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
Начальник отдела
суперкомпьютерного моделирования
НИУ ВШЭ, к.ф.-м.н., доцент

_____ П.С. Костенецкий

«___»_____ 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.
_____ 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 22.04.2024 г. № 764-13/12)

Разработка веб-приложения для генерации музыки.

2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.

3. **Исходные данные к работе**

3.1. Kailash A. Generative Adversarial Networks Projects: Build next-generation generative models using TensorFlow and Keras. – Packt Publishing Ltd., 2019. – 316 p.

3.2. Foster D. Generative Deep Learning, 2nd Edition. – O'Reilly Media, Inc., 2023. – 426 p.

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области.

4.2. Подготовить набор данных.

4.3. Выбрать топологию нейронной сети.

4.4. Реализовать нейросетевую модель и провести ее обучение.

4.5. Спроектировать приложение для генерации музыки.

4.6. Реализовать приложение.

4.7. Провести тестирование приложения.

**5. Дата выдачи задания:** 29.01.2024 г.

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

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

**ОГЛАВЛЕНИЕ**

# 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 темой исследования, которая может иметь значительное влияние на музыкальную индустрию и творческий процесс.

Генерация 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 настроения. Генератор мелодий в свою очередь мог бы помочь в поиске идей. 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.

**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 53 страницы, объем списка литературы – 25 источников.

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

Вторая глава посвящена проектированию приложения.

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

В четвертой главе содержится описание архитектуры нейросетевой модели, ее реализация и анализ обучения.

В пятой главе описана реализация приложения.

В шестой главе приводятся результаты тестирования приложения.

В приложении представлены листинги функций.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1. Анализ аналогичных проектов

Рассмотрим несколько наиболее актуальных и современных решений данной задачи.

**Jukebox**

Проект Jukebox [1] 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.

**MuseNet**

Созданная исследователями из OpenAI MuseNet [2] 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.

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

Основное отличие от Jukebox состоит в том, что MuseNet сконцентрирована на создании оригинальной музыки на основе стилей, жанров и характеристик, заданных пользователем, однако она не способна генерировать вокальные партии.

**MusicLM**

Модель MusicLM [3], разработана 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.

**Нейромузыка**

Модель Нейромузыка [4], разработана исследователями из Yandex, и включена в сервис Yandex Music для генерации бесконечного аудиопотока определенного настроения и жанра.

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

## 1.2. Анализ архитектур нейронных сетей

В обобщенном виде задача генерации аудио сводится к предсказанию последовательности по 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 [5]. Также для решения задачи подходят более сложные архитектуры, такие как трансформеры [6] и генеративно-состязательные сети (GAN), которые показывают результаты лучше, чем классические подходы [7]. Достойны упоминания и другие подходы искусственного интеллекта, такие как обучение с подкреплением [8].

**Рекуррентные нейронные сети**

Рекуррентные нейронные сети (Recurrent Neural Network, RNN) – представляют собой класс нейронных сетей, разработанный специально для обработки последовательных данных, они отличаются от стандартных нейронных сетей тем, что используют принцип обратной связи, подавая на вход рекуррентного слоя его выход в предыдущий момент времени [9]. Это делает их очень эффективными при работе с последовательными данными различной природы. Рекуррентная нейронная сеть обрабатывает последовательность, перебирая ее элементы и сохраняя состояние, полученное при обработке предыдущих элементов.

Однако простейшая рекуррентная сеть обладает таким недостатком, как затухание градиента при обучении на длинных последовательностях данных, это означает, что информация о предыдущих состояниях становится все менее доступной для текущих шагов, иными словами нейронная сеть «забывает» информацию. Для решения этой проблемы была придумана архитектура LSTM, ее схема представлена на рисунке 1.

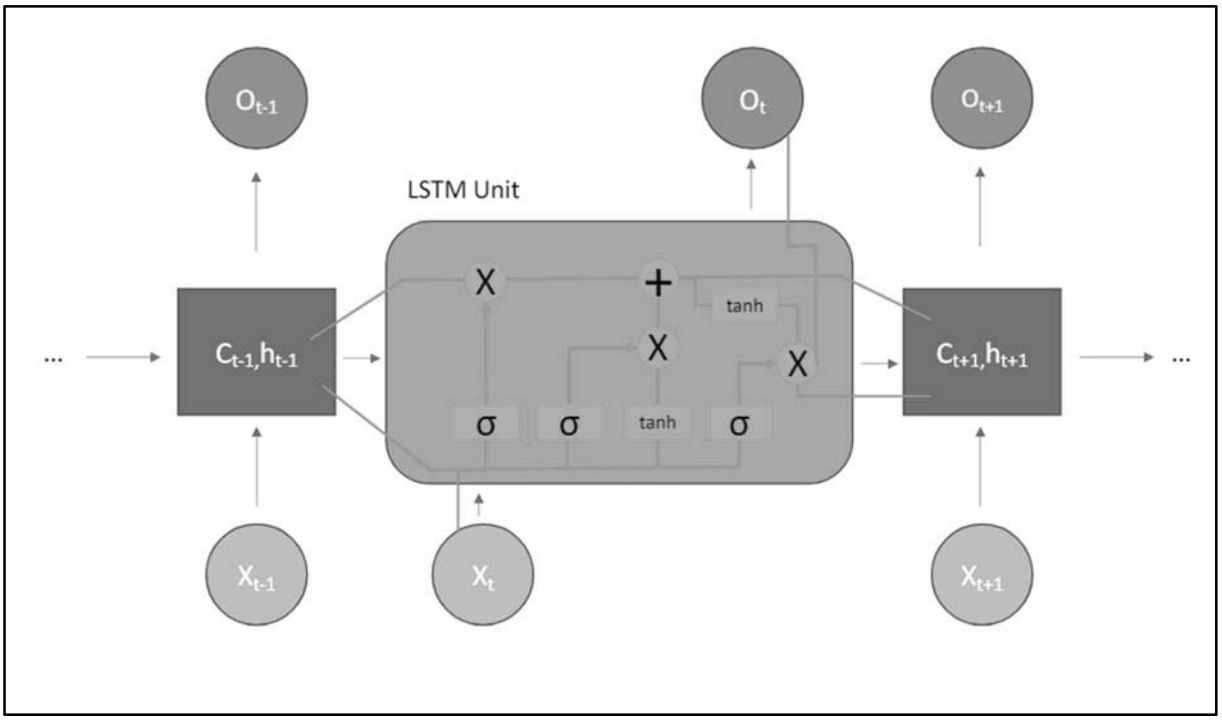


Рисунок 1 – Архитектура LSTM [10]

Ключевой особенностью данной архитектуры является механизм долгосрочной памяти. Это позволяет использовать информацию из ранних интервалов на более поздних. За реализацию этого механизма на данной схеме отвечает выходной сигнал $C_t$. Имитация забывания ненужной информации реализована операцией поэлементного умножения, а запоминание новой информации операцией поэлементного сложения [9].

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 и ускоряя процесс обучения. Ее схема представлена на рисунке 2.

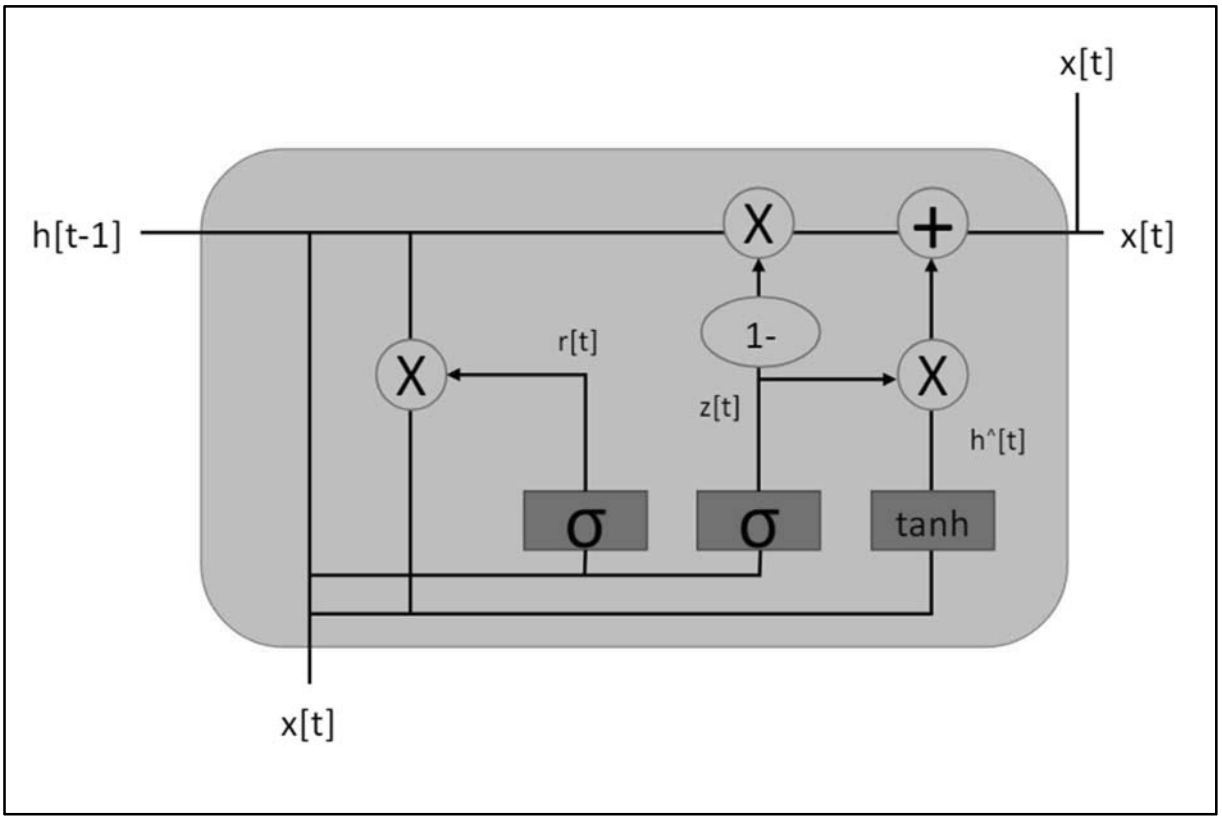


Рисунок 2 – Архитектура GRU [10]

В статье [11] 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 [12] используется иной подход, где сеть LSTM прогнозирует не только саму частоту, но и время, в которое она воспроизводится. На рисунке 3 изображена архитектура предложенной модели.

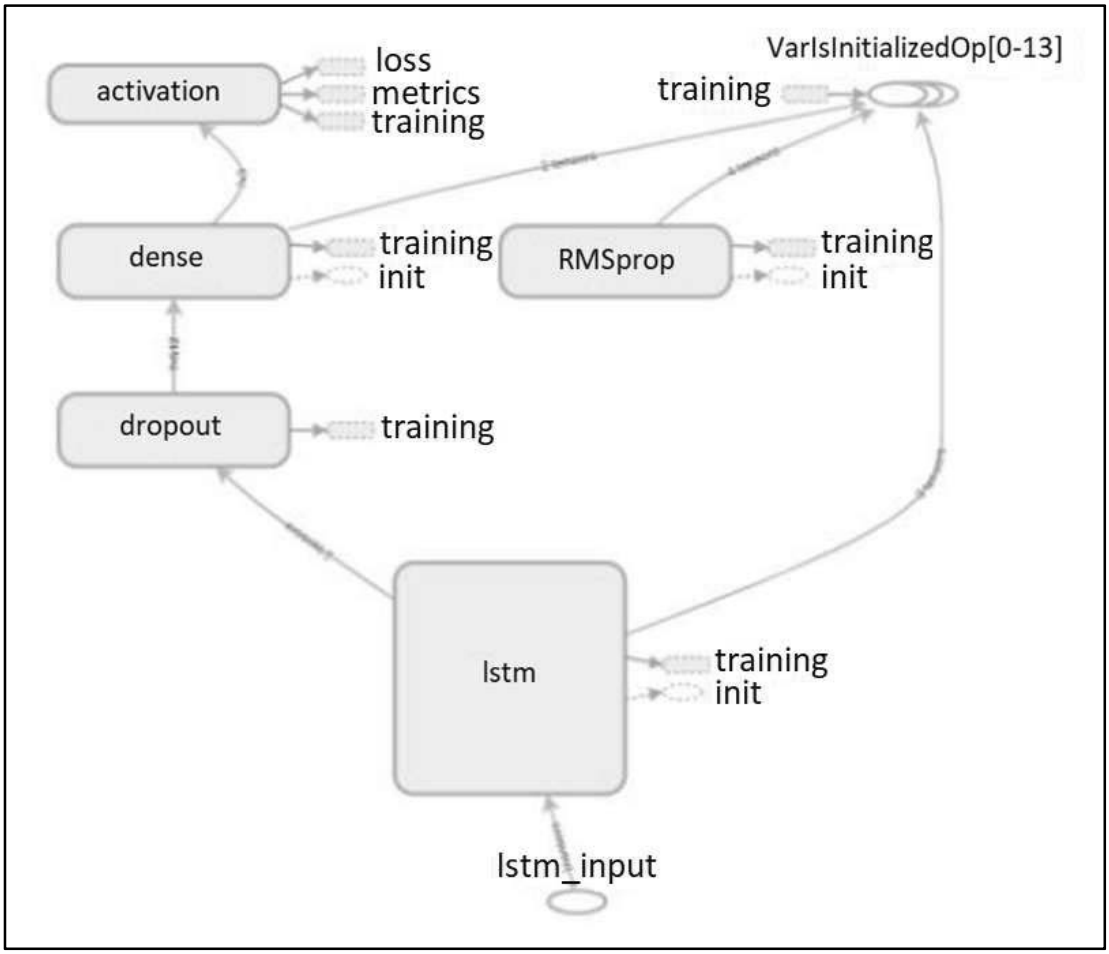


Рисунок 3 – Модель генерации аудио [12]

**Трансформеры**

Трансформер – архитектура нейронных сетей, избегающая рекуррентности и полностью полагающаяся на механизм внимания для построения зависимостей между входом и выходом [13]. Основное преимущество данной архитектуры по сравнению с RNN – высокая эффективность в условиях параллелизации [14]. Архитектура представлена на рисунке 4.

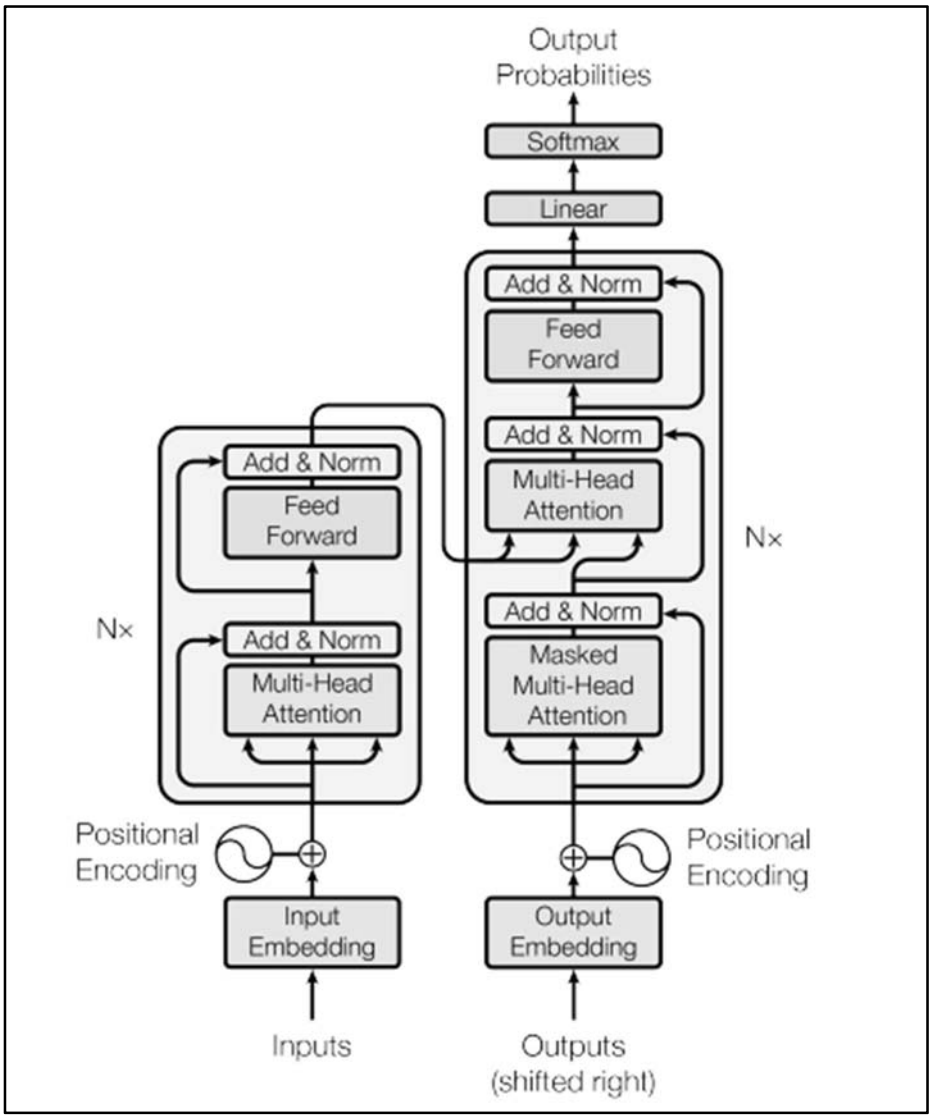


Рисунок 4 – Архитектура трансформера [13]

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):

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V. \quad (1)$$

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.

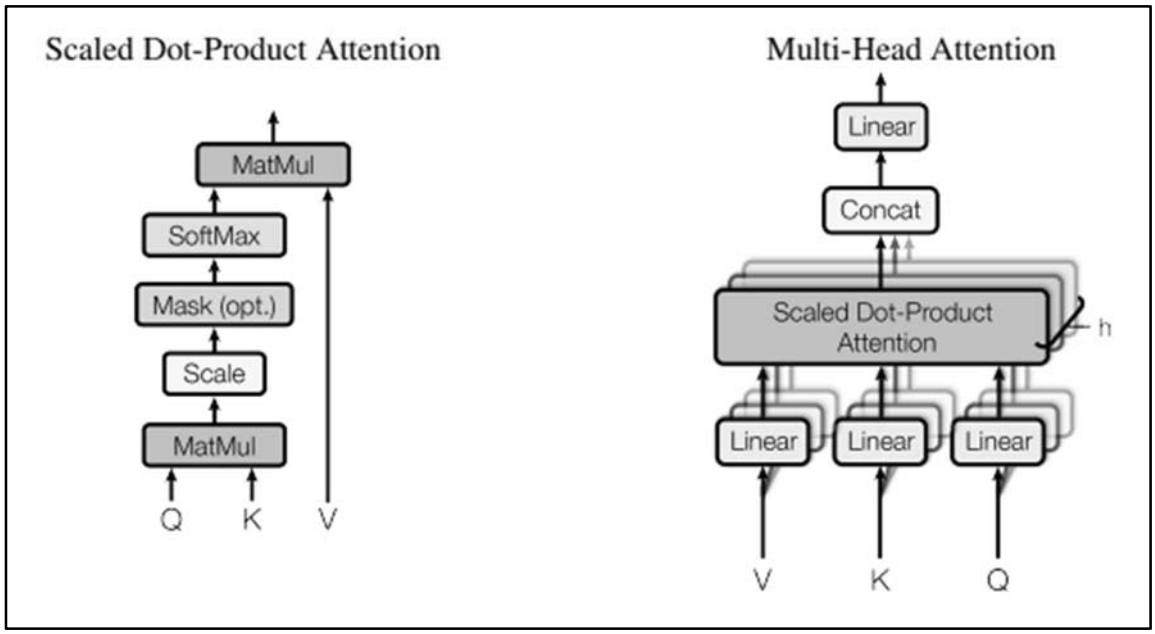


Рисунок 5 – Организация слоев внимания и множественного внимания [13]

Одной из архитектур для задачи генерации музыки является архитектура Wave2Midi2Wave [15]. 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.

На рисунке 6 представлена схема работы данной архитектуры.

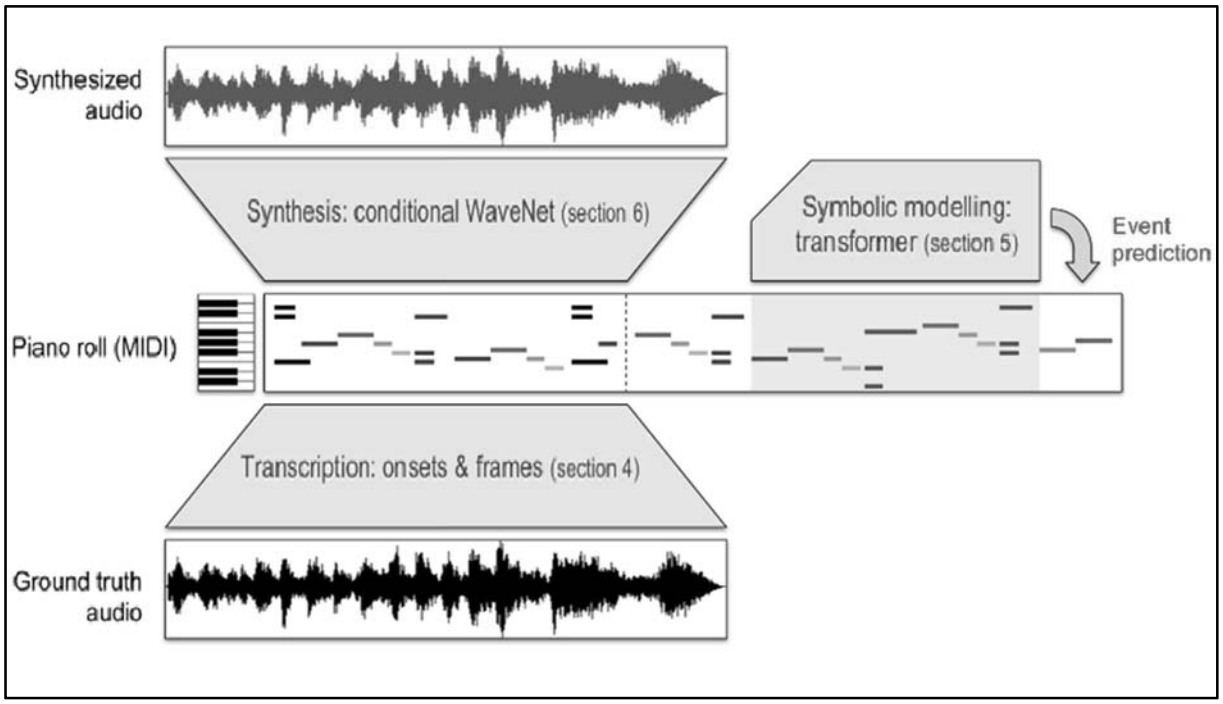


Рисунок 6 – Схема работы архитектуры Wave2Midi2Wave [15]

Также архитектура трансформера используется в задаче генерации аудио по тексту, такая архитектура представлена в статье [16]. 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.

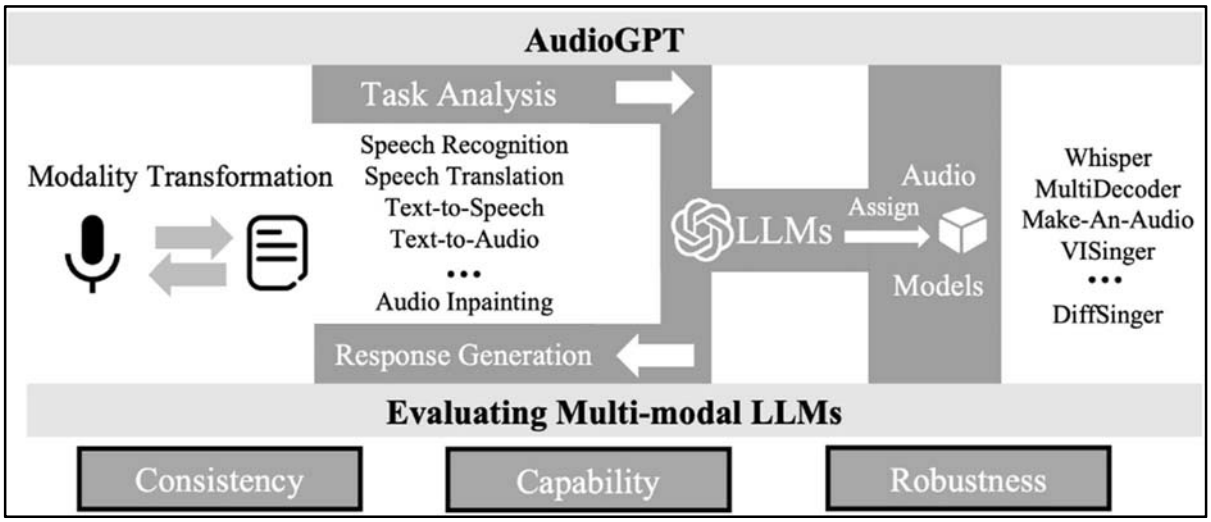


Рисунок 7 – Архитектура AudioGPT [16]

**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 [17] 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 $z_i$, $i = 1, 2, \dots, M$, где $M$ – 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 $M$ дорожек, он требует для этого одного общего для всех дорожек случайного вектора $z$, а дискриминатор исследует все сгенерированные дорожки вместе, чтобы определить является ли входная музыка настоящей или сгенерированной.

Гибридная модель состоит из $M$ генераторов и объединяет эти две модели, подавая на вход генераторов две случайные дорожки, одну из сгенерированных моделью джема, а другую из модели композитора, затем ре-

зультат генераторов объединяется в одну композицию и подается на вход одному единственному дискриминатору. Архитектуры этих трех моделей приведены на рисунке 8.

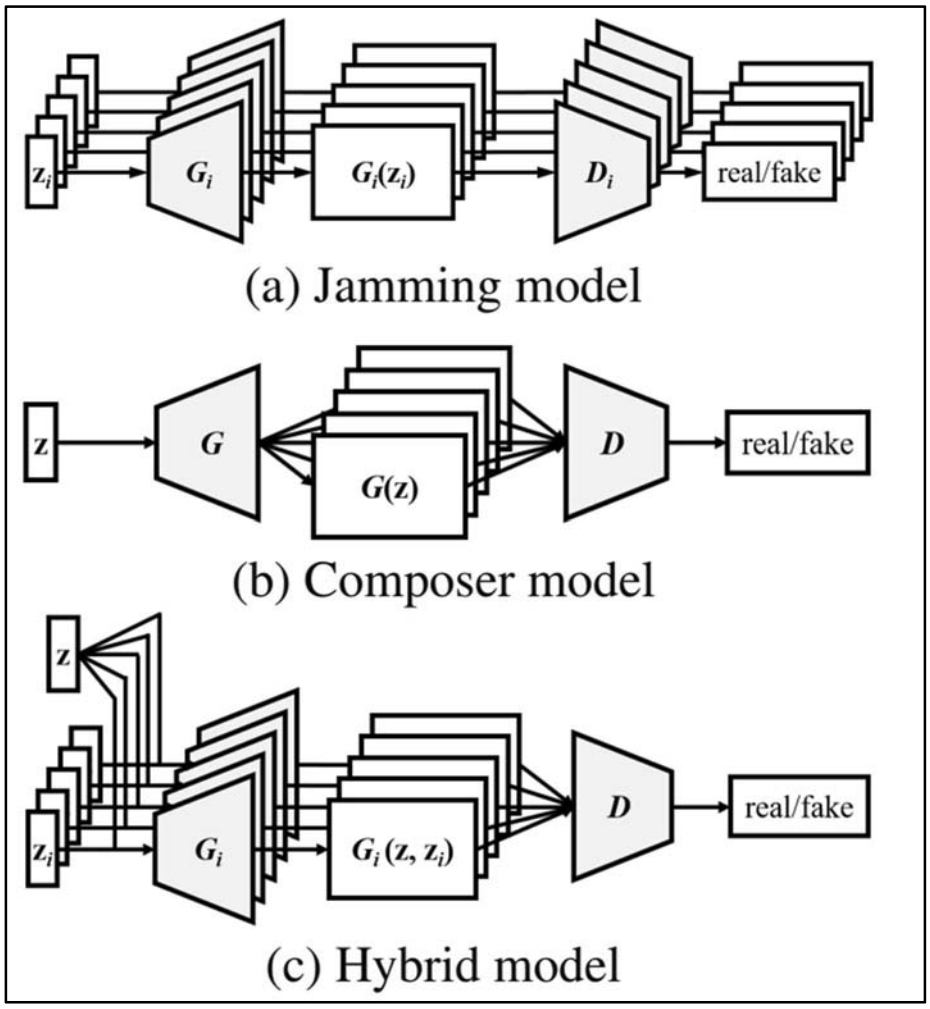


Рисунок 8 – Архитектуры моделей джема, гибридной и композитора [17]

**Вывод по первому разделу**

Были проанализированы существующие подходы к генерации мелодий. Анализ существующих решений показывает, что задача генерации музыки с применением нейронных сетей является актуальной. Для реализации системы генераций мелодии будет использован подход, использующий модель на основе GAN, т.к. модели, основанные на данной архитектуре, показывают наиболее оригинальные результаты.

# 2. ПРОЕКТИРОВАНИЕ ПРИЛОЖЕНИЯ

К разработке планируется веб-приложение MusicApp, способное генерировать мелодию с помощью нейронных сетей. MusicApp будет давать возможность обратиться к сгенерированным мелодиям в любое время и обладать функционалом для корректировки созданной мелодии.

## 2.1. Выявление требований

**Функциональные требования**

В ходе анализа мною были выделены следующие функциональные требования к приложению MusicApp:

1) пользователь MusicApp должен иметь возможность сгенерировать музыкальную партитуру;

2) пользователь MusicApp должен иметь возможность просмотреть ранее сгенерированные музыкальные партитуры;

3) пользователь MusicApp должен иметь возможность настроить параметры генерации музыкальной партитуры;

4) пользователь должен иметь возможность регистрации и авторизации в приложении MusicApp;

5) 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. На рисунке 9 представлена диаграмма вариантов использования приложения MusicApp.

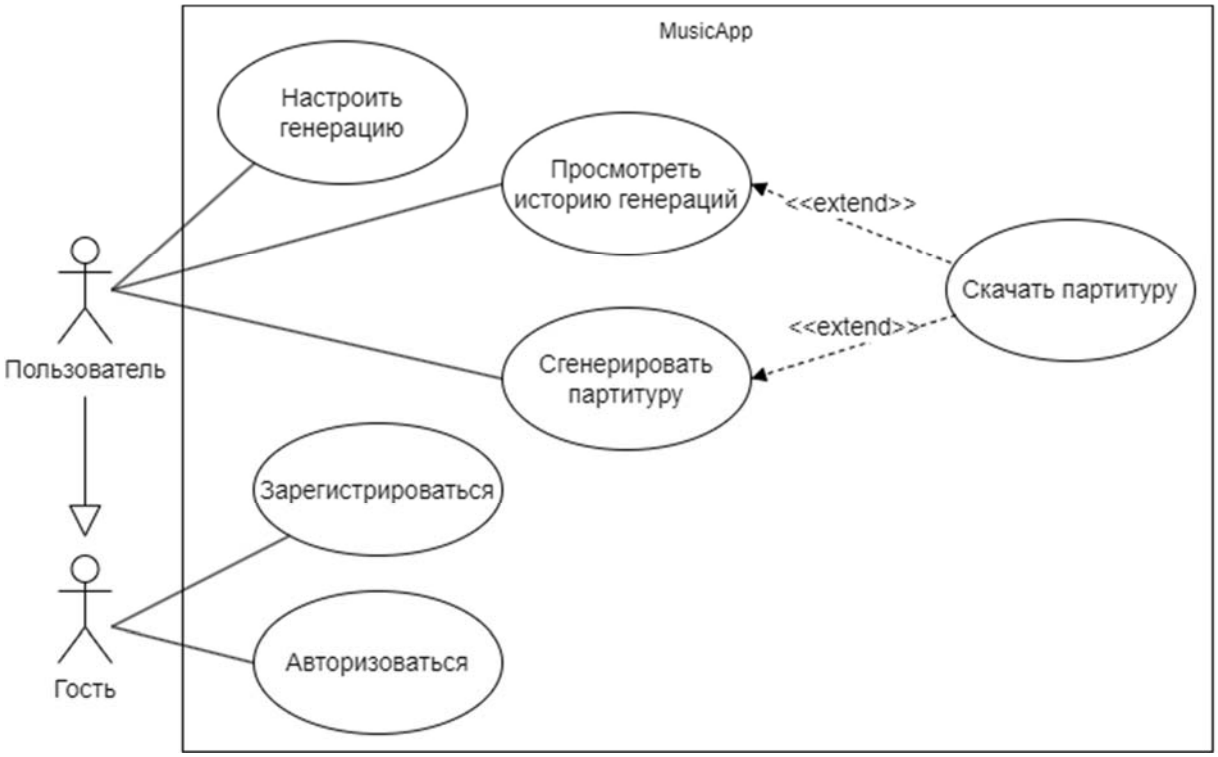


Рисунок 9 – Диаграмма вариантов использования приложения

С приложением MusicApp взаимодействует 2 актера.

1. Гость – человек, который не авторизовался в приложении.
2. Пользователь – человек, который авторизовался в приложении и использует его для генерации мелодий.

Гость может зарегистрироваться в приложении.

Гость может авторизоваться в приложении.

Пользователь может сгенерировать партитуру, то есть создать midi файл со сгенерированной мелодией.

Пользователь может просмотреть когда-либо сгенерированные им мелодии.

Пользователь может скачать сгенерированный midi файл.

Пользователь может настроить генерацию, то есть зафиксировать какие-либо из векторов шумов, тем самым управляя характеристиками мелодии.

## 2.2. Проектирование приложения

### Компоненты приложения MusicApp

Разрабатываемая система состоит из двух основных компонентов: серверной части и web-приложения. Диаграмма размещения представлена на рисунке 10.

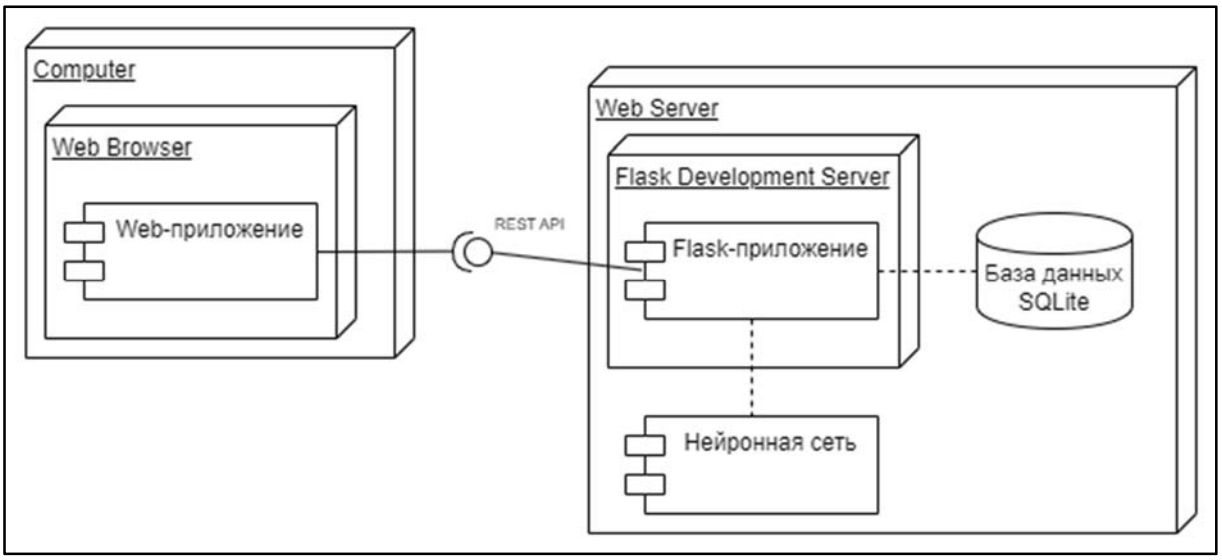


Рисунок 10 – Диаграмма размещения приложения MusicApp

Серверная часть включает в себя следующие компоненты:

1) Flask приложение, работающее на Flask Development Server, который принимает входящие HTTP запросы и передает их напрямую во Flask приложение;

2) база данных SQLite, которая хранит информацию о пользователях приложения;

3) нейронная сеть, к которой Flask-приложение обращается напрямую.

Клиентская часть представляет собой web-приложение, запущенное в любом браузере и общающееся с сервером через REST API.

**Проектирование базы данных**

Для работы сервиса была спроектирована схема базы данных, содержащая аутентификационные данные пользователей приложения, историю их генераций, включая информацию о параметрах, при которых производилась генерация, и информацию о тарифах приложения.

На рисунке 11 показана схема базы данных приложения MusicApp.

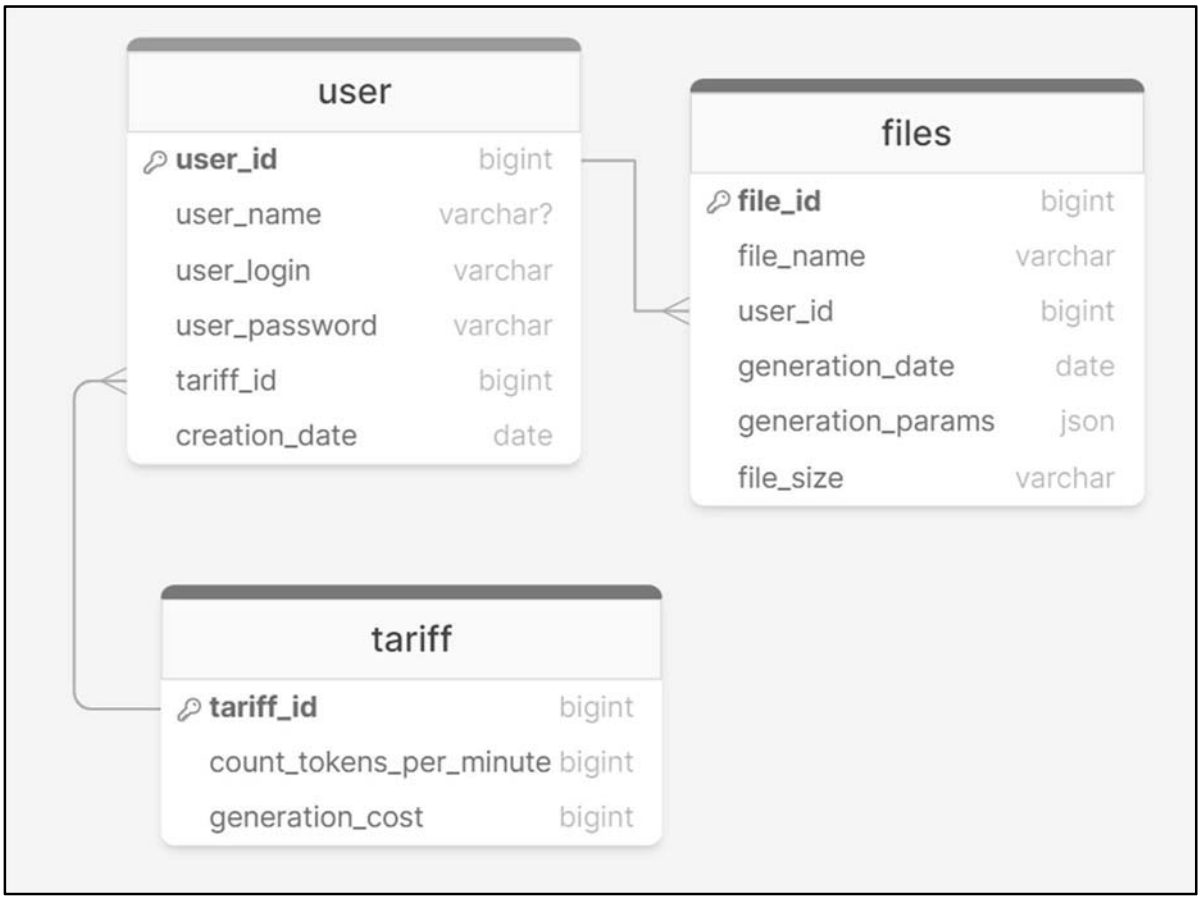


Рисунок 11 – Схема базы данных

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) tariff_id – идентификатор примененного тарифа генерации;
6) creation_date – дата регистрации пользователя.

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;
6) file_size – размер файла с указанием единицы измерения.

**Вывод по второму разделу**

В данном разделе были выделены функциональные и нефункциональные требования к приложению, определены и описаны варианты использования, описаны компоненты приложения, а также спроектирована схема базы данных.

# 3. ПОДГОТОВКА НАБОРА ДАННЫХ

## 3.1. Формат входных 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.

### 3.2. Описание набора данных

В качестве набора данных был использован набор Lakh MIDI Dataset [18], содержащий множество MIDI файлов различных жанров музыки. На рисунке 12 представлена часть информации из midi файла.

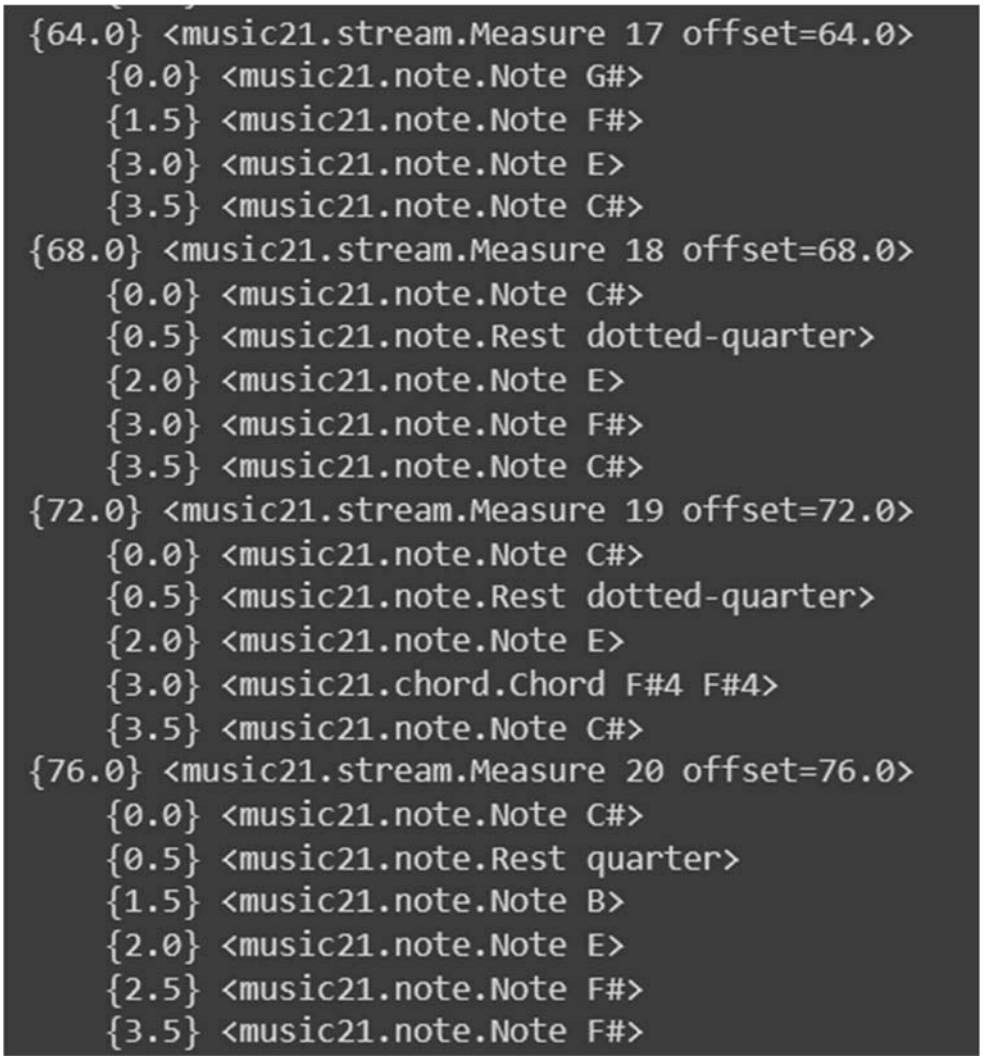


Рисунок 12 – Набор данных

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.

### 3.3. 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 в виде букв латинского алфавита с указанием октавы, поэтому каждая нота была закодирована своим числом.

На рисунке 13 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 13 – Удаление лишних данных

Каждый файл был закодирован в векторы, длина которых кратна числу временных шагов в такте, которое равно 4, умноженное на число извлеченных тактов. Таким образом, при извлечении нескольких дорожек получается матрица $M \times N$, где $M$ – число дорожек, $N$ – 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. Этот массив имеет форму $(N, S, T)$, где $N$ – число извлеченных мелодий, $S$ – число извлеченных временных шагов, $T$ – число дорожек в извлеченных мелодиях. Эти данные будут подаваться в модель как эталон, опираясь на который сеть будет обучаться генерировать данные, похожие на те, что содержит набор данных, но не идентичные им.

**Вывод по третьему разделу**

В данном разделе были проанализированы существующие форматы представления аудиоданных, в результате был выбран MIDI формат для набора данных, в соответствии с выбранным форматом был подготовлен и описан набор данных, а также было выполнено его преобразование в векторный вид.

# 4. РЕАЛИЗАЦИЯ НЕЙРОСЕТЕВОЙ МОДЕЛИ

Нейросетевая модель будет обучена с использованием набора данных, описанного в третьем разделе. Она будет использоваться на серверной части для генерации мелодии.

## 4.1. Архитектура нейронной сети

Выбранная архитектура модели представляет собой упрощенную адаптацию архитектуры MuseGAN, представленной в статье [17], 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 14 изображена архитектура реализованной модели.

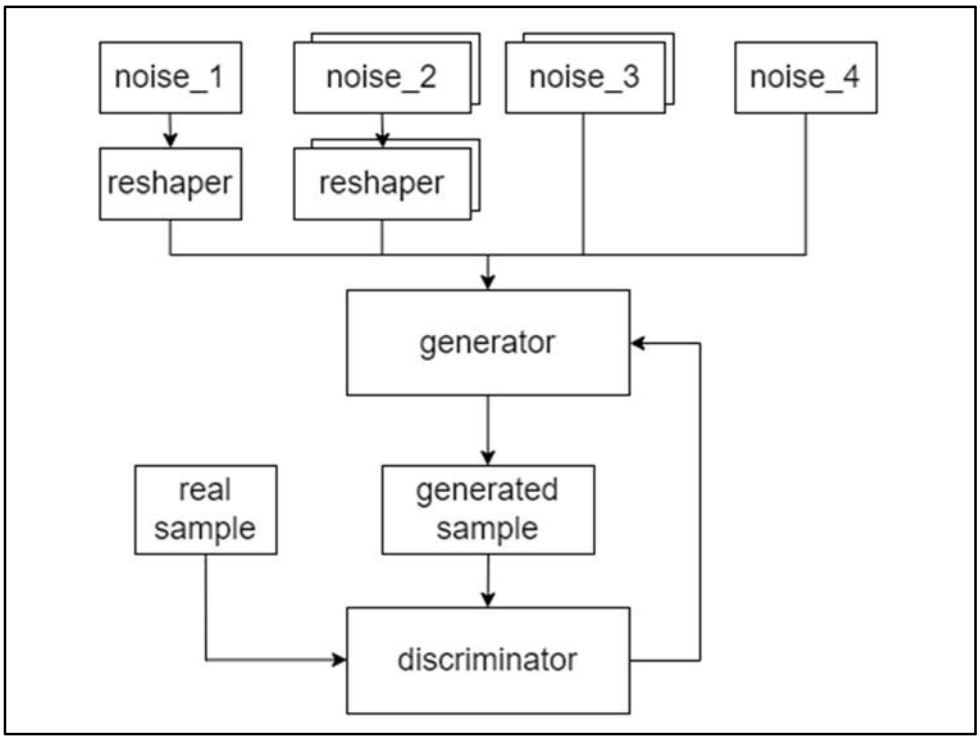


Рисунок 14 – Архитектура модели

**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, размер-

ность которого равна $(N, S)$, где $N$ – размерность вектора, в который закодирована нота, $S$ – число нот в такте. Предварительно вектор шума проходит через полносвязный слой. Программный код, демонстрирующий их структуру представлен на рисунке 15.

```
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 15 – 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 16.

```
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 16 – 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 [19], 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 [20] 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.

### 4.2. Анализ обучения модели

На рисунке 17 представлены графики потерь генератора и дискриминатора. На которых можно увидеть, как изменялись значения потерь во время обучения.

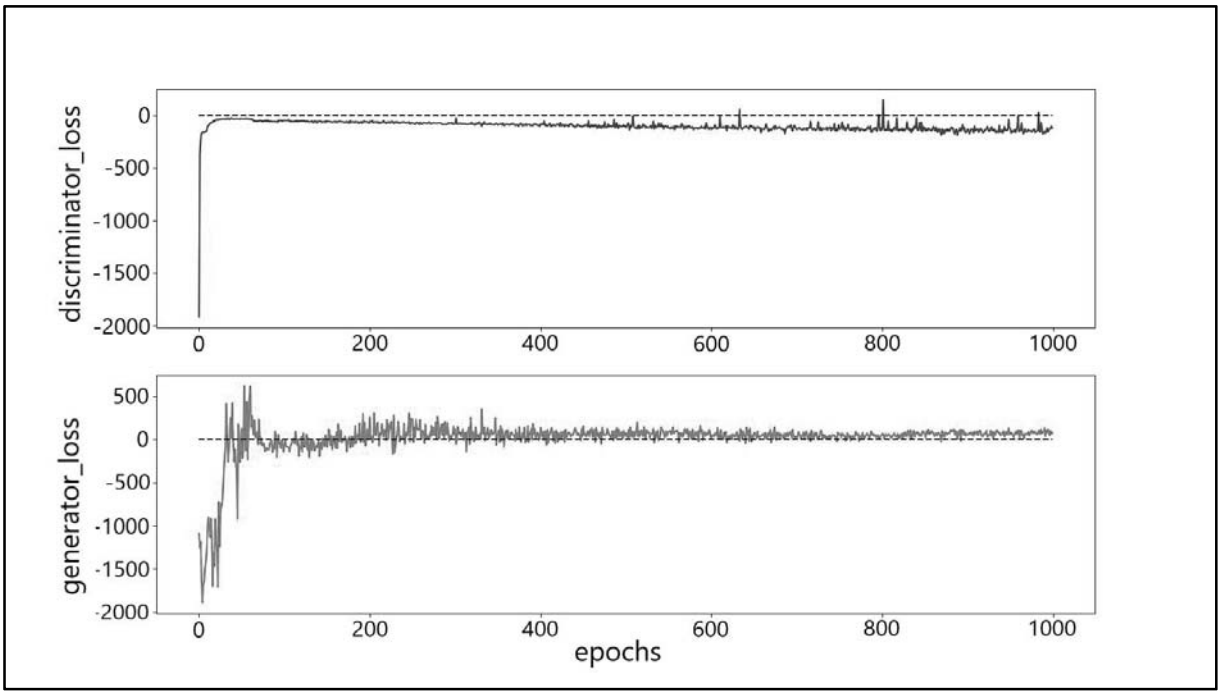


Рисунок 17 – Графики потерь модели

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.

**Вывод по четвертому разделу**

В данном разделе была реализована и обучена нейронная сеть для генерации музыкальных партитур на наборе данных, описанном в третьем разделе. Был проведен анализ обучения модели.

# 5. РАЗРАБОТКА ПРИЛОЖЕНИЯ

Для реализации серверной части приложения MusicApp был использован язык программирования Python 3, для работы с базой данных использовалась СУБД SQLite. Для верстки использовался фреймворк Bootstrap 5, язык программирования JavaScript, шаблонизатор Jinja, а также языки HTML5 и CSS3. Использованная версия языка Python – 3.11.

В процессе разработки были использованы следующие библиотеки.

1. SQLAlchemy 2.0.30 [21] – библиотека для работы с базами данных и отображения объектов на реляционные таблицы. Обеспечивает удобный способ взаимодействия с реляционными базами данных, абстрагируя различия между различными СУБД.
2. Tensorflow 2.15.0 [22] – открытая платформа для машинного обучения, предлагает обширную экосистему инструментов и библиотек для различных задач.
3. Music21 9.1.0 [23] – набор инструментов для анализа музыкальных файлов, визуализации нотного текста, обработки файлов MIDI.
4. Flask 3.0.3 [24] – фреймворк для создания веб-приложений и API с минимальным количеством избыточного кода. Flask предоставляет функции, такие как маршрутизация URL, рендеринг шаблонов, обработка запросов и управление сессиями.

## 5.1. Авторизация пользователя

Страница авторизации была реализована в виде формы с 2 кнопками – «Авторизация» и «Регистрация». При нажатии на кнопку регистрации происходит перенаправление на страницу регистрации, на ней находится форма с обязательными полями «Логин» и «Пароль». При нажатии на кнопку «Завершить регистрацию» страница посылает POST запрос на сервер, который обрабатывается декоратором route, представленным на рисунке 18.

```
@app.route('/register', methods=['POST', 'GET'])
def register():
    if request.method == 'POST':
        try:
            password_hash = generate_password_hash(
                    request.form["password"]
            )
            login = request.form["login"]
            username = request.form["username"]

            user_is_exists = db.session.execute(text(f"SELECT COUNT(*)
                                        FROM user
                                        WHERE user_login == :login"),
                                    {"login": login}).all()
            if (user_is_exists[0][0] == 0):
                new_user = User(user_name=username, user_login=login,
                                user_password=password_hash, tariff_id=1)
                db.session.add(new_user)
                db.session.commit()
                os.makedirs(os.path.join('./static/files', login))
                return redirect("/")
            else:
                return render_template("register.html", mes-
sage="user_exist")
        except:
            return render_template("register.html", mes-
sage="invalid_data")
    else:
        return render_template("register.html")
```

Рисунок 18 – Обработка POST запроса с формы регистрации

Со страницы приходят внесенные данные, из пароля извлекается хэш с помощью функции `generate_password_hash` библиотеки `Werkzeug` [25]. Затем выполняется sql запрос, возвращающий число пользователей с указанным логином, и, если таких не существует, то в базу добавляется новый пользователь, создается директория для хранения сгенерированных данным пользователем файлов и происходит перенаправление обратно на страницу авторизации, в иных случаях выводится соответствующее сообщение.

Обработка POST запроса с формы авторизации происходит следующим образом – пользователь проверяется на существование, затем функцией `check_password_hash` библиотеки `Werkzeug` извлекается хэш пароля, полученного с формы и хэш, хранящийся в базе данных, в случае если они

совпадают, извлекаются данные авторизованного пользователя и происходит перенаправление на главную форму.

Программный код обработки POST запроса с формы авторизации представлен в приложении.

### 5.2. Главная страница приложения

На главной странице находится интерфейс для генерации партитуры. При нажатии на кнопку «Сгенерировать партитуру» блокируются все элементы и происходит вызов функции `generate_music`, которая, используя обученную модель, генерирует midi файл. Ее код представлен в приложении.

Функция преобразует выход генератора, сгенерированный на основе векторов шумов, в матрицу, имеющую размерность, равную числу дорожек по одной оси и числу сгенерированных временных шагов по другой. Затем, используя созданный в процессе предобработки файл с тэгами, функция проходит вдоль оси временных шагов и декодирует ноты, создавая на их основе объекты `Note` и `Rest` библиотеки `music21`, которые записывает в объекты `Parts`, после созданные объекты `Parts` сохраняются в объект `Score`, который сохраняется как midi файл в директории пользователя.

После успешной генерации становится доступной кнопка «Скачать», по нажатию, на которую выполняется функция `send_file` библиотеки `flask`.

При установке чекбоксов на форме, замораживаются соответствующие вектора шумов предыдущей генерации, которые остаются постоянными на протяжении того времени, пока чекбоксы активны. Таким образом пользователь имеет возможность управлять характеристиками генерируемой партитуры, то есть оставлять понравившийся ему мотив, а другие аспекты мелодии генерировать снова. Так, например, при установке всех чекбоксов сгенерируется в точности та же мелодия.

Также, для ограничения траты ресурсов сервера пользователями, реализована система токенов и тарифов, которая автоматически начисляет токены спустя определенные промежутки времени и расходует имеющиеся на каждую генерацию. Текущее число токенов пользователя рассчитывается перед каждым началом генерации кодом, представленным на рисунке 19, как разница между суммой начисленных токенов за все время и суммой израсходованных за все время.

```
date_difference = datetime.now() -
                  datetime.strptime(current_user.creation_date,
                                    '%Y-%m-%d %H:%M:%S.%f')
count_minutes = int(date_difference.days*24*60 +
                    date_difference.seconds/60)
current_user_tokens = count_minutes *
                      current_user.count_tokens_per_minute -
                      current_user.count_generated_files *
                      current_user.generation_cost
current user.update current tokens(current user tokens)
```

Рисунок 19 – Расчет токенов пользователя

Информация о числе начисляемых и расходуемых токенов зависит от выбранного пользовательского тарифа.

Внешний вид главной формы представлен на рисунке 20.

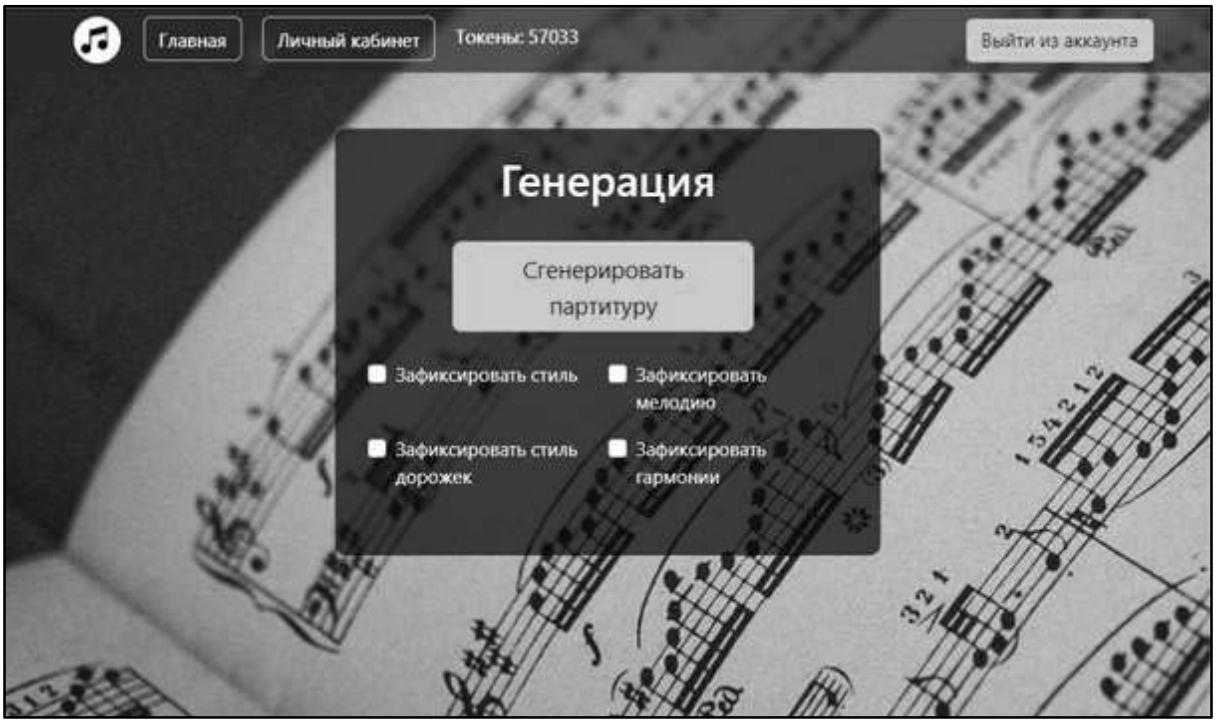


Рисунок 20 – Главная форма

### 5.3. Личный кабинет пользователя

Из главной формы доступен личный кабинет пользователя, в котором есть функционал установки нового пароля и имени пользователя, а также просмотра истории генераций с возможностью просмотреть и скачать ранее сгенерированные midi файлы.

Вкладка «История» личного кабинета представлена как список форм, на которых отображена информация о файле и кнопка «Скачать».

На рисунке 21 показан скриншот истории генераций.

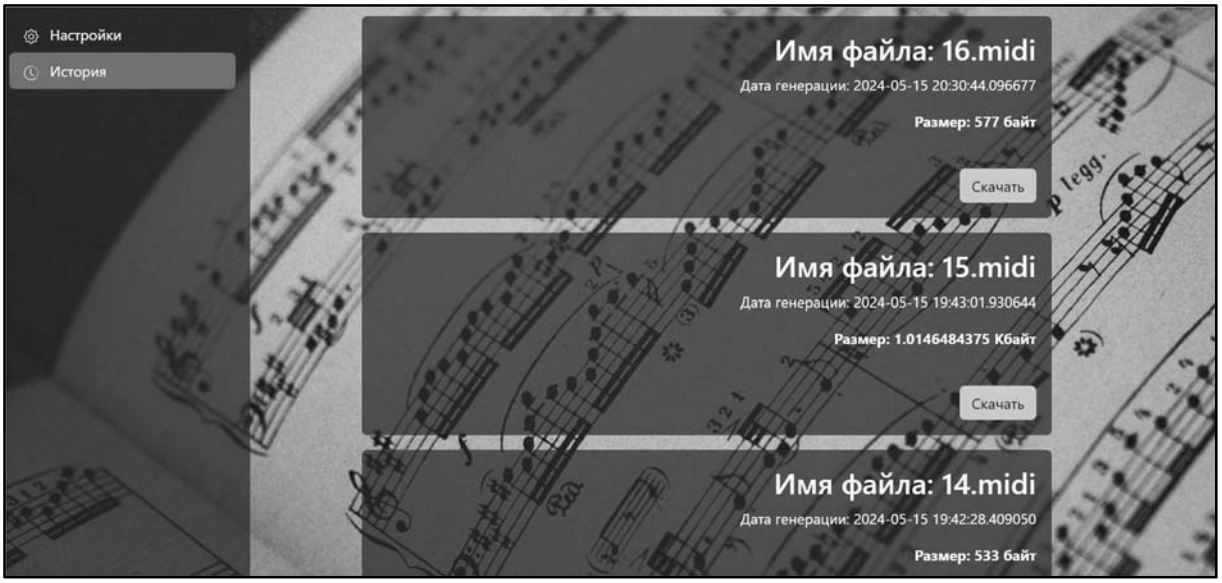


Рисунок 21 – История генераций

История генераций формируется, используя данные запроса на получение списка файлов пользователя к базе, формы создаются с помощью шаблонизатора `Jinja` для каждой записи из запроса, при клике на кнопку «Скачать» посылается POST запрос с соответствующим именем файла, далее по имени файла и имени пользователя вызывается функция `send_file` библиотеки `flask`.

Код, обрабатывающий запросы для страницы истории генераций представлен на рисунке 22.

```
@app.route('/lk_history', methods=['GET', 'POST'])
def lk_history():
    if request.method == 'POST':
        try:
            filename = request.form["filename"]
            user_files_path = os.path.join("./static/files",
                                           current_user.user_login)
            return send_file(os.path.join(user_files_path,
                                          filename),
                             as_attachment=True)
        except:
            return redirect("/")
    else:
        files = Files.query.order_by(Files.generation_date.desc()).where(
            Files.user_id == current_user.user_id).all()
        return render_template("lk_history.html",
                               tokens=current_user.user_current_tokens,
                               files=files)
```

Рисунок 22 – Обработка запросов для страницы истории генераций

На вкладке «Настройки» находится форма с 2 полями для изменения имени пользователя и его пароля, а также кнопка применить, которая записывает изменения в базу.

При вводе значений на вкладке и нажатии на кнопку «Применить» посылается POST запрос, который обрабатывается следующим образом – если значение не было введено, то соответствующая запись в базе не изменяется, если же значение было введено, то для имени оно добавляется в базу, для пароля добавляется его хэш.

При переходе на вкладку настройки открывается форма с полями «Имя пользователя» и «Пароль». При вводе значений на вкладке и нажатии на кнопку «Применить» посылается POST запрос, который обрабатывается следующим образом – если значение не было введено, то соответствующая запись в базе не изменяется, если же значение было введено, то для имени оно добавляется в базу, для пароля добавляется его хэш.

Код, обрабатывающий запросы с вкладки настроек, представлен на рисунке 23.

```
@app.route('/lk_settings', methods=['POST', 'GET'])
def lk_settings():
    if request.method == 'POST':
        try:
            username = request.form["username"]
            password = request.form["password"]

            if ((username != "" and username is not None) or
                    (password != "" and password is not None)):
                user = User.query.get(current_user.user_id)
                if (username != "" and username is not None):
                    current_user.update_username(username)
                    user.user_name = username
                if (password != "" and password is not None):
                    password_hash = generate_password_hash(password)
                    user.user_password = password_hash
                db.session.commit()
            return redirect("/lk_settings")
        except:
            return redirect("/lk_settings")
    else:
        user_name = f", {current_user.user_name}" if \
            (current_user.user_name != "" and
             current_user.user_name is not None) \
            else ""
        return render_template("lk_settings.html",
                               tokens=current_user.user_current_tokens,
                               user_name=user_name)
```

Рисунок 23 – Обработка запросов для вкладки настроек

**Вывод по пятому разделу**

В данном разделе было реализовано web-приложение для генерации музыки, которое взаимодействует с нейронной сетью, реализованной в предыдущем разделе.

# 6. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

## 6.1. Функциональное тестирование

В ходе данного тестирования проверялось соответствие приложения функциональным требованиям. В таблице 1 представлены результаты тестирования.

Таблица 1 – Функциональное тестирование приложения

| № | Название теста | Действия | Результат |
|---|---|---|---|
| 1 | Открытие страницы регистрации | Нажать на кнопку «Регистрация» | Открывается страница регистрации |
| 2 | Отсутствие ввода требуемого поля на странице регистрации | Ввести значение для поля логин, нажать кнопку «Завершить регистрацию» | Показывается подсказка о необходимости заполнения поля |
| 3 | Ввод логина существующего пользователя на странице регистрации | Ввести логин существующего пользователя, ввести случайный пароль, нажать кнопку «Завершить регистрацию» | Показывается сообщение «Пользователь уже существует» |
| 4 | Ввод логина несуществующего пользователя на странице регистрации | Ввести логин несуществующего пользователя, ввести случайный пароль, нажать кнопку «Завершить регистрацию» | Перенаправление на форму авторизации |
| 5 | Ввод логина несуществующего пользователя на странице авторизации | Ввести логин несуществующего пользователя, ввести случайный пароль, нажать кнопку «Авторизация» | Показывается сообщение «Указанного пользователя не существует» |
| 6 | Ввод логина существующего пользователя с неверным паролем на странице авторизации | Ввести логин существующего пользователя, ввести случайный пароль, нажать кнопку «Авторизация» | Показывается сообщение «Неверный пароль» |
| 7 | Ввод логина существующего пользователя с верным паролем на странице авторизации | Ввести логин существующего пользователя, ввести его пароль, нажать кнопку «Авторизация» | Перенаправление на главную форму |
| 8 | Генерация | Нажать кнопку «Сгенерировать партитуру» | Блокируются все элементы, появляется заполняющийся progress bar, через время кнопки становятся активны, появляется кнопка «Скачать», убавляются токены |

Окончание таблицы 1

| № | Название теста | Действия | Результат |
|---|---|---|---|
| 9 | Скачивание сгенерированного файла | Нажать кнопку «Скачать» | Скачивается последний сгенерированный файл |
| 10 | Корректность работы настроек генерации | Установить все чекбоксы активными, нажать кнопку «Сгенерировать партитуру» | Сгенерированный файл идентичен предыдущему |
| 11 | Начисление токенов | Подождать минуту | Количество токенов увеличилось |
| 12 | Генерация при отсутствии токенов | Нажать кнопку «Сгенерировать партитуру» при числе токенов равном 0 | Показывается сообщение «Недостаточно токенов» |
| 13 | Переход на страницу личного кабинета | Нажать на кнопку шапки «Личный кабинет» | Перенаправление на страницу личный кабинет, на вкладке история отображается история генерации текущего пользователя |
| 14 | Скачивание файла с вкладки история | Нажать на кнопку «Скачать» у любого файла на вкладке истории | Скачивается необходимый файл |
| 15 | Пустые значения на вкладке настроек | Нажать кнопку «Применить» на вкладке настроек, не заполняя поля | Нет изменений |
| 16 | Настройка имени пользователя | Нажать кнопку «Применить» на вкладке настроек, заполнив поле «Имя пользователя» | Приветственное сообщение изменилось |
| 17 | Выход из аккаунта | Нажать кнопку «Выйти из аккаунта» | Перенаправление на страницу авторизации |
| 18 | Настройка пароля пользователя | Нажать кнопку «Применить» на вкладке настроек, заполнив поле «Пароль», выйти из аккаунта, войти под старым паролем | Показывается сообщение «Неверный пароль» |

**Вывод по шестому разделу**

В данном разделе было протестирована функциональность реализованного приложения, в ходе тестирования не было обнаружено ошибок в его работе.

## ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано веб-приложение для генерации музыки. В ходе выполнения работы были решены следующие задачи:

1) был проведен обзор литературы по предметной области, а также существующих решений для генерации музыки;
2) были выявлены требования к разрабатываемому приложению;
3) в соответствии с требованиями была составлена диаграмма вариантов использования;
4) была спроектирована схема базы данных для хранения данных пользователей и сгенерированных ими партитур;
5) был подготовлен и предобработан набор данных;
6) была выбрана архитектура нейронной сети;
7) была реализована и обучена нейросетевая модель;
8) было реализовано веб приложение;
9) было проведено тестирование веб приложения.

Планируется дальнейшее развитие проекта, включающее в себя пункты, представленные ниже:

1) увеличение качества и количества данных в наборе;
2) обучение модели в течении более длительного времени;
3) разделение серверной части на Docker контейнеры для обеспечения устойчивости и масштабируемости приложения;
4) изменение протокола передачи данных на HTTPS.

**ЛИТЕРАТУРА**

1. Jukebox. [Электронный ресурс] URL: https://openai.com/research/jukebox (дата обращения: 15.01.2024 г.).

2. Musenet. [Электронный ресурс] URL: https://openai.com/research/musenet (дата обращения: 15.01.2024 г.).

3. 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 [Электронный ресурс] // arXiv.org. 2023. Дата обновления: 26.01.2023 г. URL: https://arxiv.org/abs/2301.11325 (дата обращения: 15.01.2024 г.).

4. 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.).

5. Pawar M. A., Bewoor M. S., Patil S., Patil M. S. S., Jadhav B. R., Kadam A. K. Music Generation using RNN-LSTM with GRU. – 2023. – 6 p.

6. Ens J., Pasquier P. MMM: Exploring conditional multi-track music generation with the transformer [Электронный ресурс] // arXiv.org. 2020. Дата обновления: 20.08.2020 г. URL: https://arxiv.org/abs/2008.06048 (дата обращения: 15.01.2024 г.).

7. Zhang H., Xie L., Qi K. Implement music generation with gan: A systematic review // 2021 International Conference on Computer Engineering and Application (ICCEA). – IEEE, 2021. – p. 352-355.

8. Dadman S., Bremdal B. A. Multi-agent Reinforcement Learning for Structured Symbolic Music Generation // International Conference on Practical Applications of Agents and Multi-Agent Systems. – Cham: Springer Nature Switzerland, 2023. – p. 52-63.

9. Фостер Д. Генеративное глубокое обучение. Творческий потенциал нейронных сетей. – СПб.: Питер, 2020. – 352 с.

10. Nosouhian S., Nosouhian F., Khoshouei A. K. A review of recurrent neural network architecture for sequence learning: Comparison between LSTM and GRU. – 2021. – 7 p.

11. 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.).

12. Mangal S., Modak R., Joshi P. LSTM Based Music Generation System [Электронный ресурс] // arXiv.org. 2019. Дата обновления: 02.08.2019 г. URL: https://arxiv.org/abs/1908.01080 (дата обращения: 15.01.2024 г.).

13. Vaswani, A., Shazeer, N.M., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I. Attention is All you Need [Электронный ресурс] // arXiv.org. 2017. Дата обновления: 02.08.2023 г. URL: https://arxiv.org/abs/1706.03762 (дата обращения: 15.01.2024 г.).

14. Тимаков К. А. Сравнение актуальных языковых моделей Google Bard и ChatGPT. // Современные стратегии и цифровые трансформации устойчивого развития общества, образования и науки. – 2023. – с. 168-171.

15. The MAESTRO Dataset and Wave2Midi2Wave. [Электронный ресурс] URL: https://magenta.tensorflow.org/maestro-wave2midi2wave (дата обращения: 15.01.2024 г.).

16. 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 [Электронный ресурс] // arXiv.org. 2023. Дата обновления: 25.04.2023 г. URL: https://arxiv.org/abs/2304.12995 (дата обращения: 15.01.2024 г.).

17. 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 [Электронный ресурс] //

arXiv.org. 2017. Дата обновления: 24.11.2017 г. URL: https://arxiv.org/abs/1709.06298 (дата обращения: 15.01.2024 г.).

18. Lakh MIDI dataset. [Электронный ресурс] URL: https://colinraffel.com/projects/lmd/ (дата обращения: 03.04.2024 г.).

19. Arjovsky M., Chintala S., Bottou L. Wasserstein GAN [Электронный ресурс] // arXiv.org. 2017. Дата обновления: 06.12.2017 г. URL: https://arxiv.org/abs/1701.07875 (дата обращения: 12.04.2024 г.).

20. Gulrajani I., Ahmed F., Arjovsky M., Dumoulin V., Courville A. Improved Training of Wasserstein GANs [Электронный ресурс] // arXiv.org. 2017. Дата обновления: 25.12.2017 г. URL: https://arxiv.org/abs/1704.00028 (дата обращения: 12.04.2024 г.).

21. SQLAlchemy 2.0 documentation. [Электронный ресурс] URL: https://colinraffel.com/projects/lmd/ (дата обращения: 20.04.2024 г.).

22. Tensorflow. [Электронный ресурс] URL: https://www.tensorflow.org/ (дата обращения: 20.04.2024 г.).

23. Music21 Documentation. [Электронный ресурс] URL: https://web.mit.edu/music21/doc/ (дата обращения: 20.04.2024 г.).

24. Flask documentation. [Электронный ресурс] URL: https://flask.palletsprojects.com/en/3.0.x/ (дата обращения: 20.04.2024 г.).

25. Werkzeug documentation. [Электронный ресурс] URL: https://werkzeug.palletsprojects.com/en/3.0.x/ (дата обращения: 20.04.2024 г.).

## ПРИЛОЖЕНИЕ. Листинги функций

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 isin-
stance(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)
```

```
        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
        ))
    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)
                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}
```

Листинг 5 – Обработка POST запроса с формы авторизации

```
@app.route('/', methods=['POST', 'GET'])
def auth():
    if request.method == 'POST':
        try:
            password = request.form["password"]
            login = request.form["login"]

            user_is_exists = db.session.execute(text(f"SELECT COUNT(*) "
                                                     f"FROM user "
                                                     f"WHERE user_login == :login"),
                                                {"login": login}).all()
            if (user_is_exists[0][0] == 0):
                return render_template("auth.html",
                                       message="user_not_exists")
            else:
                user_password = db.session.execute(text(f"SELECT us-
er_password "
                                                     f"FROM user "
                                                     f"WHERE user_login == :login"),
                                                {"login": login}).all()
                if check_password_hash(user_password[0][0], password):
                   user_id = db.session.execute(text(f"SELECT user_id "
                                                     f"FROM user "
                                                     f"WHERE user_login == :login"),
                                                {"login": login}).all()
                   user_name = db.session.execute(text(f"SELECT user_name "
```

Окончание листинга 5 приложения

```
                                        f"FROM user "
                                        f"WHERE user_login == :login"),
                                      {"login": login}).all()
                    generation_cost = db.session.execute(text(f"SELECT gen-
eration_cost "
                                       f"FROM user "
                                       f"  JOIN tariff USING(tariff_id)
"
                                        f"WHERE user_login == :login"),
                                     {"login": login}).all()
                    creation_date = db.session.execute(text(f"SELECT crea-
tion_date "
                                        f"FROM user "
                                        f"WHERE user_login == :login"),
                                     {"login": login}).all()
                    count_generated_files = db.session.execute(text(
                                        f"SELECT COUNT(*) "
                                        f"FROM user "
                                        f"  JOIN files USING(user_id) "
                                        f"WHERE user_login == :login"),
                                     {"login": login}).all()
                    count_tokens_per_minute = db.session.execute(text(
                                     f"SELECT count_tokens_per_minute "
                                     f"FROM user "
                                     f"  JOIN tariff USING(tariff_id) "
                                     f"WHERE user_login == :login"),
                                  {"login": login}).all()
                    current_user.set_new_user_info(user_id[0][0],
                                        user_name[0][0],
                                        login,
                                        0,
                                        generation_cost[0][0],
                                        creation_date[0][0],
                                        count_generated_files[0][0],
                                        count_tokens_per_minute[0][0])
                    current_file.set_new_file_info(None,
                                             None,
                                             user_id,
                                             None,
                                             None,
                                             None)
                    return redirect("/main")
                else:
                    return render_template("auth.html",
                                           message="invalid_password",
                                           login=login)
            except:
                return render_template("auth.html",
                                       message="invalid_data")
        else:
            return render_template("auth.html")
```

Листинг 6 – Функция `generate_music`

```
def generate_music(model, tags, file_path):
    prev_noise = [None] * 4
    if (current_file.file_id != None):
        prev_noise = [np.array(noise) for noise in cur-
rent_file.generation_params]
```

```
    notes_set = set()
    while len(notes_set) < 5:
        if (prev_noise[0] is not None and not help_class.freeze_noise[0]):
            chords_noise = np.random.normal(size=(1, 32))
        else:
            if prev_noise[0] is None:
                chords_noise = np.random.normal(size=(1, 32))
            else:
                chords_noise = prev_noise[0]

        if (prev_noise[0] is not None and not help_class.freeze_noise[1]):
            style_noise = np.random.normal(size=(1, 32))
        else:
            if prev_noise[0] is None:
                style_noise = np.random.normal(size=(1, 32))
            else:
                style_noise = prev_noise[1]

        if (prev_noise[0] is not None and not help_class.freeze_noise[2]):
            melody_noise = np.random.normal(size=(1, 4, 32))
        else:
            if prev_noise[0] is None:
                melody_noise = np.random.normal(size=(1, 4, 32))
            else:
                melody_noise = prev_noise[2]

        if (prev_noise[0] is not None and not help_class.freeze_noise[3]):
            groove_noise = np.random.normal(size=(1, 4, 32))
        else:
            if prev_noise[0] is None:
                groove_noise = np.random.normal(size=(1, 4, 32))
            else:
                groove_noise = prev_noise[3]

        noise_vectors = [
            chords_noise,
            style_noise,
            melody_noise,
            groove_noise,
        ]

        generator_output = model.generator(noise_vectors).numpy()
        max_pitches = np.argmax(generator_output, axis=3)
        generated_notes = max_pitches.reshape([9 * 16, 4])
        notes_set = set()
        for step in generated_notes:
            notes_set = notes_set.union(set(step))
    parts = music21.stream.Score()
    parts.append(music21.tempo.MetronomeMark(number=66))

    for i in range(4):
        current_code_note = int(generated_notes[:, i][0])
        new_stream = music21.stream.Part()
        dur = 0
        for idx, code_note in enumerate(generated_notes[:, i]):
            code_note = int(code_note)
            if (code_note != current_code_note or idx % 4 == 0) and idx >
0:
                if (current_code_note == 0):
                    uncode_note = music21.note.Rest()
                else:
```

```
                    uncode_note = mu-
sic21.note.Note(tags[current_code_note])
                uncode_note.duration = music21.duration.Duration(dur)
                new_stream.append(uncode_note)
                dur = 0
            current_code_note = code_note
            dur = dur + 0.25
        if (current_code_note == 0):
            uncode_note = music21.note.Rest()
        else:
            uncode_note = music21.note.Note(tags[current_code_note])
        uncode_note.duration = music21.duration.Duration(dur)
        new_stream.append(uncode_note)

        parts.append(new_stream)
    parts.write("midi", fp=f"{file_path}")

    file_name = file_path.split('/')
    file_name = file_name[len(file_name) - 1]
    noise_vectors[0] = [list(noise_vectors[0][0])]
    noise_vectors[1] = [list(noise_vectors[1][0])]
    noise_vectors[2] = [[list(vector) for vector in noise_vectors[2][0]]]
    noise_vectors[3] = [[list(vector) for vector in noise_vectors[3][0]]]
    return list(noise_vectors), file_name, datetime.now()
```