МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Южно-Уральский государственный университет**

**(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

|  |  |
| --- | --- |
| РАБОТА ПРОВЕРЕНА  Рецензент  <ученая степень, ученое звание>  \_\_\_\_\_\_\_ <И.О. Фамилия рецензента>  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. | ДОПУСТИТЬ К ЗАЩИТЕ  Заведующий кафедрой, д.ф.-м.н., профессор  \_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

**Разработка веб-приложения для генерации музыки**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ЮУрГУ – 09.04.04.2024.308-1492.ВКР

|  |  |
| --- | --- |
|  | Научный руководитель,  доцент кафедры СП, к.ф.-м.н., доцент \_\_\_\_\_\_\_\_\_\_ Г.И. Радченко  Автор работы, студент группы КЭ-229  \_\_\_\_\_\_\_\_\_\_ Д.В. Кутюшкин  Ученый секретарь  (нормоконтролер)  \_\_\_\_\_\_\_\_\_\_\_\_\_ И.Д. Володченко  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение  
высшего образования

**«Южно-Уральский государственный университет**

**(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук**

**Кафедра системного программирования**

УТВЕРЖДАЮ

Зав. кафедрой СП

\_\_\_\_\_\_\_\_\_\_ Л.Б. Соколинский

29.01.2024 г.

**ЗАДАНИЕ**

**на выполнение выпускной квалификационной работы магистранта**

студенту группы КЭ-229

Кутюшкину Дмитрию Владимировичу,

обучающемуся по направлению

09.04.04 «Программная инженерия»

(магистерская программа «Искусственный интеллект и инженерия данных»)

1. **Тема работы** (утверждена приказом ректора от \_\_.\_\_.2024 г. № \_\_\_)

Разработка веб-приложения для генерации музыки.

1. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
2. **Исходные данные к работе2**
3. Kailash A. Generative Adversarial Networks Projects: Build next-generation generative models using TensorFlow and Keras. – Packt Publishing Ltd., 2019. – 316 p.
4. Foster D. Generative Deep Learning, 2nd Edition. – O'Reilly Media, Inc., 2023. – 426 p.
5. **Перечень подлежащих разработке вопросов**
6. Провести анализ предметной области.
7. Подготовить набор данных.
8. Выбрать топологию нейронной сети.
9. Реализовать нейросетевую модель и провести ее обучение.
10. Спроектировать приложение для генерации музыки.
11. Реализовать приложение.
12. Провести тестирование приложения.
13. **Дата выдачи задания:** 29.01.2024 г.

**Научный руководитель,**

доцент кафедры СП, к.ф.-м.н., доцент Г.И. Радченко

**Задание принял к исполнению** Д.В. Кутюшкин

ОГЛАВЛЕНИЕ

[ВВЕДЕНИЕ 5](#_Toc165915746)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7](#_Toc165915747)

[1.1. Анализ аналогичных проектов 7](#_Toc165915748)

[1.2. Анализ архитектур нейронных сетей 9](#_Toc165915749)

[2. ПОДГОТОВКА НАБОРА ДАННЫХ 19](#_Toc165915750)

[2.1. Формат входных данных 19](#_Toc165915751)

[2.2. Описание набора данных 20](#_Toc165915752)

[2.3. Предобработка данных 21](#_Toc165915753)

[3. РЕАЛИЗАЦИЯ НЕЙРОСЕТЕВОЙ МОДЕЛИ 23](#_Toc165915754)

[3.1. Архитектура нейронной сети 23](#_Toc165915755)

[3.2. Анализ обучения модели 27](#_Toc165915756)

[4. РАЗРАБОТКА ПРИЛОЖЕНИЯ 29](#_Toc165915757)

[4.1. Проектирование 29](#_Toc165915758)

[4.2. Реализация приложения 31](#_Toc165915759)

[ЗАКЛЮЧЕНИЕ 32](#_Toc165915760)

[ЛИТЕРАТУРА 33](#_Toc165915761)

[ПРИЛОЖЕНИЕ 35](#_Toc165915762)

# ВВЕДЕНИЕ

**Актуальность**

В настоящее время ежемесячно появляется огромное множество разнообразного программного обеспечения, основанного на технологиях искусственного интеллекта. Эти технологии созданы для решения самых разных задач.

В современном музыкальном мире, где искусственный интеллект и машинное обучение играют все более значимую роль, генерация мелодии становится все более актуальной и инновационной темой исследования, которая может иметь значительное влияние на музыкальную индустрию и творческий процесс.

Тема генерации мелодии упрощает работу с таким важным аспектом музыкальной индустрии, как композиция музыки. Одними из ключевых проблем, с которыми сталкиваются музыканты являются сложность и затраты времени на придумывание интересного и запоминающегося мотива, особенно в ситуациях, когда появляется необходимость развивать идеи уже начатых произведений. Существование инструмента генерации мелодий может открыть больше возможностей для творчества. Написание мелодии – процесс, зачастую зависящий от вдохновения и настроен. Генератор мелодий в свою очередь мог бы помочь в поиске идей, продолжении уже существующих идей, и просто помог бы ускорить и оптимизировать процесс. Подобный инструмент также поможет начинающим музыкантам развивать свои навыки.

Эта технология также имеет потенциал для творческого экспериментирования и инноваций. Благодаря случайности в процессе генерации музыканты могут открывать новые жанры, настроения и стили, что способствует разнообразию и оригинальности в музыкальном творчестве.

Все эти факторы подтверждают актуальность и значимость исследования генерации мелодии. Этот инновационный подход не только упрощает и оптимизирует процесс создания музыки, но и стимулирует разнообразие, творческое экспериментирование и эксклюзивность в музыкальной индустрии.

**Постановка задачи**

Целью выпускной квалификационной работы является разработка веб-приложения для генерации музыки. Для достижения поставленной цели необходимо решить следующие задачи:

1. провести анализ предметной области;
2. сформировать набор данных для обучения;
3. выбрать топологию нейросети;
4. реализовать нейросетевую модель и провести ее обучение;
5. спроектировать приложение для генерации музыки;
6. реализовать приложение;
7. провести тестирование приложения.

**Структура и содержание работы**

Работа состоит из введения, двух глав, заключения и списка литературы. Объем работы составляет 28 страниц, объем списка литературы – 12 источников.

В первой главе описываются существующие аналоги, а также производится анализ архитектур нейронных сетей, применимых в задаче генерации звука.

Вторая глава посвящена формату входных данных, описанию использованного при обучении набора данных и его обработке.

В третьей главе содержится описание архитектуры нейросетевой модели, и эксперименты по подбору оптимальных параметров и структуры.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Анализ аналогичных проектов

На данный момент тема относительно новая, и задача генерации музыки не очень часто затрагивается в статьях, а какие-либо проекты только начинают появляться.

Рассмотрим несколько наиболее актуальных и современных решений данной задачи.

**Jukebox**

Проект разработан исследователями из OpenAI, он предназначен для генерации полноценных музыкальных композиций, включая мелодии, гармонии, ритмы и тексты. Отличительной особенностью является его способность генерировать как инструментальные, так и вокальные партии.

Jukebox стремится к созданию новых и уникальных произведений, которые звучат аутентично и имеют характеристики определенного жанра.

Система реализована с помощью автоэнкодера, который сжимает музыку в дискретное пространство с помощью модели VQ-VAE. Эта модель основана на квантизации. Иерархическая VQ-VAE способна к генерации коротких инструментальных паттернов аудио из небольшого набора музыкальных инструментов.

Исследователи используют три уровня сжатия, которые сжимают аудиозапись в 8, 32 и 128 раз соответственно. В результате таких преобразований данные теряют большую часть деталей, но сохраняют информацию о тоне, тембре и громкости мелодии.

Модель обучали на собранном из интернета наборе из 1,2 миллионов композиций. Половина из них на английском языке. Для каждой аудиозаписи доступны метаданные и текст песни из LyricWiki [8].

**MuseNet**

Созданная исследователями из OpenAI MuseNet способна генерировать полноценные музыкальные композиции с использованием 10 различных инструментов и сочетать стили. Сеть научилась обнаруживать закономерности гармонии, ритма и стиля и предсказывать следующий токен. MuseNet использует ту же модель, что и GPT-2 – трансформер, обученный предсказывать следующий токен последовательности, будь то аудио или текст [9].

В качестве набора данных использовались коллекции MIDI-файлов, пожертвованные ClassicalArchives и BitMidi, дополнительно использовался набор данных MAESTRO.

**MusicLM**

Это модель, разработанная исследователями из Google. Модель предназначена для генерации музыки по текстовому описанию. Для решения задачи использовался decoder-only Transformer, состоящий из 24 слоев и 16 слоев внимания. Обучение производилось на наборе данных Free Music Archive [11].

**Нейромузыка**

Это модель, разработанная исследователями из Yandex, включенная в сервис Yandex Music для генерации бесконечного аудиопотока определенного настроения и жанра.

Для решения задачи использовались трансформеры. Модель обучалась на триплетах (басовая, гармоническая и мелодическая петли), закодированных в текстовой форме, впоследствии научившись генерировать новые петли и продолжать существующие [10].

1.2. Анализ архитектур нейронных сетей

Если упростить задачу, генерация аудио сводится к предсказанию последовательности по ее уже имеющимся элементам, последовательность может являться как существующим аудио фрагментом, так и случайно сгенерированным шумом. В случае генерации мелодий, задача упрощается, т.к. последовательность нот имеет гораздо меньше возможных значений элементов, чем абстрактный звуковой сигнал.

В задаче обработки последовательностей хорошо себя зарекомендовали рекуррентные нейронные сети (RNN) и различные их архитектуры, например, LSTM (Long-short term memory) и GRU (Gated Recurrent Units), чьи способности улавливать предыдущий контекст сыграли в этом главную роль.

Также для решения задачи подходят более сложные архитектуры, такие как трансформеры, генеративно-состязательные сети (GAN), которые показывают более хорошие результаты.

Достойны упоминания и другие подходы искусственного интеллекта, такие как обучение с подкреплением.

**Рекуррентные нейронные сети**

Рекуррентная нейронная сеть (Recurrent Neural Network, RNN) использует принцип обратной связи, подавая на вход рекуррентного слоя его выход в предыдущий момент времени [1]. Она обрабатывает последовательность, перебирая ее элементы и сохраняя состояние, полученное при обработке предыдущих элементов. Однако простейшая рекуррентная сеть обладает таким недостатком как «забывание» информации через длительное время.

Для решения этой проблемы была придумана архитектура LSTM, ее схема представлена на рисунке 1.

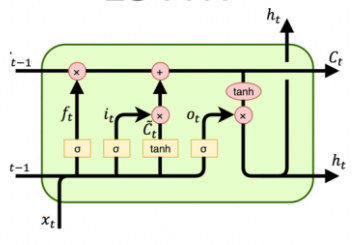


Рисунок 1 – Архитектура LSTM

Основное отличие заключается в реализации механизма долгосрочной памяти, таким образом информация из ранних интервалов может использоваться на поздних. За реализацию этого механизма на данной схеме отвечает выходной сигнал , имитация забывания ненужной информации реализована операцией поэлементного умножения, а запоминание новой информации операцией поэлементного сложения [1].

Существует упрощенная версия архитектуры LSTM – архитектура GRU (Gated Recurrent Unit), которая схожа с сетью LSTM, с тем отличием, что механизм памяти и выход (скрытое состояние) реализованы в одном сигнале, это уменьшает число параметров, тем самым уменьшая вычислительную сложность. Ее схема представлена на рисунке 2.

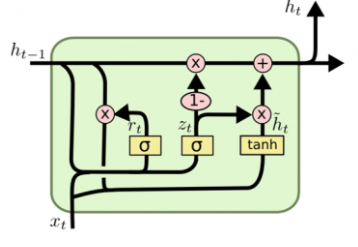


Рисунок 2 – Архитектура GRU

В статье [2] исследователи используют один блок GRU и один блок LSTM. В качестве входных данных в статье используются аудиосэмплы, представленные в виде фаз и амплитуд частот. После генерации скрытого состояния используется афинное преобразование для преобразования его в представление частоты. Затем оценивается ошибка L2 для истинного и сгенерированного значения. В результате получается сеть, обученная предсказывать следующие элементы последовательности.

Для генерации выполняется прямое распространение на исходной последовательности, в результате получается вектор предсказанных следующих элементов последовательности. Затем этот вектор используется как исходный для генерации новых элементов, и снова выполняется прямой проход до тех пор, пока не будет сгенерирована последовательность необходимой длины.

В результате экспериментов с генерацией было выяснено, что более крупные начальные последовательности дают лучшие результаты, число итераций также зависит от длины начальной последовательности. Эмпирически, генерация последовательности, которая примерно в три раза больше исходной имеет тенденцию генерировать связную музыку, которая не приводит к проблеме петель (случаев, когда предыдущее скрытое состояние похоже на текущее, а все последующие сохраняют эту тенденцию).

По генерируемым результатам архитектуры GRU и LSTM сильно отличались: GRU показала неудовлетворительные результаты, генерируя в основном белый шум, в то время как LSTM смогла воспроизводить вполне правдоподобную музыку. Таким образом сеть LSTM намного лучше справляется с задачей.

В статье [3] используется слегка иной подход, сеть LSTM прогнозирует не только саму частоту, но и время, в которое она воспроизводится. На рисунке 3 изображена архитектура модели.

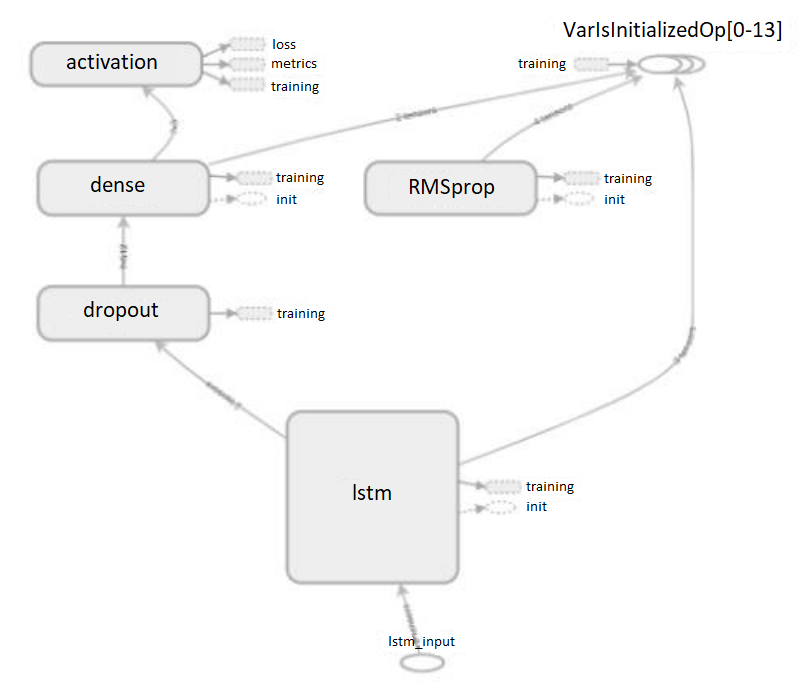


Рисунок 3 – Модель генерации аудио

**Трансформеры**

Трансформер – архитектура нейронных сетей, избегающая рекуррентности и полностью полагающаяся на механизм внимания для построения зависимостей между входом и выходом [4]. Основное преимущество данной архитектуры по сравнению с RNN – высокая эффективность в условиях параллелизации. Архитектура представлена на рисунке 4.

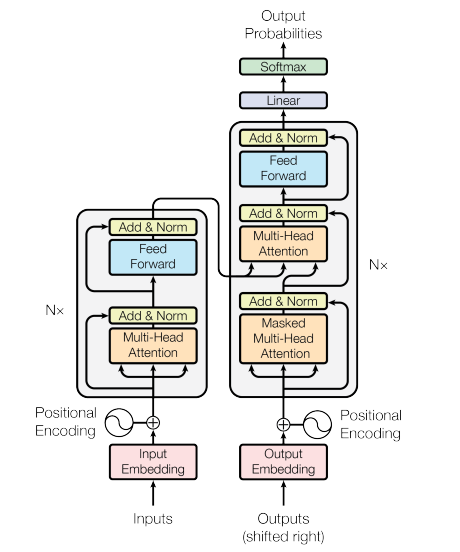


Рисунок 4 – Архитектура трансформера

Устройство трансформера состоит из кодирующего и декодирующего блока, они состоят из нескольких кодеров и энкодеров соответственно. На вход поступает некоторая последовательность, формируется ее векторное представление, прибавляется вектор позиционного кодирования, затем эта последовательность без учета порядка поступает в кодирующий блок, после этого декодирующий блок получает на вход часть последовательности и выход кодирующего блока, в результате получается новая последовательность.

Кодировщики состоят из слоя, реализующего механизм самовнимания, и простого полносвязного слоя, декодировщики также состоят из этих слоев, а также реализуют механизм внимания к выходам кодировщиков.

Механизм внимания основан на присвоении элементам последовательности значений, отражающих их значимость на текущем шаге, то есть модель обращает свое «внимание» на данные элементы.

Функцию внимания можно описать как сопоставление запроса и набора пар ключ-значение с выходными данными, где запрос, ключи, значения и выходные данные являются векторами. Результат рассчитывается как взвешенная сумма значений, где вес, присвоенный каждому значению, вычисляется функцией совместимости запроса с соответствующим ключом. Расчет для матрицы результата представлен в формуле (1).

|  |  |  |
| --- | --- | --- |
|  |  | (1) |

В этом лежит основа работы трансформеров. Также имеются механизмы, улучшающие результаты работы трансформеров, например, множественное внимание, здесь создаются несколько наборов матриц, затем, результаты объединяются, благодаря чему модель может фокусироваться на нескольких аспектах. Можно также выделить позиционное кодирование, позволяющее учитывать порядок во входной последовательности.

На рисунке 5 изображены схемы работы механизмов внимания и множественного внимания.

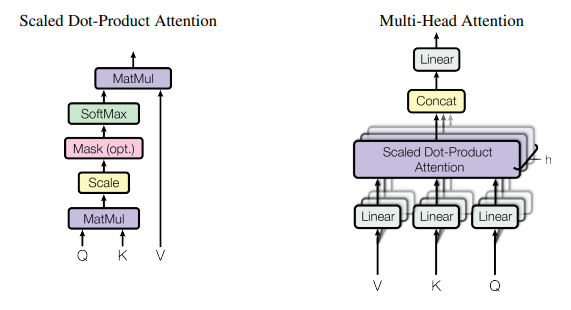


Рисунок 5 – Организация слоев внимания и множественного внимания

Одной из архитектур для задачи генерации музыки является архитектура Wave2Midi2Wave. Эта архитектура представляет собой комбинацию из трех моделей.

1. Первая модель используется для представления аудио в MIDI формате.
2. Вторая сеть является особым типом трансформера, генерирующим новые последовательности музыки с долгосрочной когерентностью. Т.к. данные от первой модели приходят в MIDI формате, задача сводится к работе с символьными данными, что упрощает обучение. В отличии от стандартных моделей трансформеров, данная сеть использует относительное внимание, что позволяет ей генерировать последовательности, выходящие за рамки обучающих данных.
3. Третья сеть, используя набор данных MAESTRO, содержащий записи с конкурса фортепианных исполнителей, и сгенерированные трансформером последовательности в MIDI формате, синтезирует эти последовательности в реалистичный звук [5].

На рисунке 6 представлена схема работы данной архитектуры.

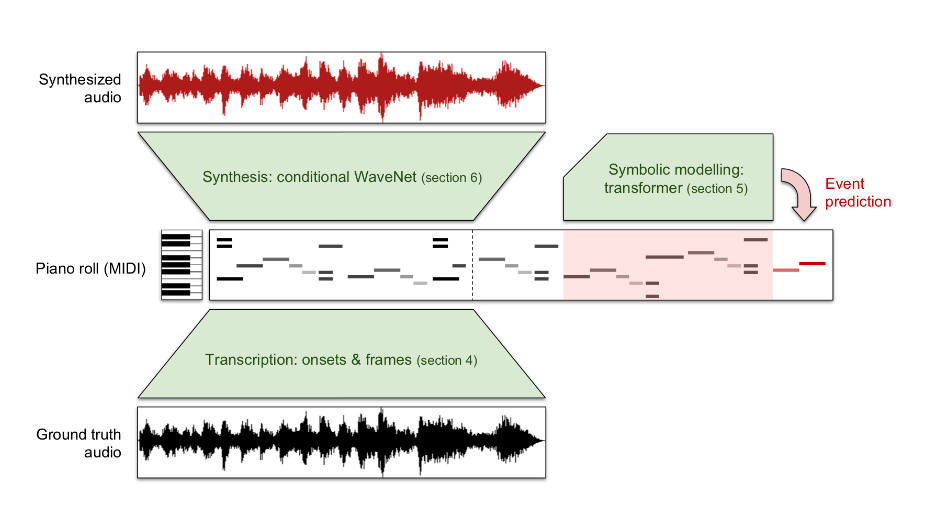


Рисунок 6 – Схема работы архитектуры Wave2Midi2Wave

Также архитектура трансформера используется в задаче генерации аудио по тексту, такая архитектура представлена в статье [6]. Сеть AudioGPT состоит из слоев самовнимания и слоев с обратной связью. Она способна улавливать неочевидные корреляции и производить последовательные выводы. Архитектура представлена на рисунке 7.

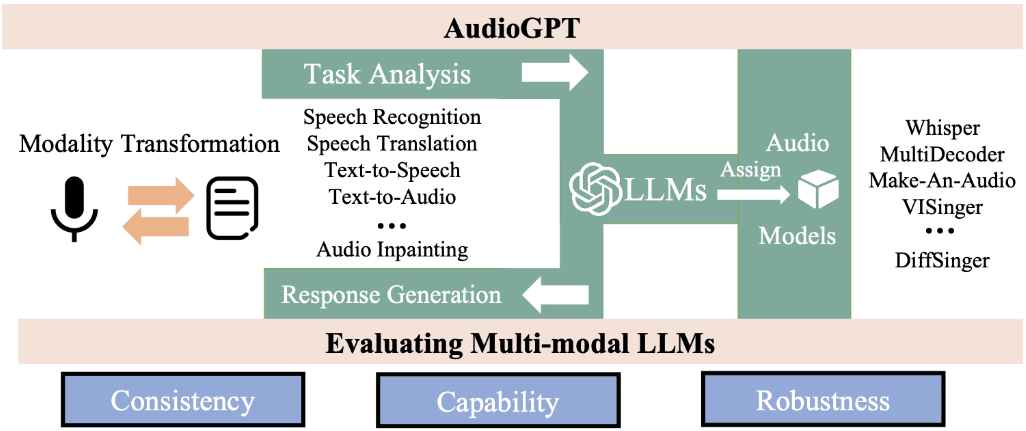


Рисунок 7 – Архитектура AudioGPT

**Генеративно-состязательные сети**

Генеративно-состязательные сети (GAN) получили широкое применение в задачах генерации. Такая архитектура построена на двух нейронных сетях – генераторе и дискриминаторе.

Задачей генератора является генерирование правдоподобных данных, а задача дискриминатора – отличать сгенерированные генератором данные от реальных. В этом кроется идея данной архитектуры – последовательно обучая дискриминатор все лучше и лучше классифицировать реальные и сгенерированные данные, а генератор генерировать все более правдоподобные данные, мы получаем очень качественную генерацию.

В статье [7] представлена архитектура для генерации музыки, состоящей из нескольких партий. Данные представлены как тензор, состоящий из матриц с бинарными значениями, представляющими наличие нот на каждом временном шаге. Архитектура построена на моделировании реальных композиционных подходах и состоит из трех GAN моделей – джемов, композитора и гибридной.

Модель джемов представляет собой несколько независимых генераторов, генерирующих музыку из собственного случайного вектора , где – число генераторов (дорожек), эти генераторы получают «критику», то есть сигналы обратного распространения, от разных дискриминаторов, их число равно числу генераторов.

Модель композитора состоит из одного единственного генератора и одного дискриминатора. Генератор создает аудиозапись, состоящую из дорожек, он требует для этого одного общего для всех дорожек случайного вектора , а дискриминатор исследует все сгенерированные дорожки вместе, чтобы определить является ли входная музыка настоящей или сгенерированной.

Гибридная модель состоит из генераторов и объединяет эти две модели, подавая на вход генераторов две случайные дорожки, одну из сгенерированных моделью джема, а другую из модели композитора, затем результат генераторов объединяется в одну композицию и подается на вход одному единственному дискриминатору. Архитектуры этих трех моделей приведены на рисунке 8, а структура общей модели показана на рисунке 9.

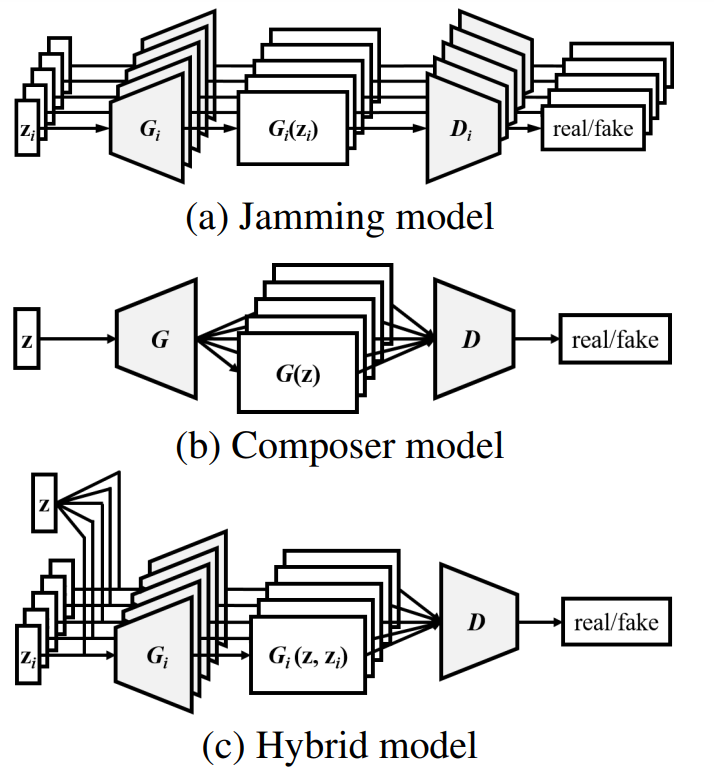


Рисунок 8 – Архитектуры моделей джема, гибридной и композитора

# 2. ПОДГОТОВКА НАБОРА ДАННЫХ

2.1. Формат входных данных

Поговорим немного о формате данных. Поскольку нейронная сеть напрямую не может принимать на вход аудиофайлы, то необходимо преобразовать их в векторный вид.

**Представление аудио данных**

Рассмотрим несколько вариантов представления аудиоданных.

1. Временные последовательности. Простой способ представления аудио – это использование временного сигнала, который представляет изменения амплитуды аудиоволны во времени.
2. Спектрограммы. Преобразование аудиосигнала в двумерное представление, где по одной оси временная шкала, а по другой – частотная. Как правило для ее получения используется оконное преобразование Фурье.
3. Мел-спектрограммы. Это модификация спектрограммы, которая имитирует восприятие звука человеческим ухом. Она использует мел-шкалу частот. Переход к мелам осуществляется с помощью применения мел-фильтров к исходной спектрограмме.
4. MFCC (Mel-Frequency Cepstral Coefficients). Спектрограмма аудиосигнала подвергается процедуре выделения мел-частотных кепстральных коэффициентов, которые представляют собой компактное представление.
5. Embedding-подход. Представление аудиосигналов в виде векторов фиксированной длины, что упрощает их использование в нейронных сетях, предназначенных для обработки последовательностей.

В работе было решено использовать аудиоданные в MIDI формате, который обладает несколькими преимуществами перед mp3 и wav:

1. хранит дискретную информацию о звуке в виде нот, их длительности, громкости и прочих важных характеристик;
2. файл явно разделен на каналы, что позволяет с легкостью извлечь музыкальные дорожки.

2.2. Описание набора данных

В качестве набора данных был использован набор Lakh MIDI Dataset [12], содержащий множество MIDI файлов различных жанров музыки. На рисунке 9 представлена часть информации из midi файла.

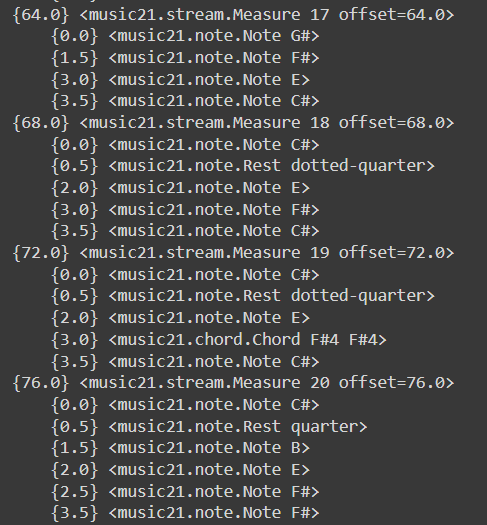


Рисунок 9 – Набор данных

Файл разбит на дорожки, которые в свою очередь разбиты на объекты, такие как ноты, паузы, аккорды и различные перкуссионные инструменты. В фигурных скобках указана длительность объекта в тактах.

2.3. Предобработка данных

Для представления данных было решено выбрать из MIDI файла ноты, аккорды и паузы, а метаданные и информацию о перкуссионных и специфичных для конкретного MIDI файла инструментах вырезать.

Для предобработки и восстановления данных использовалась библиотека music21, предоставляющая обширный инструментарий для обработки MIDI файлов. Библиотека представляет ноты их реальными именами, поэтому каждая нота была закодирована своим числом.

На рисунке 10 представлен метод delete\_percussion, удаляющий ненужные данные из MIDI файлов.

def delete\_percussion(self):

files = os.listdir(self.\_\_temp\_path)

for file in files:

score = converter.parse(os.path.join(self.\_\_temp\_path, file))

new\_score = stream.Score()

for i, part in enumerate(score):

if (i == 0):

continue

is\_part\_to\_delete = False

new\_part = stream.Part()

for item in part.recurse():

if (isinstance(item, note.Unpitched)

or isinstance(item, percussion.PercussionChord)):

is\_part\_to\_delete = True

break

if not ((isinstance(item, note.Note))

or (isinstance(item, note.Rest))

or (isinstance(item, chord.Chord))

or (isinstance(item, stream.Measure))):

dur = item.duration.quarterLength

tmp\_rest = note.Rest()

tmp\_rest.duration.quarterLength = dur

new\_part.append(tmp\_rest)

else:

new\_part.append(copy.deepcopy(item))

if (is\_part\_to\_delete):

continue

else:

new\_score.append(new\_part)

os.remove(os.path.join(self.\_\_temp\_path, file))

new\_score.write('midi',

fp=os.path.join(self.\_\_temp\_path, f"{file}.mid"))

Рисунок 10 – Удаление лишних данных

Каждый файл был закодирован в векторы, длина которых кратна числу временных шагов в такте, которое равно 4. Таким образом, при извлечении нескольких дорожек получается матрица , где – число дорожек, – число временных шагов, элементами которой являются закодированные ноты. Программный код метода preprocess, реализующего предобработку, представлен в приложении.

В ходе предобработки был получен файл, который содержит массив numpy с данными в удобном для обучения формате.

# 3. РЕАЛИЗАЦИЯ НЕЙРОСЕТЕВОЙ МОДЕЛИ

3.1. Архитектура нейронной сети

Выбранная архитектура модели представляет собой упрощенную адаптацию архитектуры MuseGAN, представленной в статье [7], которая представляет собой генеративно-состязательную сеть.

В данной архитектуре задачей генератора является генерация из нескольких векторов шума, интерпретируемых как высокоуровневые признаки мелодии, тактов мелодии, которые по окончании объединяются в одну партитуру. Дискриминатор, в свою очередь, пытается отличить сгенерированную партитуру от настоящей.

Для улучшения результатов была применена функция потерь Вассерштейна со штрафом за градиент.

На рисунке 11 изображена архитектура реализованной модели.

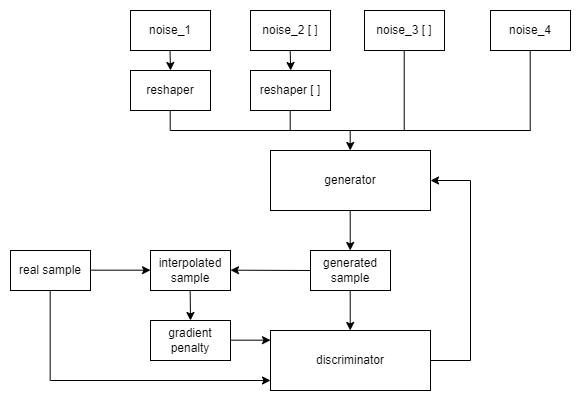


Рисунок 11 – Архитектура модели

**Вектора шумов**

На вход модели подается 4 тензора шума, каждый из которых управляет признаками мелодии. Рассмотрим каждый из них подробнее.

1. Первый тензор является одномерным, он преобразуется промежуточной сетью с помощью операции обратной свертки в двумерный тензор, размерность одной оси которого равна числу генерируемых тактов, а другой – изначальной размерности, таким образом мы имеем набор векторов, каждый из которых подается при генерации соответствующего такта для всех дорожек этого такта. Данный тензор влияет на динамику мелодии, сохраняя при этом связь дорожек.
2. Второй тензор, являющийся двумерным, каждый его вектор также проходит через промежуточную сеть, в результате получается трехмерный тензор, одна из осей которого равна числу дорожек, вторая числу тактов, а третья имеет размерность вектора шума, одинаковую для всех тензоров шума, таким образом, для каждой дорожки каждого такта подается уникальный для нее вектор шума. Этот тензор обеспечивает уникальность каждой дорожки.
3. Третий также является двумерным тензором, одна из осей которого равна числу дорожек, при генерации каждой дорожки на протяжении всех тактов подается соответствующий вектор шума. Таким образом обеспечивается общий тон для каждой дорожки, уникальный для нее.
4. Четвертый является одномерным и подается для всех дорожек и всех тактов. Он обеспечивает общий тон для всей мелодии.

При генерации конкретной дорожки конкретного такта соответствующие вектора объединяются в один, который подается для генерации.

**Генератор**

Генератор состоит из нескольких сетей с одинаковой структурой, задача каждой из них – генерация одного экземпляра дорожки размером в такт. Генерация реализована в виде нескольких операций обратной свертки, преобразующих входной вектор шума в двумерный тензор, размерность которого равна (, ), где – размерность вектора, в который закодирована нота, – число нот в такте. Предварительно вектор шума проходит через полносвязный слой. Программный код, демонстрирующий их структуру представлен на рисунке 12.

def bar\_generator\_initialize(initializer,

noise\_length,

count\_steps\_per\_bar,

count\_notes):

input = Input(shape=(noise\_length \* 4,))

reshape = Dense(1024)(input)

reshape = BatchNormalization(momentum=0.9)(reshape)

reshape = Activation("relu")(reshape)

reshape = Reshape([2, 1, 512])(reshape)

conv\_transpose1 = Conv2DTranspose(1024, (2, 1), (2, 1), "same",

kernel\_initializer=initializer)(reshape)

conv\_transpose1 = BatchNormalization(momentum=0.9)(conv\_transpose1)

conv\_transpose1 = Activation("relu")(conv\_transpose1)

conv\_transpose2 = Conv2DTranspose(512, (2, 1), (2, 1), "same",

kernel\_initializer=initializer)(conv\_transpose1)

conv\_transpose2 = BatchNormalization(momentum=0.9)(conv\_transpose2)

conv\_transpose2 = Activation("relu")(conv\_transpose2)

conv\_transpose3 = Conv2DTranspose(256, (2, 1), (2, 1), "same",

kernel\_initializer=initializer)(conv\_transpose2)

conv\_transpose3 = BatchNormalization(momentum=0.9)(conv\_transpose3)

conv\_transpose3 = Activation("relu")(conv\_transpose3)

conv\_transpose4 = Conv2DTranspose(256, (1, 5), (1, 5), "same",

kernel\_initializer=initializer)(conv\_transpose3)

conv\_transpose4 = BatchNormalization(momentum=0.9)(conv\_transpose4)

conv\_transpose4 = Activation("relu")(conv\_transpose4)

conv\_transpose5 = Conv2DTranspose(256, (1, 5), (1, 5), "same",

kernel\_initializer=initializer)(conv\_transpose4)

conv\_transpose5 = BatchNormalization(momentum=0.9)(conv\_transpose5)

conv\_transpose5 = Activation("relu")(conv\_transpose5)

conv\_transpose6 = Conv2DTranspose(1, (1, 3), (1, 3), "same",

kernel\_initializer=initializer)(conv\_transpose5)

conv\_transpose6 = Activation("tanh")(conv\_transpose6)

output = Reshape([1, count\_steps\_per\_bar,

count\_notes, 1])(conv\_transpose6)

return Model(input, output)

Рисунок 12 – Структура сетей, генерирующих дорожки

Цикл работы генератора состоит в следующем – на вход генератора поступают тензоры шумов, далее из этих тензоров генерируются дорожки, которые затем объединяются в такт, а затем сгенерированные такты объединяются в мелодию. Программный код генератора представлен в приложении.

**Дискриминатор**

Дискриминатор представляет собой сверточную сеть, на выходе которой прогноз того, насколько партитура соответствует настоящей. Программный код дискриминатора представлен на рисунке 13.

def discriminator\_initialize(initializer, count\_bars, count\_steps\_per\_bar, count\_notes, count\_tracks):

input = Input(shape=(count\_bars, count\_steps\_per\_bar,

count\_notes, count\_tracks))

conv1 = Conv3D(128, (2, 1, 1), (1, 1, 1), "valid")(input)

conv1 = LeakyReLU()(conv1)

conv2 = Conv3D(128, (count\_bars - 1, 1, 1), (1, 1, 1),

"valid")(conv1)

conv2 = LeakyReLU()(conv2)

conv3 = Conv3D(128, (1, 1, 3), (1, 1, 3), "same")(conv2)

conv3 = LeakyReLU()(conv3)

conv4 = Conv3D(128, (1, 1, 5), (1, 1, 5), "same")(conv3)

conv4 = LeakyReLU()(conv4)

conv5 = Conv3D(128, (1, 1, 5), (1, 1, 5), "same")(conv4)

conv5 = LeakyReLU()(conv5)

conv6 = Conv3D(128, (1, 2, 1), (1, 2, 1), "same")(conv5)

conv6 = LeakyReLU()(conv6)

conv7 = Conv3D(128, (1, 2, 1), (1, 2, 1), "same")(conv6)

conv7 = LeakyReLU()(conv7)

conv8 = Conv3D(256, (1, 2, 1), (1, 2, 1), "same")(conv7)

conv8 = LeakyReLU()(conv8)

conv9 = Conv3D(512, (1, 2, 1), (1, 2, 1), "same")(conv8)

conv9 = LeakyReLU()(conv9)

flatten = Flatten()(conv9)

dense = Dense(1024, kernel\_initializer=initializer)(flatten)

dense = LeakyReLU()(dense)

output = Dense(1, None, kernel\_initializer=initializer)(dense)

return Model(input, output)

Рисунок 13 – Структура дискриминатора

**Обучение**

Шаг обучения состоит из попеременного обучения дискриминатора и генератора. Обучение дискриминатора состоит в следующем – генератор генерирует образцы, затем на вход дискриминатора подаются сгенерированные и реальные партитуры, затем вычисляются потери Вассерштейна, впервые описанные в статье [13], в которой представлен алгоритм для расчета потерь Вассерштейна, резюмируя который потерями является разница между средней оценкой критика на реальных образцах и средней оценкой на сгенерированных. Потери Вассерштейна можно интерпретировать как меру расстояния между сгенерированными и реальными образцами. Для использования такой оценки, необходимо максимизировать разницу между выходами для реальных и сгенерированных данных, поэтому на выходе дискриминатора используется линейная функция активации, которая не ограничена в своем диапазоне. Таким образом, дискриминатор является своего рода критиком, задача которого максимизировать разницу между сгенерированными и реальными образцами. Такая функция потерь решает проблему затухающего градиента, возникающую, когда дискриминатор становится слишком силен. Однако, ввиду отсутствия ограничения на ее значения, возникает противоположная проблема – взрывной рост градиента, которая решается в статье [14] с помощью штрафа при отклонении нормы градиента от 1, накладываемого на функцию потерь дискриминатора. Штраф за градиент вычисляется как квадрат разности между нормой градиента и 1, для поддержания баланса, градиент вычисляется для интерполированного образца.

После корректировки весов дискриминатора обучается генератор, функцией потерь которого является отрицательный выход дискриминатора, таким образом, чем выше дискриминатор оценит сгенерированный образец, тем ниже будут потери для генератора. Программный код обучения модели и функции, вычисляющей штраф за градиент, представлен в приложении.

3.2. Анализ обучения модели

На рисунке 14 представлены графики потерь генератора и дискриминатора. На которых можно увидеть, как изменялись значения потерь во время обучения.

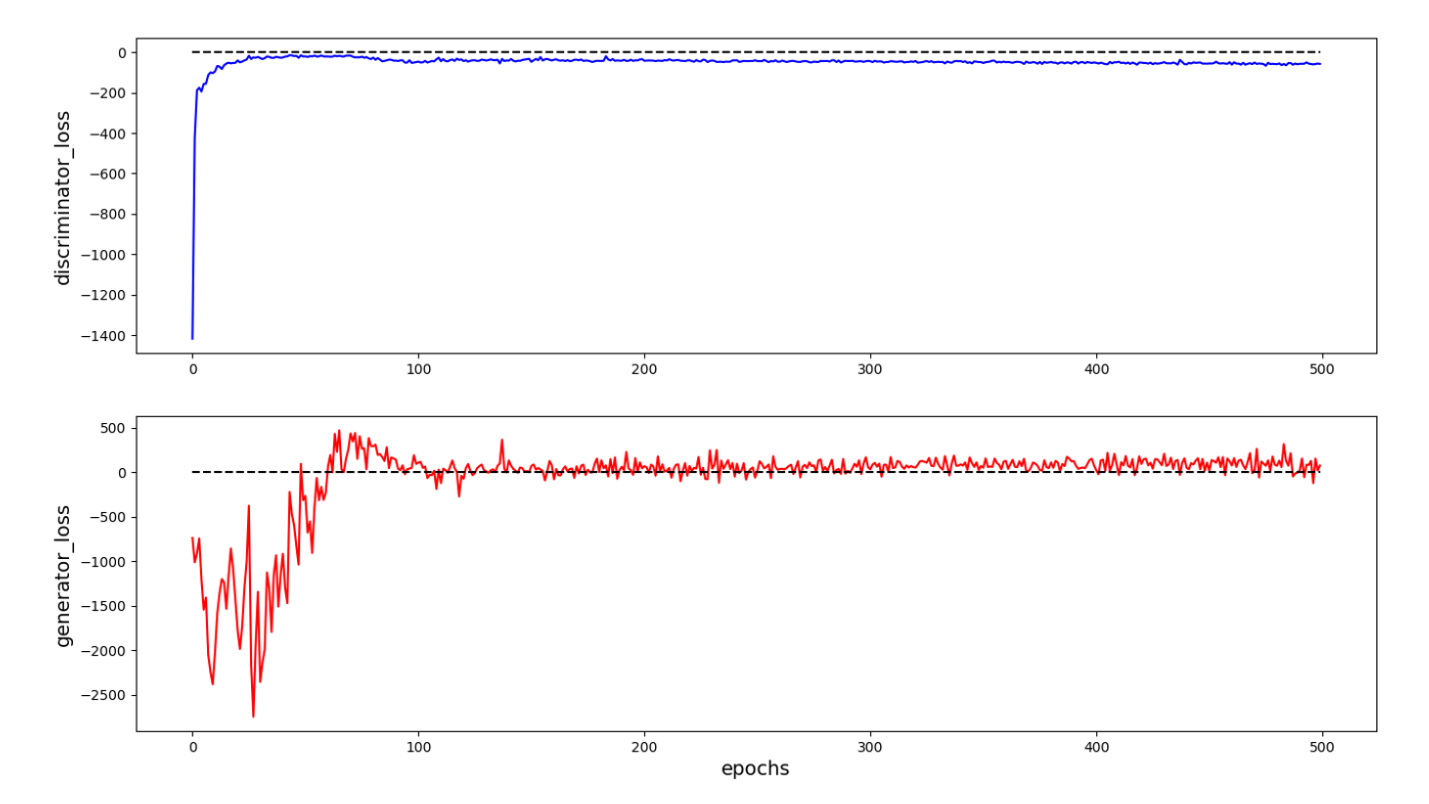


Рисунок 14 – Графики потерь модели

Так как генератор обучался меньшее число раз, чем дискриминатор, второй довольно быстро стал сильнее и далее стал подстраиваться под улучшающийся генератор. Генератор в свою очередь первое время показывал хорошие результаты, но затем дискриминатор стал достаточно сильным, чтобы отличать реальные образцы от сгенерированных, в результате чего потери генератора также вышли на плато, и далее процесс обучения шел стабильно, улучшая результаты обеих сетей.

# 4. РАЗРАБОТКА ПРИЛОЖЕНИЯ

4.1. Проектирование

**Функциональные требования**

В ходе проектирования были выделены следующие функциональные требования:

1. пользователь должен иметь возможность сгенерировать музыкальную партитуру;
2. пользователь должен иметь возможность просмотреть ранее cгенерированные файлы;
3. пользователь должен иметь возможность управлять генерацией;
4. пользователь должен иметь возможность изменить пароль;
5. система должна иметь возможность регистрации и авторизации;
6. система должна иметь систему токенов, для ограничения использования ресурсов;

**Нефункциональные требования**

Также были выделены следующие нефункциональные требования:

1. приложение должно быть написано с использованием языка программирования Python 3;
2. приложение должно использовать библиотеку tensorflow для работы с моделью;
3. приложение должно генерировать партитуры в формате midi;
4. пароли пользователей должны храниться в хэшированном виде;

**Диаграмма вариантов использования**

При проектировании приложения был использован язык графического моделирования UML. С системой взаимодействует 1 актер – пользователь. На рисунке 15 представлена диаграмма вариантов использования приложения.

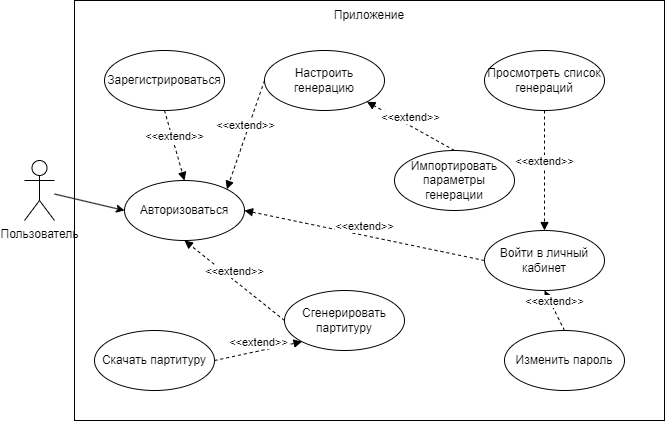


Рисунок 15 – Диаграмма вариантов использования приложения

**Проектирование базы данных**

На рисунке 16 показана схема базы данных приложения.

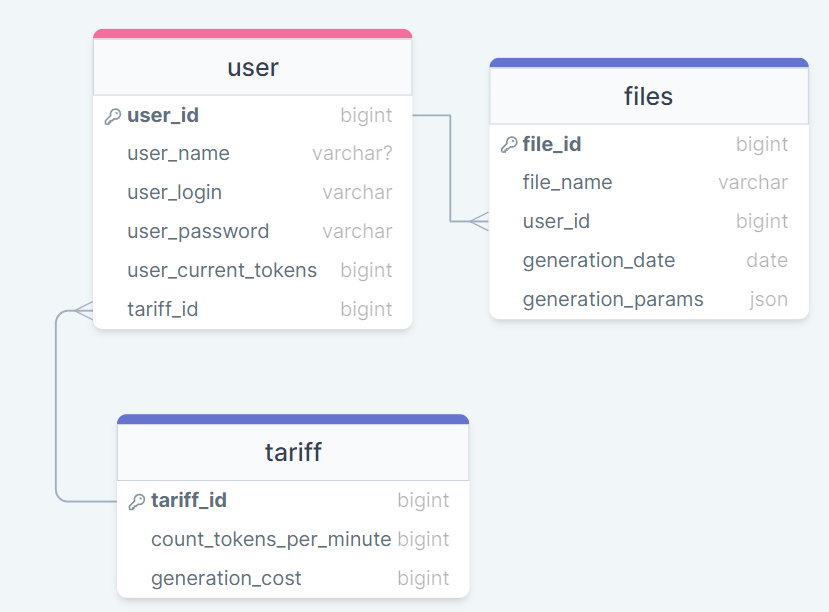


Рисунок 16 – Схема базы данных

Таблица user содержит информацию о существующих в приложении пользователях, описание полей таблицы user:

1. user\_id – идентификатор;
2. user\_name – имя пользователя для представления в системе;
3. user\_login – логин пользователя для авторизации;
4. user\_password – хэш пароля пользователя;
5. user\_current\_tokens – число токенов пользователя;
6. tariff\_id – идентификатор примененного тарифа генерации.

Таблица tariff содержит информацию о существующих в приложении тарифах, описание полей таблицы tariff:

1. tariff\_id – идентификатор;
2. count\_tokens\_per\_minute – число токенов начисляемых каждую минуту;
3. generation\_cost – цена генерации одной партитуры.

Таблица files содержит информацию о файлах, сгенерированных пользователями:

1. file\_id – идентификатор;
2. file\_name – имя файла на сервере;
3. user\_id – пользователь, сгенерировавший файл;
4. generation\_date – дата генерации файла;
5. generation\_params – вектора шумов, использованных при генерации партитуры.

4.2. Реализация приложения

Для реализации серверной части приложения была использована библиотека flask, для работы с базой данной использовалась библиотека SQLAlchemy.

# ЗАКЛЮЧЕНИЕ

В рамках данной работы был проведен обзор существующих проектов для генерации музыки.

Был проведен анализ архитектур нейронных сетей, применяемых в задаче генерации аудио.

Были рассмотрены способы представления аудиоданных.

Был рассмотрен и обработан набор данных.

Была выбрана архитектура нейросетевой модели.

# ЛИТЕРАТУРА

1. Фостер Д. Генеративное глубокое обучение. Творческий потенциал нейронных сетей. – СПб.: Питер, 2020. – 352 с.
2. Nayebi A., Vitelli M. GRUV: Algorithmic Music Generation using Recurrent Neural Networks. [Электронный ресурс] URL: https://cs224d.stanford.edu/reports/NayebiAran.pdf (дата обращения: 15.01.2024 г.).
3. Mangal S., Modak R., Joshi P. LSTM Based Music Generation System. // ArXiv, 2019. – 6 p.
4. Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I. Attention is All you Need. // ArXiv, 2017. – 15 p.
5. The MAESTRO Dataset and Wave2Midi2Wave. [Электронный ресурс] URL: https://magenta.tensorflow.org/maestro-wave2midi2wave (дата обращения: 15.01.2024 г.).
6. Huang R., Li M., Yang D., Shi J., Chang X., Ye Z., Wu Y., Hong Z., Huang J., Liu J., Ren Y., Zhao Z., Watanabe S. Audiogpt: Understanding and generating speech, music, sound, and talking head. // ArXiv, 2023. – 14 p.
7. Hao-Wen Dong, Wen-Yi Hsiao, Li-Chia Yang, Yi-Hsuan Yang. MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment. // ArXiv, 2017. – 13 p.
8. Jukebox. [Электронный ресурс] URL: https://openai.com/research/jukebox (дата обращения: 15.01.2024 г.).
9. Musenet. [Электронный ресурс] URL: https://openai.com/research/musenet (дата обращения: 15.01.2024 г.).
10. Glazyrin N. CONTEXT-AWARE GENERATION OF MELODIC MIDI LOOPS. [Электронный ресурс] URL: https://archives.ismir.net/ismir2021/latebreaking/000037.pdf (дата обращения: 15.01.2024 г.).
11. Agostinelli A., Denk T. I., Borsos Z., Engel J., Verzetti M., Caillon A., Huang Q., Jansen A., Roberts A., Tagliasacchi M., Sharifi M., Zeghidour N., Frank C. Musiclm: Generating music from text. // arXiv, 2023. – 15 p.
12. Lakh MIDI dataset. [Электронный ресурс] URL: https://colinraffel.com/projects/lmd/ (дата обращения: 20.03.2024 г.).
13. Arjovsky M., Chintala S., Bottou L. Wasserstein GAN. // arXiv, 2017. – 32 p.
14. Gulrajani I., Ahmed F., Arjovsky M., Dumoulin V., Courville A. Improved Training of Wasserstein GANs. // arXiv, 2017. – 20 p.

# ПРИЛОЖЕНИЕ

Листинг 1 – Метод preprocess

def preprocess(self, n\_bars, n\_tracks=2, n\_steps\_per\_bar=4):

step\_time = 1 / n\_steps\_per\_bar

to\_npy = []

files = os.listdir(self.\_\_temp\_path)

for file in files:

score = converter.parse(os.path.join(self.\_\_temp\_path, file))

score\_matrix = [[] for i in range(n\_tracks)]

melody\_start\_offset = 0

is\_start\_find = False

for i, part in enumerate(score):

if (i == 0):

continue

if (i == n\_tracks + 1):

break

current\_time = 0

for item in part.flatten():

if (i == 1 and not is\_start\_find):

if (isinstance(item, note.Note) or isinstance(item, chord.Chord)):

is\_start\_find = True

if not (is\_start\_find):

melody\_start\_offset += item.duration.quarterLength

current\_time += item.duration.quarterLength

if (is\_start\_find):

if ((isinstance(item, note.Note) or isinstance(item, chord.Chord) or isinstance(item, note.Rest))

and (current\_time >= melody\_start\_offset)

and ((current\_time - melody\_start\_offset) < n\_bars)):

if (isinstance(item, note.Note)):

score\_matrix[i - 1] += ([str(item.nameWithOctave)] \*

int(item.duration.quarterLength / step\_time))

if (isinstance(item, chord.Chord)):

score\_matrix[i - 1] += ['.'.join(n.nameWithOctave for n in item.pitches)] \* int(item.duration.quarterLength / step\_time)

if (isinstance(item, note.Rest)):

score\_matrix[i - 1] += ([str(item.name)] \*

int(item.duration.quarterLength / step\_time))

to\_npy.append(score\_matrix)

max\_len = 0

all\_notes = {'rest'}

for matrix in to\_npy:

for part in matrix:

if (max\_len < len(part)):

all\_notes = all\_notes.union(set(part))

max\_len = len(part)

all\_notes.discard('rest')

all\_notes = ['rest'] + list(all\_notes)

for matrix in to\_npy:

for j, part in enumerate(matrix):

part += ['rest'] \* (max\_len - len(part))

matrix[j] = [part.index(n) for n in part]

to\_npy = np.array(to\_npy)

Окончание листинга 1

np.save(os.path.join(self.\_\_preprocessed\_data\_path, "dataset"), to\_npy)

}

}

public void continueInPausePressed()

{

Cursor.SetCursor(cursor\_game, game\_hot\_spot, CursorMode.Auto);

pause\_menu.SetActive(false);

Time.timeScale = 1f;

}

public void continueInUpgradePressed()

{

Cursor.SetCursor(cursor\_game, game\_hot\_spot, CursorMode.Auto);

upgrade\_menu.SetActive(false);

Time.timeScale = 1f;

}

public void continueInMapPressed()

{

Cursor.SetCursor(cursor\_game, game\_hot\_spot, CursorMode.Auto);

map\_menu.SetActive(false);

Time.timeScale = 1f;

}

public void mainMenuPressed()

{

SceneManager.LoadScene("Menu");

Time.timeScale = 1f;

}

}

Листинг 2 – Реализация генератора

def generator\_initialize(initializer, noise\_length, count\_tracks, count\_bars, count\_steps\_per\_bar, count\_notes):

chords = Input(shape=(noise\_length,))

style = Input(shape=(noise\_length,))

melody = Input(shape=(count\_tracks, noise\_length))

groove = Input(shape=(count\_tracks, noise\_length))

chords\_reshaper = noise\_reshaper\_initialize(

initializer, noise\_length, count\_bars

)

chords\_reshaped = chords\_reshaper(chords)

melody\_reshapers = []

melody\_reshaped = []

for track\_ix in range(count\_tracks):

melody\_reshapers.append(noise\_reshaper\_initialize(

initializer, noise\_length, count\_bars

))

melody\_track = Lambda(lambda x, track\_ix=track\_ix: x[:, track\_ix, :])(melody)

melody\_reshaped.append(melody\_reshapers[track\_ix](melody\_track))

bar\_generators = []

for track\_ix in range(count\_tracks):

bar\_generators.append(bar\_generator\_initialize(

initializer, noise\_length, count\_steps\_per\_bar, count\_notes

))

Окончание листинга 2

bars\_output = []

chord\_noise = []

for bar\_ix in range(count\_bars):

chord\_noise.append(Lambda(lambda x, bar\_ix=bar\_ix: x[:, bar\_ix, :])(chords\_reshaped))

style\_noise = style

tracks\_output = []

for track\_ix in range(count\_tracks):

melody\_noise = Lambda(lambda x, bar\_ix=bar\_ix: x[:, bar\_ix, :])(melody\_reshaped[track\_ix])

groove\_noise = Lambda(lambda x, track\_ix=track\_ix: x[:, track\_ix, :])(groove)

concat\_input = Concatenate(axis=1)([chord\_noise[bar\_ix], style\_noise, melody\_noise, groove\_noise])

tracks\_output.append(bar\_generators[track\_ix](concat\_input))

bars\_output.append(Concatenate(axis=-1)(tracks\_output))

output = Concatenate(axis=1)(bars\_output)

return Model([chords, style, melody, groove], output)

Листинг 3 – Функция штрафа за градиент

def gradient\_penalty(self, batch\_size, real, fake):

alpha = tf.random.normal([batch\_size, 1, 1, 1, 1], 0.0, 1.0)

difference = fake - real

interpolated = real + alpha \* difference

with tf.GradientTape() as tape:

tape.watch(interpolated)

pred = self.discriminator(interpolated, training=True)

grads = tape.gradient(pred, [interpolated])[0]

norm = tf.sqrt(tf.reduce\_sum(tf.square(grads), axis=[1, 2, 3]))

gradient\_penalty = tf.reduce\_mean((norm - 1.0) \*\* 2)

return gradient\_penalty

Листинг 4 – Обучение модели

def train\_step(self, real):

batch\_size = tf.shape(real)[0]

for i in range(self.discriminator\_steps):

chords\_input = tf.random.normal(shape=(batch\_size, self.noise\_length))

style\_input = tf.random.normal(shape=(batch\_size, self.noise\_length))

melody\_input = tf.random.normal(shape=(batch\_size, self.count\_tracks, self.noise\_length))

groove\_input = tf.random.normal(shape=(batch\_size, self.count\_tracks, self.noise\_length))

input = [chords\_input, style\_input, melody\_input, groove\_input]

with tf.GradientTape() as tape:

fake = self.generator(input, training=True)

fake\_preds = self.discriminator(fake, training=True)

real\_preds = self.discriminator(real, training=True)

wasserstein\_loss = tf.reduce\_mean(fake\_preds) - tf.reduce\_mean(real\_preds)

Окончание листинга 4

gradient\_penalty = self.gradient\_penalty(batch\_size, real, fake)

discriminator\_loss = wasserstein\_loss + gradient\_penalty \* self.gradient\_penalty\_weight

gradient = tape.gradient(discriminator\_loss, self.discriminator.trainable\_variables)

self.discriminator\_optimizer.apply\_gradients(zip(gradient, self.discriminator.trainable\_variables))

chords\_input = tf.random.normal(shape=(batch\_size, self.noise\_length))

style\_input = tf.random.normal(shape=(batch\_size, self.noise\_length))

melody\_input = tf.random.normal(shape=(batch\_size, self.count\_tracks, self.noise\_length))

groove\_input = tf.random.normal(shape=(batch\_size, self.count\_tracks, self.noise\_length))

input = [chords\_input, style\_input, melody\_input, groove\_input]

with tf.GradientTape() as tape:

fake = self.generator(input, training=True)

fake\_preds = self.discriminator(fake, training=True)

generator\_loss = -tf.reduce\_mean(fake\_preds)

gradient = tape.gradient(generator\_loss, self.generator.trainable\_variables)

self.generator\_optimizer.apply\_gradients(zip(gradient, self.generator.trainable\_variables))

self.discriminator\_loss\_metric.update\_state(discriminator\_loss)

self.generator\_loss\_metric.update\_state(generator\_loss)

return {item.name: item.result() for item in self.metrics}