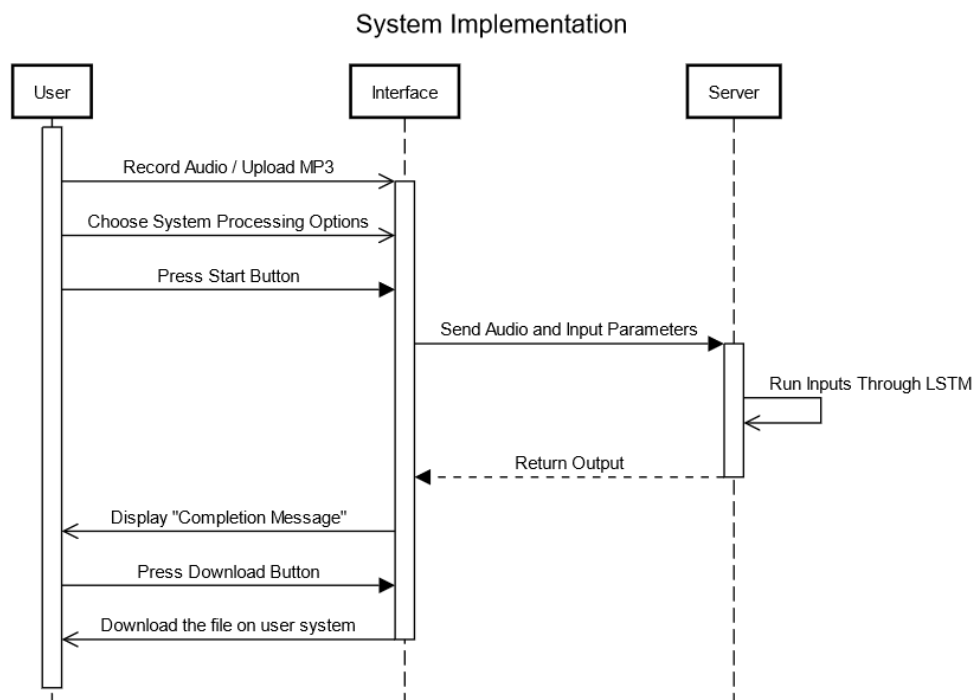


Figure 6: Sequence Diagram

The system's working can be summarized as shown

6.7 Policies and Tactics

This section will describe any design policies and/or tactics that do not have sweeping architectural implications (meaning they would not significantly affect the overall organization of the system and its high-level structures), but which nonetheless affect the details of the interface and/or implementation of various aspects of the system. Such decisions might concern (but are not limited to) things like the following:

- Using Python (with Django) to implement the Backend
- Using ReactJS for the Frontend
- Evolutionary Prototyping Design Paradigm
- Empirical Testing
- Single Static Webpage

6.7.1 Using Python (with Django) to implement the Backend

We had the option of implementing our backend in a language like C++ in combination with ASP.Net and C#. We chose against this because Python makes it easier to code and implement Machine Learning based projects due to its large collection of Libraries. Additionally, Python Django makes it possible to code a backend as well.

6.7.2 Using ReactJS for the Frontend

We will be implementing a frontend with ReactJS instead of with ASP.Net because the backend being in Python Django makes it easy to connect it to a React frontend.

6.7.3 Evolutionary Prototyping Design Paradigm

We are using the Evolutionary Prototyping Design Paradigm because our system will go through various stages of prototyping, each iteration building upon the previous one. This would be our preferred method instead of the Waterfall Method or perhaps even DevOps and Agile because we have a small team which will be working on the entire product together.

6.7.4 Empirical Testing

The results will be evaluated based on empirical measures and objective measures cannot be used because judging music is subjective. People can have varying tastes.

6.7.5 Single Static Webpage

The system will have a single webpage as opposed to multiple dynamic ones because there is very little user functionality needed on the frontend. Most development will be focused on the backend of the system.

Chapter 7: Implementation and Test Cases

This chapter will focus on providing the implementation details of the project along with the designed test cases intended to find out any bugs that may exist in the system.

7.1 Implementation

This section provides details of the algorithms that were implemented along with the platform and the APIs which were used. It is important to note that the implementation details provided are based on a prototype and do not represent a final product at this stage.

7.1.1 Dataset Selection

Piano MIDI files are a type of file containing instructions for a digital piano. They contain instructions to recreate the sound of a piano by playing different keys on the keyboard. The [dataset selected](#) is composed of classical piano midi files that contain compositions by 19 famous or well-known composers, scraped from <http://www.piano-midi.de>. It consists of 295 .MID files and the larger scope helps to show a wide view of the kind of data that can be found in these kinds of sources. We selected the dataset because it has a lot of different pieces, and our predictions can be tuned more specifically to the users' tastes. Furthermore, the dataset is a good representation of classical piano music compositions that we can use to train the music composition engine.

7.1.2 Data Loading

First, we generate a list of all the midi files in the music folder. We will be using the library Music21 to process these MIDI streams. The components (notes and chords) are then extracted from these MIDI file streams. The midi files only feature the piano indicated in the dataset. As a result, the file's components would be either piano chords or piano notes. We write a method to extract chords and notes from the data, resulting in a corpus.

Notes: Musical notes are the fundamental building blocks of music. It refers to a pitch connected with a certain audio vibration. Twelve musical notes are used in classical western music.

Chord: A set of notes that sound excellent together.

The Music21 stream extracted using this process comprises both chords and notes; we will extract them in the form of notes (by further breaking down chords into their corresponding notes). The result will be that we acquire a sequence of notes in the musical composition. This sequence of notes forms our data corpus.

7.1.3 Preprocessing

In this section we'll be completing the following data preprocessing steps.

7.1.3.1 Making a Dictionary

Making a dictionary to map the notes and their indices. We have the name of the note as a string in the corpus. These names are only symbols to the system and therefore we need to represent them using integers. Therefore, we develop a dictionary to assign a number to each unique note in our corpus. This dictionary will be used to get the integer value of a specific note during the training stage and vice versa during testing / forecasting stage. Henceforth, the dictionary will encode and decode the data entering and exiting the RNN + LSTM model.

7.1.3.2 Splitting and Encoding Corpus

Now we will be encoding and dividing the data corpus into equal-length sequences. The corpus currently contains notes. We will encode this corpus and generate smaller sequences of features and targets of similar length. Each feature and target will have a mapped index in the dictionary for the unique characters that they represent. After that, the features are reshaped and normalized, the targets, are one-hot encoded. The data is now ready to be delivered to the RNN for training.

7.1.3.3 Creating Train – Test Split

To assess how effectively our machine learning model works, we must divide a dataset into train and test sets. The statistics of the train set are known and are utilized to fit the model. The test data set, which is the second set, is utilized just for predictions. The train-test split is used to assess the performance of machine learning algorithms suitable for prediction-based algorithms (such as LSTM). This approach is a quick and simple procedure that allows us to compare our own model outcomes to machine results. The Test set is made up of 20% actual data while the Training set is made up of 80% actual data.

7.1.4 Model Architecture

Our model is built on the Recurrent Neural Network (RNN) architecture, which is a type of neural network that can remember previous inputs and is widely used for sequence analysis tasks. We also include the Long Short-Term Memory (LSTM) layer, which helps to avoid the vanishing gradient problem that arises in traditional RNNs.

Our model architecture consists of an input layer, an LSTM layer with 512 neurons, a dropout layer, and a dense output layer. The input layer accepts input sequences of encoded notes, while the LSTM layer processes the input sequences and remembers previous inputs. The dropout layer is added to prevent overfitting. Finally, the dense output layer produces the predicted note sequences.

7.1.5 Training

The model is trained using the Adam optimizer, which is a popular optimization algorithm for training neural networks. The loss function used is categorical cross-entropy, and the batch size is set to 128. We train the model for 50 epochs, and save the best model based on the validation loss.

7.1.6 Evaluation

To evaluate the model's performance, we use the test dataset to generate music compositions. We use a seed sequence of notes and let the model predict the next note in the sequence. This process is repeated until we reach the desired length of the composition. The generated compositions are then evaluated based on musical metrics such as note diversity and structure coherence.

7.1.7 Deployment

The model is deployed as a web application using Flask, a micro web framework written in Python. The web application allows users to input a seed sequence and generate a music composition based on the trained model. The generated compositions can be downloaded as MIDI files and the score can be downloaded as a PDF containing sheet music.

7.2 Test Case Design and Description

7.2.1 MIDI File Scrapping Test Case

Scrapping Component			
1			
Test Case ID:	1	QA Test Engineer:	Riva Nouman Khan
Test case Version:	1.0	Reviewed By:	Muhammad Maarij
Test Date:	2023-03-15	Use Case Reference(s):	Verifies if the web scraper is properly accessing MIID files hosted on freemidi.org and downloading them to the appropriate directory
Revision History:	-		
Objective	To verify that the MIDI files are being scraped and downloaded correctly from the specified website (https://freemidi.org/genre-XYZ) for the given genre.		
Product/Ver/Module:	TuneGen Music Composition Engine v1.0		
Environment:	<ul style="list-style-type: none">Operating System: Ubuntu 22.04.1 LTSPython v3.9.13Libraries:<ul style="list-style-type: none">beautifulsoup 4.11.1requests 2.28.1		
Assumptions:	<ul style="list-style-type: none">The internet connection is stable and the website is accessible.The specified genre exists on the website.The scraping function has been implemented correctly.		
Pre-Requisite:	None		
Step No.	Execution description	Procedure result	
1	Open the code editor and navigate to the test file	Passed	
2	Run the scraping function for the specified genre	Passed	
3	Check that the scraping function successfully extracts the download links for MIDI files	Passed	
4	Check that the number of extracted download links is greater than 0	Passed	
5	Check that the downloaded MIDI files are saved in the correct genre directory	Passed	
6	Check that the downloaded MIDI files have the correct file extension (.midi)	Passed	
7	Check that the downloaded MIDI files are not empty	Passed	
Comments: This test case ensures that the scraping function is working correctly and that the MIDI files are being scraped and downloaded successfully. If this test case passes, it indicates that the scraping function is correctly extracting the download links and saving the MIDI files in the correct directory. It also ensures that the downloaded MIDI files are valid and not empty.			
<div><input checked="" type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Not Executed</div>			

7.2.2 Dataset Loading and Preprocessing Test Case

Music Input Component			
1			
Test Case ID:	2	QA Test Engineer:	Riva Nouman Khan
Test case Version:	1.0	Reviewed By:	Muhammad Maarij
Test Date:	2023-03-15	Use Case Reference(s):	Verifies if the MIDI files are being properly read and broken down into their subcomponents
Revision History:	-		
Objective	To verify that the dataset is being loaded correctly and that the components are being extracted as expected.		
Product/Ver/Module:	TuneGen Music Composition Engine v1.0		
Environment:	<ul style="list-style-type: none">Operating System: Ubuntu 22.04.1 LTSPython v3.9.13Libraries:<ul style="list-style-type: none">tensorflow 2.11.0midi-to-dataframe 0.1		
Assumptions:	<ul style="list-style-type: none">The dataset has been downloaded and saved in the correct folder.The folder path for the dataset is properly configured in the code.		
Pre-Requisite:	<ul style="list-style-type: none">Data has been scraped and downloaded in the correct directory.		
Step No.	Execution description	Procedure result	
1	Open the code editor and navigate to the test file	Passed	
2	Run the dataset loading function.	Passed	
3	Check that the number of files loaded is equal to 295.	Passed	
4	Check that the components are being extracted as expected.	Passed	
Comments: This test case ensures that the dataset is being loaded correctly and the components are being extracted as expected. If this test case passes, it indicates that the dataset loading function is working correctly and that the components are being extracted as expected. Any errors in the code will be identified during this testing process.			
<div><input checked="" type="checkbox"/>Passed <input type="checkbox"/>Failed <input type="checkbox"/>Not Executed</div>			

7.2.3 Model Training Test Case

Data Training Component			
2			
Test Case ID:	3	QA Test Engineer:	Muhammad Maarij
Test case Version:	1.0	Reviewed By:	Riva Nouman Khan
Test Date:	2023-03-15	Use Case Reference(s):	Verifies if the model is being properly trained with acceptable levels of accuracy and loss
Revision History:	-		

Objective	<i>To verify that the model is being trained correctly and that it is able to predict notes accurately.</i>	
Product/Ver/Module:	<i>TuneGen Music Composition Engine v1.0</i>	
Environment:	<ul style="list-style-type: none"> • <i>Using Docker Container on Portainer</i> <ul style="list-style-type: none"> ○ <i>Operating System: Ubuntu 22.04.1 LTS</i> ○ <i>Python v3.9.13</i> ○ <i>Libraries:</i> <ul style="list-style-type: none"> ▪ <i>TensorFlow 2.11.0 with CUDA</i> ○ <i>GPU: NVIDIA Tesla P40</i> 	
Assumptions:	<ul style="list-style-type: none"> • <i>The dataset has been loaded correctly.</i> • <i>The dataset has been preprocessed and encoded correctly.</i> • <i>The model architecture has been defined and compiled correctly.</i> 	
Pre-Requisite:	<i>Dataset Loading and Preprocessing</i>	
Step No.	Execution description	Procedure result
1	<i>Open the code editor and navigate to the test file.</i>	<i>Passed</i>
2	<i>Run the model training function.</i>	<i>Passed</i>
3	<i>Check that the model has been trained for the expected number of epochs.</i>	<i>Passed</i>
4	<i>Check that the loss function has decreased over time.</i>	<i>Passed</i>
5	<i>Check that the model is able to predict notes accurately.</i>	<i>Passed</i>
Comments: This test case ensures that the model is being trained correctly and that it can predict notes accurately. If this test case passes, it indicates that the model training function is working correctly and that the model can predict notes accurately. Any errors in the code will be identified during this testing process.		
<input checked="" type="checkbox"/> <i>Passed</i> <input type="checkbox"/> <i>Failed</i> <input type="checkbox"/> <i>Not Executed</i>		

7.2.4 Music Composition Test Case

Music Composition Component			
3			
Test Case ID:	<i>4</i>	QA Test Engineer:	<i>Hadiya Kashif</i>
Test case Version:	<i>1.0</i>	Reviewed By:	<i>Muhammad Maarij</i>
Test Date:	<i>2023-03-15</i>	Use Case Reference(s):	<i>Verify the generated musical composition meets the user's preferences</i>
Revision History:	<i>-</i>		
Objective	<i>To verify that the music composition engine generates a musical composition that meets the user's preferences.</i>		
Product/Ver/Module:	<i>TuneGen Music Composition Engine v1.0</i>		
Environment:	<ul style="list-style-type: none"> • <i>Operating System: Ubuntu 22.04.1 LTS</i> • <i>Python v3.9.13</i> • <i>Libraries:</i> <ul style="list-style-type: none"> ○ <i>tensorflow 2.11.0</i> ○ <i>midi-to-dataframe 0.1</i> 		
Assumptions:	<ul style="list-style-type: none"> • <i>The model has been trained on a dataset of pop genre midi files.</i> • <i>The user has not provided any illegal inputs</i> 		

Pre-Requisite:		<ul style="list-style-type: none"> The user has provided valid inputs.
Step No.	Execution description	Procedure result
1	Set the user's preferred genre.	Passed.
2	Run the music composition engine on the provided inputs.	Passed.
3	Generate the musical composition output.	Passed.
4	Record and save the generated musical composition.	Passed.
5	Compress the composition into a ZIP file	Passed.
Comments: <ul style="list-style-type: none"> If the generated composition does not meet the user's preferences, the test case will fail. If the generated composition meets the user's preferences, the test case will pass. 		
<input checked="" type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Not Executed		

7.2.5 Sheet Music Output Test Case

Sheet Music Output Component			
4			
Test Case ID:	5	QA Test Engineer:	Hadiya Kashiif
Test case Version:	1.0	Reviewed By:	Muhammad Maarij
Test Date:	2023-03-15	Use Case Reference(s):	This test case verifies if the sheet music output is generated and saved as a PNG file.
Revision History:	-		
Objective	To verify if the sheet music output is generated and saved as a PNG file.		
Product/Ver/Module:	TuneGen Music Composition Engine v1.0		
Environment:	<ul style="list-style-type: none"> Operating System: Ubuntu 22.04.1 LTS Python v3.9.13 Libraries: <ul style="list-style-type: none"> tensorflow 2.11.0 midi-to-dataframe 0.1 Music21 8.1.0 		
Assumptions:	<ul style="list-style-type: none"> The Music Composition Engine is installed and running properly. The input file is a valid MIDI file. The output path is valid and has write permissions. The necessary libraries for generating and saving PNG files are installed. 		
Pre-Requisite:	<ul style="list-style-type: none"> The Music Composition Engine is running and has successfully generated a MIDI file. 		
Step No.	Execution description	Procedure result	
1	Set the output path to a valid folder with write permissions.	The output path is set successfully.	

2	Set the filename of the output file to a valid name with ".png" extension.	The filename is set successfully.
3	Invoke the function to generate sheet music output as a PNG file by passing the MIDI file path and output file path as arguments.	The function is invoked successfully without any errors.
4	Check if the output file exists in the specified output path with the specified filename.	The output file exists and is a valid PNG file.
5	Open the output file and visually verify if the sheet music output is generated as expected.	The sheet music output is generated and saved as a PNG file as expected.
6	Delete the output file from the specified output path.	The output file is deleted successfully.
Comments: The test case verifies if the sheet music output is generated and saved as a PNG file. It assumes that the input MIDI file is valid, and the necessary software and hardware dependencies are met. The test case passes if the output file is generated and saved as a valid PNG file and the sheet music output is visually verified as expected.		
<input checked="" type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Not Executed		

7.2.6 Upload, Record and Play Music Test Case

Upload/Record/Play Component			
4			
Test Case ID:	6	QA Test Engineer:	Hadiya Kashiif
Test case Version:	1.0	Reviewed By:	Muhammad Maarij
Test Date:	2023-03-15	Use Case Reference(s):	This test case verifies if we are able to correctly upload/record audio files in our frontend
Revision History:	-		
Objective	To verify that the file upload functionality is working correctly and that the uploaded file is processed as expected.		
Product/Ver/Module:	TuneGen Music Composition Engine v1.0		
Environment:	<ul style="list-style-type: none"> Operating System: Windows 10 21H2 Browser: Firefox 113.0.2 HTML + CSS + JavaScript 		
Assumptions:	<ul style="list-style-type: none"> The website is accessible. 		
Pre-Requisite:	None		
Step No.	Execution description	Procedure result	
1	Open the website in the browser	Passed	
2	Click on the "Upload MP3" button	Passed	
3	Select an MP3 file from the local file system	Passed	
4	Verify that the selected file is displayed correctly	Passed	
5	Click on the "Record Audio" button	Passed	

6	Verify that the audio recording functionality is working correctly	Passed
7	Click on the "Upload MP3" button again	Passed
8	Select a different MP3 file from the local file system	Passed
9	Verify that the newly selected file is displayed correctly	Failed
10	Verify that the previously uploaded file is replaced by the new file	Passed
11	Click the "Play" button to make sure the file Plays	Passed
12	Click on the "Record Audio" button again	Passed
13	Verify that the previously recorded audio is replaced by the new recording	Passed
14	Click the "Play" button to make sure the file Plays	Passed
Comments: This test case ensures that the file upload functionality is working correctly and that the uploaded file is processed as expected. It also verifies that the selected file is displayed correctly and that the previously uploaded/recorded files are replaced when new files/recordings are selected. Additionally, it checks that canceling the file selection or audio recording does not result in unwanted changes. Lastly, it checks if the selected/recorded files play as expected.		
<input checked="" type="checkbox"/> Passed <input type="checkbox"/> Failed <input type="checkbox"/> Not Executed		

7.2.7 Download Generated Music Test Case

Download Component			
4			
Test Case ID:	7	QA Test Engineer:	Hadiya Kashiif
Test case Version:	1.0	Reviewed By:	Muhammad Maarij
Test Date:	2023-03-15	Use Case Reference(s):	This test case verifies if we can correctly download the ZIP file.
Revision History:	-		
Objective	To verify that the ZIP file download functionality is working correctly and that the downloaded ZIP file contains the expected files.		
Product/Ver/Module:	TuneGen Music Composition Engine v1.0		
Environment:	<ul style="list-style-type: none"> Operating System: Windows 10 21H2 Browser: Firefox 113.0.2 HTML + CSS + JavaScript 		
Assumptions:	<ul style="list-style-type: none"> The website is accessible. 		
Pre-Requisite:	<ul style="list-style-type: none"> The user has already uploaded/recorded an MP3 file and selected desired options. 		
Step No.	Execution description	Procedure result	
1	Open the website in the browser	Passed	

2	Upload an MP3 file	Passed
3	Select desired options for processing	Failed
4	Click on the "Download ZIP" button	Failed
5	Verify that the ZIP file download starts	Failed
6	Wait for the download to complete	Failed
7	Locate the downloaded ZIP file in the local file system	Failed
8	Extract the contents of the ZIP file	Failed
9	Verify that the extracted files are in the expected formats (PDF, MIDI, MP3)	Failed
Comments: This test case ensures that the ZIP file download functionality is working correctly and that the downloaded ZIP file contains the expected files. It also verifies that the extracted files match the selected options and are in the expected formats. If this test case passes, it indicates that the download functionality is functioning as expected and providing the user with the correct output files.		
<input type="checkbox"/> Passed <input checked="" type="checkbox"/> Failed <input type="checkbox"/> Not Executed		

7.3 Test Metrics

7.3.1 Test Case Metrics

Metric	Value
Number of Test Cases	7
Number of Test Cases Passed	6
Number of Test Cases Failed	1
Test Case Defect Density	$\frac{\text{No of test cases failed} \times 100}{\text{No of test cases executed}} = \frac{1 \times 100}{7}$ $= 14.3\%$
Test Case Effectiveness	$\frac{\text{No of defects found with test cases} \times 100}{\text{No of total defects found}} = \frac{1 \times 100}{8}$ $= 12.5\%$
Traceability Matrix	-

7.4 Conclusion

As part of the chapter, while implementing our team utilized Python as the primary programming language along with various libraries such as Music21, Keras, TensorFlow, Matplotlib, and NumPy to aid in the music generation process. To deploy the service, we opted for a static webpage as the frontend interface. We generated a customized dataset for our needs, which was genre-based, and we collected it through web scraping techniques since no existing dataset fulfilled our requirements. The user's input audio is fed into an RNN LSTM, which is trained on the accumulated data. We thoroughly tested an AI-based music generation system

and verified several critical aspects of the system's reliability and accuracy. We performed several test cases, including verifying the accuracy of data loading, component extraction, model training, and the ability to generate musical compositions that meet the user's preferences. We also ensured that the music composition engine could generate compositions of different genres and incorporate the user's preferred tempo, rhythm, and melody. Furthermore, we tested the system's ability to predict notes accurately and generate compositions that accurately reflected the user's preferences. Overall, these tests ensured that the AI-based music generation system was reliable, accurate, and efficient, and could generate high-quality musical compositions based on the user's preferences. Our chapter's findings can provide useful insights for researchers and developers working on AI-based music generation systems, and our test cases can serve as a guide for testing similar systems.

Chapter 8: User Manual

The following is what the final implemented user interface of our application looks like.

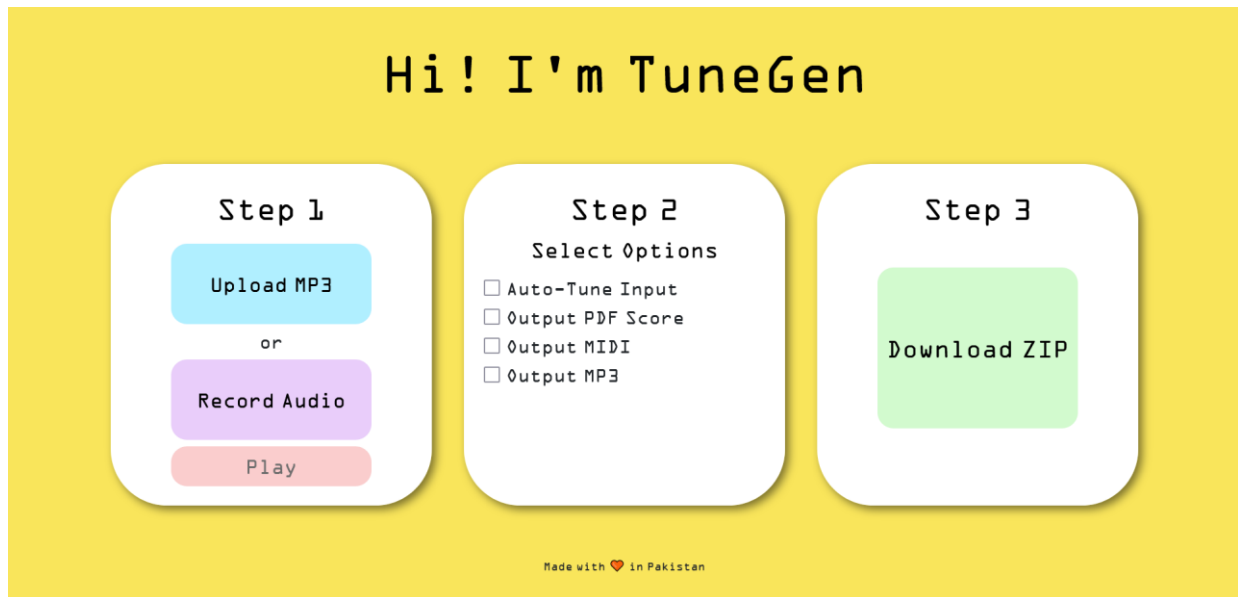


Figure 7: Final User Interface

The final user interface that has been developed

8.1 Step 1: Upload or Record Audio

Start by clicking on either the "Upload MP3" button or the "Record Audio" button, depending on your preference. If you choose to upload an MP3 file, simply select the file from your computer's directory. If you choose to record audio, allow the website to access your microphone, and then start recording. In both cases the user can preview the uploaded/recorded audio by clicking on the "Play" button.

8.2 Step 2: Select Options

After you've uploaded or recorded your audio, select the options you want. You can choose to auto-tune the input, output the PDF score, output MIDI, and/or output an MP3 file. Simply check the box next to each option you want.

8.3 Step 3: Download ZIP

Once you've selected your options, click the "Download ZIP" button to start the conversion process. After a few minutes, your converted files will be ready for download in a ZIP folder.

Chapter 9: Experimental Results and Discussion

The aim of this study was to develop a music composition engine capable of generating an extensive variety of original music pieces across different genres, with a particular emphasis on genre-based models. The engine is designed to accept input parameters and generate unique music compositions accordingly. This chapter presents the experimental results and discussions, with a focus on evaluating the performance of the model.

To evaluate the performance of our music composition engine, we trained an RNN LSTM model using the Pop music scraped from www.freemidi.org. The model underwent training for 150 epochs, during which it achieved an impressive accuracy of 0.97281 and a low loss value of 0.08916.

These results demonstrate the effectiveness of our model in capturing the essence of Pop music and generating compositions that align with the genre's characteristics. The high accuracy indicates that the model successfully learned the patterns and structures inherent in the training data, enabling it to generate coherent and stylistically consistent music compositions.

Moreover, the low loss value indicates that the model was able to minimize the discrepancies between the generated compositions and the original training data. This suggests that the music compositions produced by our engine closely resemble the examples provided in the dataset, further validating the model's performance.

By achieving such promising results, our open-source music composition engine presents a significant contribution to the field. It builds upon existing open-source projects while providing users with the unique capability to select a seed tune, enabling greater user customization and creative exploration.

In summary, the experimental results indicate that our RNN LSTM model trained on the Pop music dataset exhibits high accuracy and low loss values. These findings emphasize the performance of our music composition engine and support its effectiveness in generating original music compositions across various genres.

Chapter 10: Conclusion and Future Work

Our team has made significant progress in the development of a music generation system, focusing specifically on the Pop genre. Through an extensive literature review and comparative analysis, we determined that utilizing RNN and LSTM models would be the most effective approach. We trained an LSTM model using a dataset obtained from freemidi.org, where we scraped and processed MIDI files to extract notes, forming a tailored corpus. With this approach, we successfully created a music composition engine capable of generating original compositions based on user input.

To facilitate the development process, we employed Python as our primary programming language, leveraging key libraries such as midi-to-df, Music21, Keras, TensorFlow, Matplotlib, and NumPy. Our front-end interface was implemented as a static webpage, providing users with the ability to upload MP3 clips or hum melodies for input and receive output in the form of an MP3 file, PDF sheet music, and a MIDI file.

Moving forward, our future work revolves around improving and expanding our music generation system. Firstly, we plan to enhance our model's training by incorporating a larger and more diverse dataset. By expanding our data sources and including various genres of music from freemidi.org, we aim to improve the accuracy and variety of the music compositions generated by our engine. This expansion will allow us to cater to a wider audience with different musical preferences.

Furthermore, we are committed to optimizing the front-end of our web interface to ensure an intuitive and seamless user experience. We understand the importance of a user-friendly interface that allows easy interaction with our music composition engine. By refining the interface and incorporating user feedback, we aim to create a system that is efficient, reliable, and enjoyable to use.

In addition to our efforts in diversifying the dataset beyond genres, we are also exploring opportunities to incorporate a broader range of musical styles. Our goal is to produce inclusive and appealing music that transcends conventional genre boundaries, appealing to a more diverse audience. This endeavor presents exciting possibilities for us to explore innovative methods of generating music and push the boundaries of music composition.

In conclusion, we have made substantial progress in developing a music generation system focused on the Pop genre. By leveraging RNN and LSTM models, along with a specifically tailored dataset scraped from freemidi.org, we have successfully created a music composition engine. Our future work entails expanding our dataset, optimizing the front-end interface, and incorporating diverse musical styles to create a more robust and inclusive music generation system. We remain dedicated to advancing the field of music composition and providing users with a unique and satisfying musical experience.

References

- [1] A. Alpern, "Techniques for Algorithmic Composition of Music", vol. 95, p. 120, 1995. Available: <http://hamp.hampshire.edu/adaF92/algocomp/algocomp>. [Accessed 16 September 2022].
- [2] Burns and Kristine, "Algorithmic composition, a definition", *Florida International University*, 1997. Available: <http://music.dartmouth.edu/~wowem/hardware/algorithmdefinition.html>. [Accessed 16 September 2022]
- [3] J. Morgan, "5 Reasons Why The Cloud Is Environmentally Friendly", *Missioncloud.com*, 2022. [Online]. Available: <https://www.missioncloud.com/blog/5-reasons-why-the-cloud-is-environmentally-friendly>. [Accessed: 07- Oct- 2022].
- [4] M. Schmidt, "How to Generate Music with AI", Rootstrap, 2022. [Online]. Available: <https://www.rootstrap.com/blog/how-to-generate-music-with-ai/>. [Accessed: 11- Oct- 2022].
- [5] J. R. Beltran and C. Hernandez-Olivan, "MUSIC COMPOSITION WITH DEEP LEARNING: A REVIEW", *Arxiv.org*, 2022. [Online]. Available: <https://arxiv.org/pdf/2108.12290.pdf>. [Accessed: 11- Oct- 2022].
- [6] H. Hu, "Research on the Interaction of Genetic Algorithm in Assisted Composition", *Downloads.hindawi.com*, 2022. [Online]. Available: <https://downloads.hindawi.com/journals/cin/2021/3137666.pdf>. [Accessed: 11- Oct- 2022].
- [7] P. Agrawal, S. Kaushik, S. Banga, N. Pathak and S. Goel, "Automated Music Generation using LSTM", 2022. [Online]. Available: https://www.researchgate.net/profile/Piyush-Agrawal-9/publication/332182816_Automated_Music_Composition_using_LSTM/links/5ca50f21a6fdcc12ee912400/Automated-Music-Composition-using-LSTM.pdf. [Accessed: 11- Oct- 2022].
- [8] J. Briot and F. Pachet, "Deep learning for music generation: challenges and directions", *Neural Computing and Applications*, vol. 32, no. 4, pp. 981-993, 2018. Available: 10.1007/s00521-018-3813-6 [Accessed 11 October 2022].
- [9] R. Whorley and D. Conklin, "Music Generation from Statistical Models of Harmony", *Journal of New Music Research*, vol. 45, no. 2, pp. 160-183, 2016. Available: 10.1080/09298215.2016.1173708 [Accessed 11 October 2022].
- [10] J. Fernandez and F. Vico, "AI Methods in Algorithmic Composition: A Comprehensive Survey", *Journal of Artificial Intelligence Research*, vol. 48, pp. 513-582, 2013. Available: 10.1613/jair.3908 [Accessed 8 October 2022].
- [11] I. Tham and M. Kim, "MusicGenDL/CIS 522 Final Report Github.pdf at main · thamsuppp/MusicGenDL", *GitHub*, 2021. [Online]. Available: <https://github.com/thamsuppp/MusicGenDL/blob/main/CIS%20522%20Final%20Report%20Github.pdf>. [Accessed: 11- Oct- 2022].
- [12] J. Maurer, "The History of Algorithmic Composition", *Ccrma.stanford.edu*, 1999. [Online]. Available: <https://ccrma.stanford.edu/~blackrse/algorithm.html>. [Accessed: 08- Oct- 2022].

- [13] M. Simoni, "Algorithmic Composition: A Gentle Introduction to Music Composition Using Common LISP and Common Music", 2003. Available: 10.3998/spobooks.bbv9810.0001.001 [Accessed 11 October 2022].
- [14] B Dunnett, "Canon Music - Music Theory Academy", Music Theory Academy, 2022. [Online]. Available: <https://www.musictheoryacademy.com/understanding-music/canon-music/>. [Accessed: 11- Oct- 2022].
- [15] G. Korvel and B. Kostek, "Discovering rule-based learning systems for the purpose of music analysis", Proceedings of Meetings on Acoustics, 2019. Available: 10.1121/2.0001221 [Accessed 11 October 2022].
- [16] R. Sugandhi, R. Kulkarni, R. Gaikwad, P. Kulkarni and S. Kone, "Survey on Deep Learning in Music using GAN", IJERT, vol. 8, no. 9, 2019. Available: <https://www.ijert.org/survey-on-deep-learning-in-music-using-gan>. [Accessed 11 October 2022].
- [17] Y. Yu, A. Srivastava and S. Canales, "Conditional LSTM-GAN for Melody Generation from Lyrics", ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 17, no. 1, pp. 1-20, 2021. Available: 10.1145/3424116 [Accessed 11 October 2022].
- [18] R. Levi, A field investigation of the composing processes used by second-grade children creating original language and music pieces. Case Western Reserve University, 1991.
- [19] C. Walton, Basic Forms in Music. Alfred Music, 2005.
- [20] J. Titon, Worlds of music. Belmont, Calif.: Schirmer/Thomson Learning, 2009.
- [21] J. Moorer, "Music and computer composition", Communications of the ACM, vol. 15, no. 2, pp. 104-113, 1972. Available: 10.1145/361254.361265 [Accessed 8 October 2022].
- [22] P. Prusinkiewicz, "Score generation with L-systems", ICMC, pp. 455-457, 2022. Available: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.324.3009&rep=rep1&type=pdf>. [Accessed 8 October 2022].
- [23] D. Cope, "Computer Modeling of Musical Intelligence in EMI", Computer Music Journal, vol. 16, no. 2, p. 69, 1992. Available: 10.2307/3680717 [Accessed 8 October 2022].
- [24] G. Wiggins, "Computer Models of Musical Creativity: A Review of Computer Models of Musical Creativity by David Cope", Literary and Linguistic Computing, vol. 23, no. 1, pp. 109-116, 2007. Available: 10.1093/llc/fqm025 [Accessed 8 October 2022].
- [25] C. Manning, A. Ng, C. Lin and R. Socher, "Parsing natural scenes and natural language with recursive neural networks", Proceedings of the 28th International Conference on International Conference on Machine Learning, pp. 129–136, 2011. Available: 10.5555/3104482.3104499 [Accessed 9 October 2022].
- [26] S. Schwanauer and D. Levitt, Machine models of music. Cambridge, Mass.: The MIT Press, 1993, pp. 511-532.
- [27] M. Marques, V. Oliveira, S. Vieira and A. Rosa, "Music composition using genetic evolutionary algorithms", Proceedings of the 2000 Congress on Evolutionary Computation. CEC00 (Cat. No.00TH8512). Available: 10.1109/cec.2000.870368 [Accessed 9 October 2022].

- [28] M. Johnson, D. Tauritz and R. Wilkerson, "Evolutionary Computation Applied to Melody Generation", 2004. Available: <https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.125.6788&rep=rep1&type=pdf>. [Accessed 9 October 2022].
- [29] D. Birchfield, "Evolving intelligent musical materials", Columbia University, 2003. Available: <https://www.proquest.com/openview/2ab963432e55982470d4452317968c99/1?pq-origsite=gscholar&cbl=18750&diss=y>. [Accessed 9 October 2022].
- [30] I. Goodfellow et al., "Generative adversarial networks", Communications of the ACM, vol. 63, no. 11, pp. 139-144, 2020. Available: 10.1145/3422622 [Accessed 11 October 2022].
- [31] R. Awati, "What are convolutional neural networks?", SearchEnterpriseAI, 2022. [Online]. Available: <https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network>. [Accessed: 11- Oct- 2022].
- [32] M. Kim and I. Tham, "Generating Music Using Deep Learning", University of Pennsylvania, 2021. Available: <https://towardsdatascience.com/generating-music-using-deep-learning-cb5843a9d55e>. [Accessed 11 October 2022].
- [33] S. Tanberk and D. Tükel, "Style-Specific Turkish Pop Music Composition with CNN and LSTM Network", 2021 IEEE 19th World Symposium on Applied Machine Intelligence and Informatics (SAMI), 2021. Available: 10.1109/sami50585.2021.9378654 [Accessed 11 October 2022].