**ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИ УНИВЕРСИТЕТ**

**«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»**

Факультет компьютерных наук

Департамент программной инженерии

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. №подл |  |

СОГЛАСОВАНО

Старший преподаватель департамента программной инженерии факультета компьютерных наук

Д.В. Пантюхин

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

УТВЕРЖДАЮ

Академический руководитель образовательной программы «Программная инженерия», к.т.н., профессор ДПИ ФКН

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.В. Шилов

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

**Серверная часть приложения для генерации музыки**

**Пояснительная записка**

**ЛИСТ УТВЕРЖДЕНИЯ**

**RU. 17701729.05.01-01 34 01-1-ЛУ**

**Исполнитель**

Студент группы БПИ181

\_\_\_\_\_\_\_\_\_\_\_\_ / М. С. Никифоров /

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

**Москва 2021**

**УТВЕРЖДЕН**

**RU. 17701729.05.01-01 34 01-1 ЛУ**

|  |  |
| --- | --- |
| Подп. и дата |  |
| Инв. № дубл. |  |
| Взам. инв. № |  |
| Подп. и дата |  |
| Инв. №подл |  |

**Серверная часть приложения для генерации музыки**

**Пояснительная записка**

**RU. 17701729.05.01-01 34 01-1**

**Листов 22**

**Москва 2021**

[1. Введение 3](#_Toc69143811)

[1.1. Наименование программы 3](#_Toc69143812)

[1.2. Документы, на основании которых ведется разработка 3](#_Toc69143813)

[2. Условия выполнения программы 4](#_Toc69143814)

[2.1. Назначение программы 4](#_Toc69143815)

[2.1.1. Функциональное назначение 4](#_Toc69143816)

[2.1.2. Эксплуатационное назначение 4](#_Toc69143817)

[2.2. Краткая характеристика области применения 4](#_Toc69143818)

[3. Технические характеристики 5](#_Toc69143819)

[3.1. Постановка задачи на разработку программы 5](#_Toc69143820)

[3.2. Описание используемого математического аппарата 5](#_Toc69143821)

[3.2.1. Генерация музыки 5](#_Toc69143822)

[3.2.2. Архитектура базы данных 9](#_Toc69143823)

[3.3. Описание алгоритма и функционала программы 9](#_Toc69143824)

[3.4. Описание и обоснование выбора технических средств 10](#_Toc69143825)

[4. Ожидаемые технико-экономические показатели 11](#_Toc69143826)

[4.1. Предполагаемая потребность 11](#_Toc69143827)

[4.2. Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами. 11](#_Toc69143828)

[5. Перспективы развития 12](#_Toc69143829)

[Список источников 13](#_Toc69143830)

[ПРИЛОЖЕНИЕ 1 14](#_Toc69143831)

# Введение

## Наименование программы

Наименование программы - «Серверная часть приложения для генерации музыки». Наименование программы на английском – «backend for application for music generation».

## Документы, на основании которых ведется разработка

Программа выполнена в рамках темы курсового проекта – «Серверная часть приложения для генерации музыки», в соответствии с учебным планом подготовки бакалавров по направлению 09.03.04 «Программная инженерия».

# Условия выполнения программы

## Назначение программы

**2.1.1. Функциональное назначение**

«Серверная часть приложения для генерации музыки» реализует основные функции пользователя, которые должно поддерживать «Приложение для генерации музыки». Приложение представляет из себя RESTful API сервис, основные функции серверной части – регистрация и авторизация пользователя, поддержка баз данных для хранения нейронных сетей для генерации музыки, поддержка и хранение плейлистов с музыкой разных стилей и разных пользователей.

**2.1.2. Эксплуатационное назначение**

«Серверная часть приложения для генерации музыки» создано для отделения сложного и тяжеловесного функционала от интерфейса и для возможности использования данного функционала в других версиях приложения на других устройствах.

## Краткая характеристика области применения

Данное программное обеспечение предназначено для создания музыкальных композиций на основе предложенных треков с использованием технологий искусственного интеллекта и систем машинного обучения. Для сохранения контента и его структуризации используется система регистрации пользователя и авторизации пользователя.

# Технические характеристики

## Постановка задачи на разработку программы

«Серверная часть приложения для генерации музыки» состоит из базы данных и набора скриптов для обработки запросов от пользователя. База данных должна состоять из следующих таблиц:

1. Таблица Users для хранения информации о пользователях;
2. Таблица Tracks для хранения музыкальных треков;
3. Таблица Playlists для хранения плейлистов;
4. Таблица Networks для хранения информации о нейронных сетях, используемых при генерации.

API должно содержать следующие функции:

1. Регистрация пользователя;
2. Авторизация пользователя;
3. Создание плейлиста;
4. Редактирование плейлиста;
5. Сгенерировать музыку;
6. Добавить трек в плейлист;
7. Вывести информацию о пользователях;
8. Вывести информацию о плейлистах;
9. Вывести информацию обо всех треках.

## Описание используемого математического аппарата

## Генерация музыки

Задача генерации музыки достаточно сложна. Представить музыку можно в двух форматах: в виде временного ряда (WAV или MP3 формат) и в виде набора команд с инструкциями о проигрывании каждой конкретной ноты (MIDI[[1]](#footnote-1) формат).

Ниже приведен пример мелодии в качестве временного ряда (рис. 1)

**Изображение выглядит как текст, оружие

Автоматически созданное описание**

Рисунок 1 – хранение мелодии в качестве временного ряда

Ниже приведен пример мелодии в MIDI формате (рис. 2)

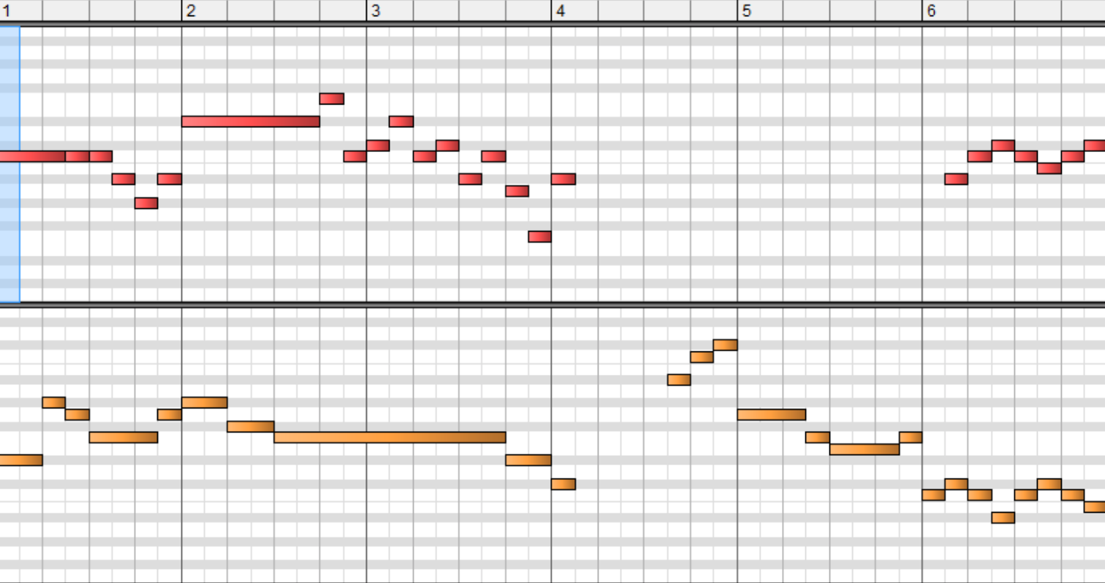


Рисунок 2 – хранение мелодии в MIDI формате

Формат MIDI искажает исходную мелодию, так как не хранит информацию об инструменте, на котором проигрывается мелодия. Потому формат MIDI использовался только при генерации классической музыки, так как в таком случае мы знаем инструмент, на котором воспроизводится мелодия.

При генерации мелодии с использованием MIDI формата задача сводится к задаче построения языковой модели. Из нот и их длинны создается набор токенов, которые предсказываем в каждый момент времени. Задача языкового моделирования хорошо изучена и существуют проверенные архитектуры нейронных сетей для ее решения, но при генерации нот и их длин существующие решения работают уже не так хорошо.

Для генерации музыки в формате MIDI лучшие результаты показывает сеть MuseNet[[2]](#footnote-2). Данная сеть была создана как аналог сети GPT2[[3]](#footnote-3), но представление исходных данных и их обработка немного изменены.

Ниже приведен пример хранения данных в MIDI формате для подачи их в сеть MuseNet (рис. 3)

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 3 – пример хранения нот в качестве токенов для входа модели

Модель MuseNet использует механизм SparseTransformers[[4]](#footnote-4) для уменьшения времени и предсказания модели.

MuseNet показывает лучшие результаты на классической музыке и даже дает концерты. Потому данная сеть будет использоваться при генерации мелодий на фортепьяно, скрипке и других классических музыкальных инструментах.

Генерация музыки с использованием представления мелодии в виде временного ряда позволяет забыть об инструментах и не потерять информацию о входных данных. Однако, задача генерации музыки становится гораздо сложнее. Каждая секунда аудиодорожки кодируется 44100 числами их диапазона от -1 до 1. При генерации мелодий придется генерировать очень длинные интервалы временного ряда. Для этого приходится хранить информацию об очень большом интервале, предшествующего сгенерированному интервалу. Были проведены эксперименты на нескольких архитектурах нейронных сетей для генерации мелодии в виде временного ряда:

1. WaveNet[[5]](#footnote-5). Данная сеть первоначально использовалась в задаче «Text to speach» для генерации речи. В основе сети лежат сверточные слои[[6]](#footnote-6), которые ищут паттерны в мелодии, тем самым изучая стиль автора, создавшего мелодию. По итогам экспериментов данная модель показала не лучшие результаты. Планируется в дальнейшем использовать данную сеть, так как есть примеры[[7]](#footnote-7) успешной генерации мелодий данной архитектурой. Также есть ряд гипотез, которые планируется проверить с данной архитектурой, например, использование wavenet блока в GAN сети в качестве дискриминатора для предотвращения затухания обучения.

Ниже приведена архитектура модели WaveNet (рис. 4)

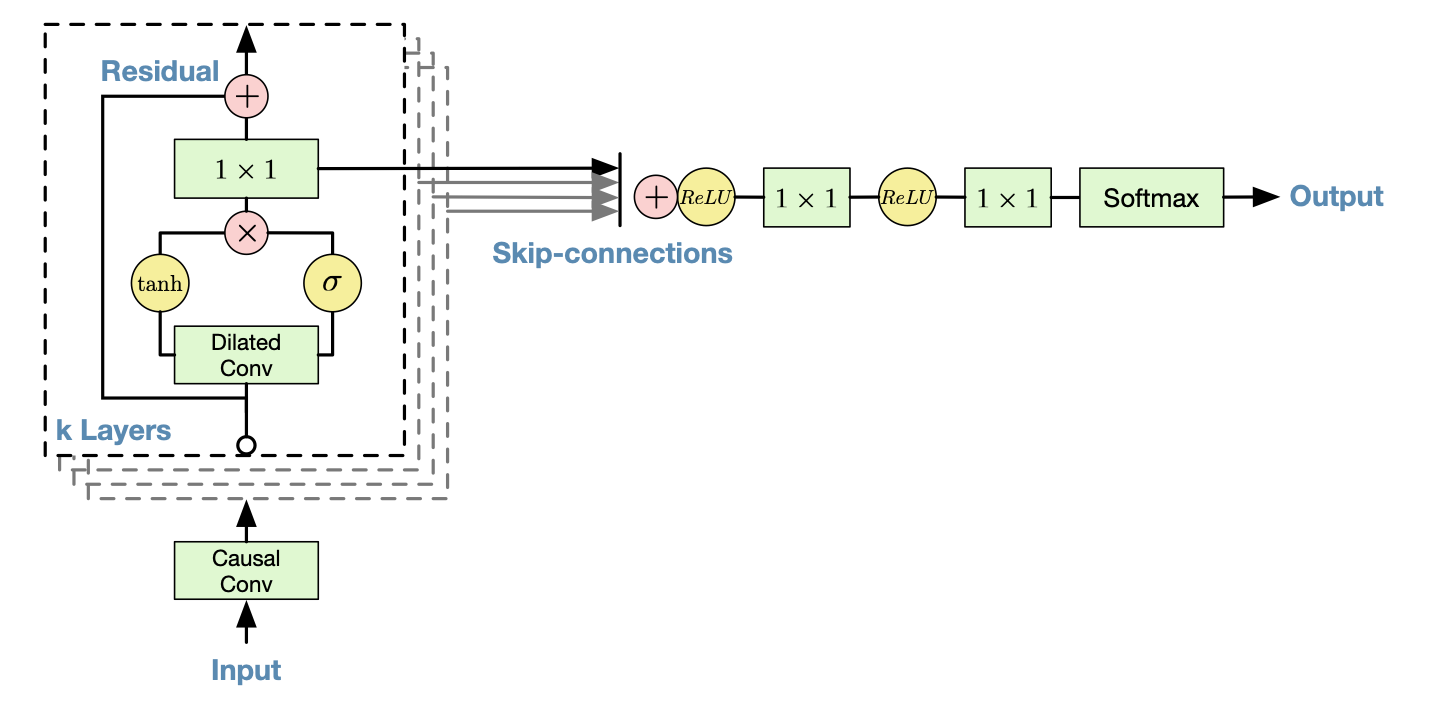


Рисунок 4 – архитектура сети WaveNet

1. GAN[[8]](#footnote-8). Генеративно-состязательная сеть. Данный метод подходит для генерации новых интервалов. Скорость обучения генератора выше, чем у WaveNet. Данный метод хорош также тем, что можно использовать разные по сложности архитектуры внутри генератора и дискриминатора.

Идея данной сети состоит в том, что модель генератора получает на вход вектор случайных значений из заранее выбранного распределения, далее генератор переводит шум в вектор, который модель дискриминатора должна отличить от реальных объектов. Таким образом, модель генератора учится получать из случайного шума объекты, похожие на объекты реального мира.

Сложность обучения данной сети состоит в том, что дискриминатор может «затухнуть». В таком случае генератор не сможет обучиться в полной мере и сгенерированные объекты не будут похожи на реальные.

Ниже приведена схема модели GAN (рис. 5)

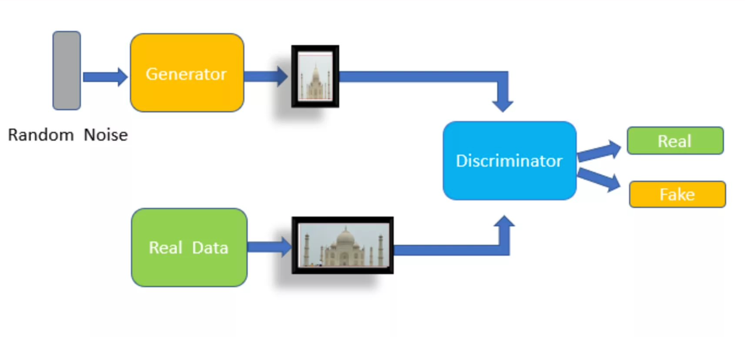


Рисунок 5 – структура GAN сети.

На данный момент в силу отсутствия сервера с поддержкой вычислений на GPU в финальной версии приложения пользователю предлагается при генерации использовать именно обученный генератор.

Как видно из схемы, мы не ограничены выбором архитектуры для генератора и дискриминатора модели, это могут быть сверточные сети, рекуррентные или любая другая архитектура. В ходе экспериментов стало ясно, что рекуррентные сети совсем не подходят для генерации музыки, или для обучения дискриминатора, так как такого вида сети подвержены эффекту «забывания» информации. Сверточные сети показали гораздо лучшие результаты, но скорость генерации и обучения значительно ниже ожидаемого значения. Это обусловлено тем, что свертки в процессе обучения должны пройти по всему временному ряду. Для генерации берутся последние четыре секунды аудио, частота дискретизации равна сорока четырем тысячам ста. Значит, на входе модель получает сто семьдесят шесть тысяч четыреста чисел. Данная сеть показала неплохие результаты при обучении, но не способна работать на сервере без GPU.

В финальной версии проекта в качестве архитектуры генератора и дискриминатора используются многослойные полносвязные сети, которые работают достаточно быстро на CPU, хотя и результаты генерации не отличаются особым качеством.

1. Jukebox[[9]](#footnote-9). Данная архитектура была представлена компанией OpenAI в апреле 2020 года. В основе данной модели лежат три составляющие: вариационный автокодировщик и две модели, построенных на основе трансформера[[10]](#footnote-10).

Иерархический автокодировщик обучается сжимать и разжимать данные в набор векторов. При генерации музыки используется только декодировщик. Кодировщик переводит звуковые фрагменты в дискретное пространство, которые потом можно интерпретировать как ембединги[[11]](#footnote-11), на которые обучается transformer.

Ниже приведена архитектура вариационного автокодировщика сети jukebox (рис. 6)

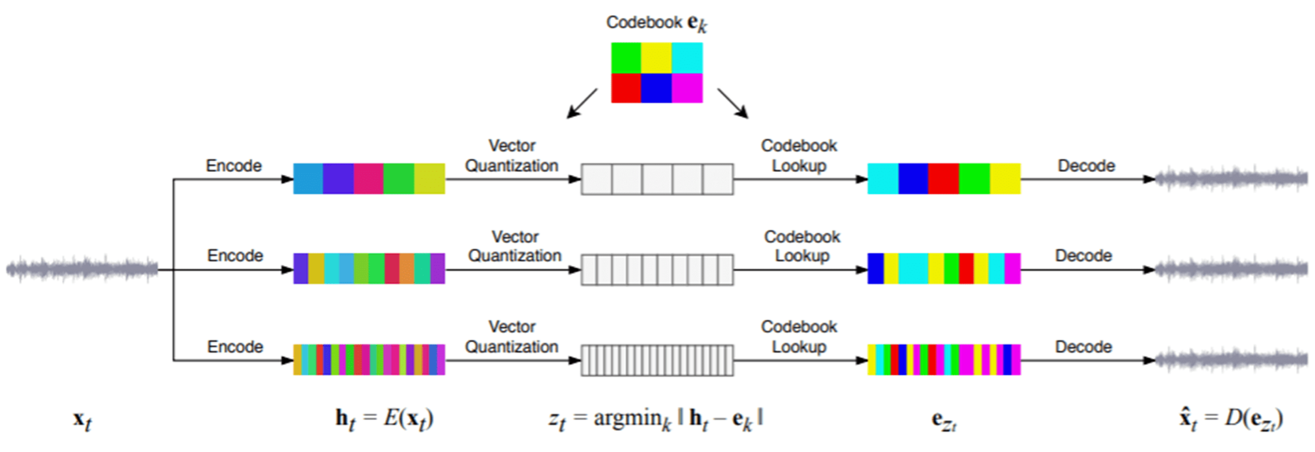


Рисунок 6 – структура вариационного кодировщика сети Jukebox

Ниже приведена архитектура модели сети Jukebox (рис. 7)

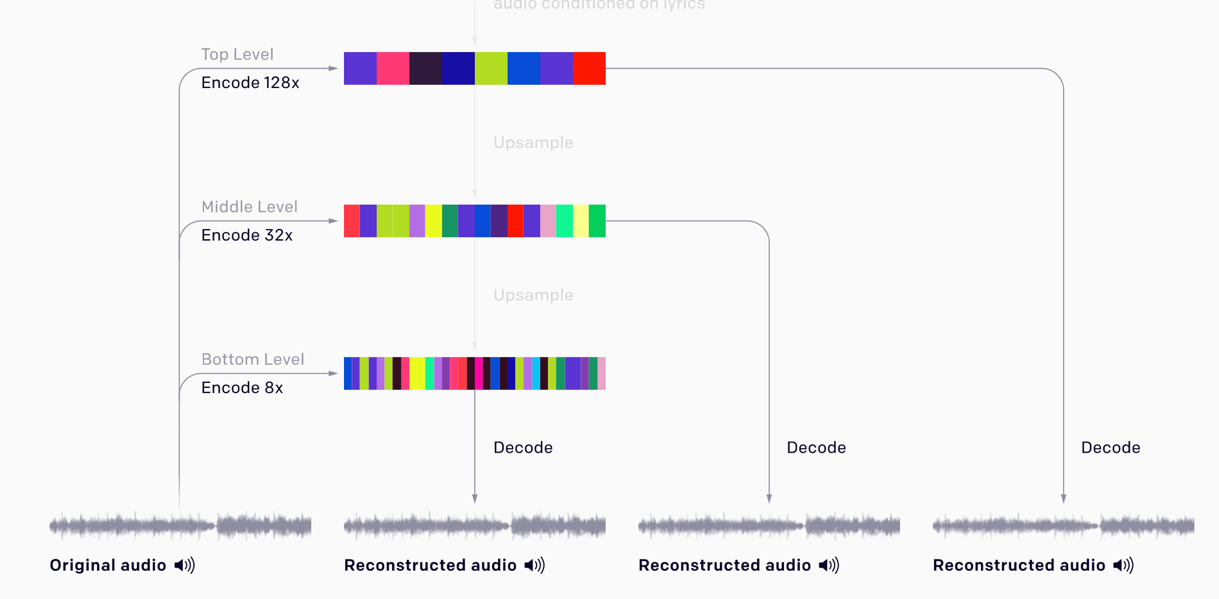


Рисунок 7 – архитектура сети Jukebox

Результаты данной модели гораздо выше остальных. В процессе экспериментов была обучена одна модель Jukebox на треках музыканта Solar Fields. В сумме обучение на двух GPU на нескольких часах аудио заняло три дня. Результаты модели оказались лучшими из всех, что были получены ранее. Для генерации других треков были использованы предобученые Jukebox модели от компании OpenAI. Пока что у пользователя нет возможности воспользоваться данной сетью, так как для генерации требуются большие вычислительные ресурсы.

В финальной версии проекта для генерации музыкальных композиций используется генеративная сеть, так как для нее не требуется значительных ресурсов. Заранее сгенерированные треки, доступные всем пользователям, были сгенерированы сетью Jukebox, так как прослушивание сгенерированных ею композиций показало, что преимущество выбранной модели над всеми альтернативами.

## Архитектура базы данных

База данных состоит из трех таблиц: Users, Tracks, Playlists.

Ниже приведена структура базы данных, используемой в приложении (рис. 8)

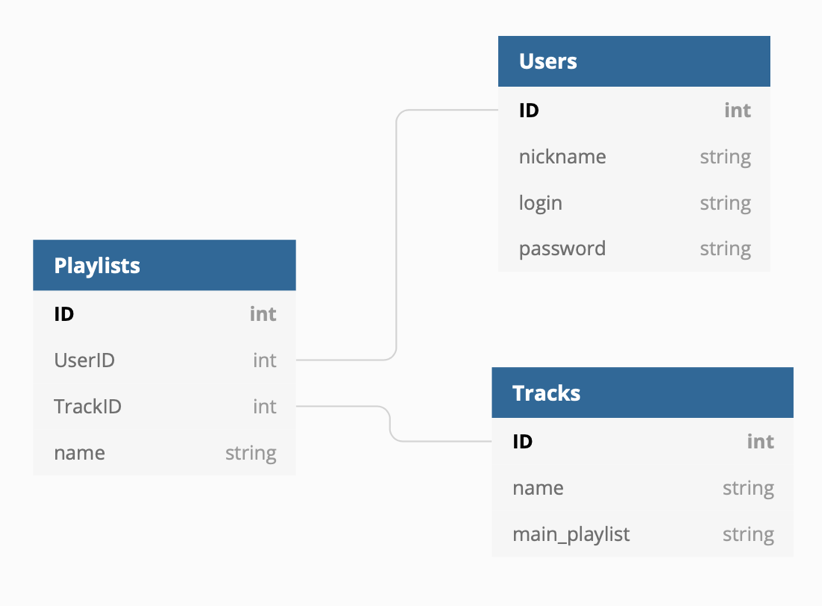


Рисунок 8 – схема базы данных

## Описание алгоритма и функционала программы

Музыка генерируется в авторегрессионном режиме. После генерации нового фрагмента он добавляется в конец файла, далее процесс повторяется.

На вход принимается 4 секунды моноканального аудиосигнала, при частоте дискретизации в сорок четыре тысячи сто единиц. Модель принимает на вход данный интервал, генерирует новые пятьсот двенадцать значений, после чего добавляет их к концу уже существующей последовательности. Далее алгоритм повторяется до тех пор, пока длина полученной последовательности не будет равна десяти секундам.

Ниже приведен алгоритм генерации музыки в авторегрессионном формате (рис. 9)

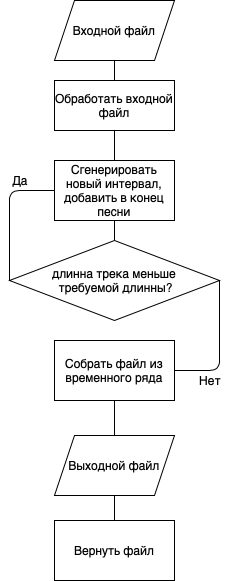


Рисунок 9 – алгоритм генерации музыки в авторегрессионном формате

## Описание и обоснование выбора технических средств

Для надежной и бесперебойной работы сервера требуется стабильное интернет-соединение скоростью не меньше, чем 1Мб/с, персональный компьютер с виртуальным окружением Python со следующими характеристиками: видеокарта nvidia c 12 Gb памяти, процессор: 2.8ГГц, память: 16Гб, свободное место на жестком диске 150Мб, клавиатура, мышь, монитор, разрешением не менее 1920х1080 пикселей. Для корректной работы сервера необходимы библиотеки: numpy версии 1.18.1, sklearn версии 0.22.2, PyTorch версии 1.8.0, Flask Restful версии 1.1.2., librosa версии 1.0.0.

# Ожидаемые технико-экономические показатели

## Предполагаемая потребность

В рамках данной работы оценка предполагаемой потребности не предусмотрена.

## Экономические преимущества разработки по сравнению с отечественными и зарубежными аналогами.

В рамках данной работы оценка экономических показателей не предусмотрена.

# Перспективы развития

«Серверную часть приложения по генерации музыки» можно развивать в следующих направлениях:

1. Исследование новых архитектур позволит генерировать интервалы все более похожие на музыку, написанную человеком. На основе jukebox архитектуры можно изменить процесс восстановления временного интервала, чтобы повысить скорость генерации музыки.
2. Решения по отчистке трека от шума позволило бы снизить планку качества сети JukeBox.
3. Отслеживание прослушивания треков от пользователей позволило бы создать систему рекомендаций, основанную на интересах каждого конкретного пользователя.

# Список источников

1. Wavenet: A Generative Model for Raw Audio, Электронный ресурс: URL: <https://arxiv.org/abs/1609.03499> (Дата обращения: 12.12.2020);
2. Jukebox: A Generative Model for Music, Электронный ресурс: URL: <https://arxiv.org/abs/2005.00341> (Дата обращения: 08.02.2021);
3. Generating music with AI (or transformers go brrr), Электронный ресурс: <https://alxmamaev.medium.com/generating-music-with-ai-or-transformers-go-brrrr-3a3ac5a04126> (Дата обращения: 12.01.2021);
4. Receive or Return files-Flask-API, Электронный ресурс: URL: <https://medium.com/analytics-vidhya/receive-or-return-files-flask-api-8389d42b0684> (Дата обращения: 25.03.2020)
5. Music Generation using Deep Learning, Электронный ресурс: <https://medium.com/@harsh2000.hp/music-generation-using-deep-learning-59159b95fe68> (Дата обращения: 09.11.2020);
6. How to Generate Music using a LSTM Neural Network in Keras, Электронный ресурс: URL: <https://towardsdatascience.com/how-to-generate-music-using-a-lstm-neural-network-in-keras-68786834d4c5> (Дата обращения: 11.09.2020);
7. Generative Adversarial Networks, Электронный ресурс: URL: <https://arxiv.org/abs/1406.2661> (Дата обращения: 25.02.2021);
8. AI That Can Generate Music, Электронный ресурс: URL: <https://medium.com/the-research-nest/ai-that-can-generate-music-fddc5813376a> (Дата обращения: 01.04.2021)
9. MuseNet, Электронный ресурс: URL: <https://openai.com/blog/musenet/> (Дата обращения: 10.03.2021).

# ПРИЛОЖЕНИЕ 1

**Описание используемых классов**

|  |  |
| --- | --- |
| **Класс** | **Описание класса** |
| Generator | Модель – генератор, часть генеративной модели для генерации музыки. |
| Discriminator | Модель – дискриминатор, часть генеративной модели для генерации музыки. |
| SongDataset | Класс – обертка над данными, взятыми для обучения из одной конкретной песни. |
| GanSongTrainer | Класс для запуска обучения и мониторинга состояния GAN сети. |
| MusicDataset | Класс – обертка над данными, взятыми из файла с песнями одного исполнителя. |
| Trainer | Класс для запуска обучения и мониторинга WaveNet сети. |
| Wave\_Block | Часть WaveNet сети для операций со свертками. |
| Classifier | Класс – классификатор для генерации музыки на основе сети WaveNet. |

**Описание класса Generator**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Назначение** |
| \_\_init\_\_ | Метод | Конструктор класса |
| forward | Метод | Метод для прямого прохода через нейронную сеть |
| style\_linear | Поле | Линейный слой над элементом песни, предшествующем сгенерированному интервалу |
| batch\_norm\_style | Поле | Слой батч-нормализации для предотвращения переобучения |
| noise\_layer | Поле | Линейный слой для обработки случайного шума |
| batch\_norm\_noise | Поле | Слой батч-нормализации для предотвращения переобучения |
| final | Поле | Линейный слой для финальной обработки данных перед возвратом значений из модели |
| relu | Поле | Функция активации ReLU |
| tanh | Поле | Функция активации Tanh |

**Описание класса Discriminator**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Назначение** |
| \_\_init\_\_ | Метод | Конструктор класса |
| forward | Метод | Метод для прямого прохода через нейронную сеть |
| style\_linear | Поле | Линейный слой над элементом песни, предшествующем сгенерированному интервалу |
| final | Поле | Линейный слой для финальной обработки данных перед возвратом значений из модели |
| noise\_layer | Поле | Линейный слой для обработки вывода генератора |
| style\_batch\_norm | Поле | Слой батч-нормализации для предотвращения переобучения |
| candidate\_batch\_norm | Поле | Слой батч-нормализации для предотвращения переобучения |
| relu | Поле | Функция активации ReLU |
| sigm | Поле | Функция активации «сигмоида» |

**Описание класса SongDataset**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Назначение** |
| \_\_init\_\_ | Метод | Конструктор класса |
| \_\_getitem\_\_ | Метод | Позволяет использовать объект класса как итератор |
| \_\_len\_\_ | Метод | Возвращает кол-во элементов датасета |
| data | Поле | Переменная для хранения данных для подачи в модель |
| sr | Поле | Частота дискретизации |
| input\_size | Поле | Размер окна отрывка от песни |
| output\_size | Поле | Размер сгенерированного окна |

**Описание класса GanSongTrainer**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Назначение** |
| \_\_init\_\_ | Метод | Конструктор класса |
| train\_one\_epoch | Метод | Запускает одну эпоху обучения модели |
| train | Метод | Запускает процесс обучения модели |
| generator\_loss | Метод | Функция потерь для обучения генератора |
| discriminator\_loss | Метод | Функция потерь для обучения дискриминатора |
| generator | Поле | Объект модели-генератора |
| discriminator | Поле | Объект модели-дискриминатора |
| dataset | Поле | Переменная для хранения данных |
| dataloader | Поле | Переменная для подачи данных в модель |
| gen\_optim | Поле | Оптимизатор для модели-генератора |
| disc\_optim | Поле | Оптимизатор для модели-дискриминатора |
| steps\_to\_print | Поле | Кол-во шагов для вывода информации об обучении |
| steps\_to\_save | Поле | Кол-во шагов для сохранения весов модели |
| steps\_to\_stop | Поле | Кол-во шагов для остановки обучения модели на одной эпохе |
| device | Поле | Переменная, отвечающая за обучение модели на GPU или CPU |
| n\_critic | Поле | Соотношение шагов для обучения дискриминатора для одного шага генератора |
| input\_size | Поле | Размер окна отрывка от песни |
| noise\_size | Поле | Размер вектора случайных значений |
| output\_size | Поле | Размер сгенерированного интервала |

**Описание класса MusicDataset**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Назначение** |
| \_\_init\_\_ | Метод | Конструктор класса |
| \_\_getitem\_\_ | Метод | Позволяет использовать объект класса как итератор |
| \_\_len\_\_ | Метод | Возвращает кол-во элементов датасета |
| root | Поле | Корень файловой системы, откуда начинается поиск музыкальных файлов |
| slice\_size | Поле | Размер окна песни для предсказания следующего символа |
| files | Поле | Список путей к файлам песен, на которых проходит обучение |

**Описание класса Trainer**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Назначение** |
| \_\_init\_\_ | Метод | Конструктор класса |
| train\_one\_epoch | Метод | Запускает одну эпоху обучения модели |
| train | Метод | Запускает обучение модели |
| model | Поле | Объект модели, которую обучает класс Trainer |
| device | Поле | Переменная, отвечающая за обучение модели на GPU или CPU |
| optimizer | Поле | Класс, реализующий метод оптимизации |
| dataset | Поле | Объект – обертка над данными для подачи данных на обучение |
| loss | Поле | Класс-обертка над функцией потерь обучения модели |
| batch\_size | Поле | Кол-во объектов на одном проходе модели |
| steps\_to\_print | Поле | Кол-во шагов для вывода информации об обучении |
| steps\_to\_save | Поле | Кол-во шагов для сохранения весов модели |
| steps\_to\_stop | Поле | Кол-во шагов для остановки обучения модели на одной эпохе |

**Описание класса Wave\_Block**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Назначение** |
| num\_rates | Поле | Кол-во слоев модели WaveNet |
| convs | Поле | Список классических сверток для обработки входных данных |
| filter\_convs | Поле | Список фильтрующих сверток для предотвращения обучения |
| gate\_convs | Поле | Список «арочных» сверток для поиска паттернов в исходных данных |
| \_\_init\_\_ | Метод | Конструктор класса |
| forward | Метод | Метод для прямого прохода через нейронную сеть |

**Описание класса Classifier**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Назначение** |
| \_\_init\_\_ | Метод | Конструктор класса |
| forward | Метод | Метод для прямого прохода через нейронную сеть |
| wave\_block1 | Поле | Сверточный блок сети архитектуры WaveNet |
| wave\_block2 | Поле | Сверточный блок сети архитектуры WaveNet |
| wave\_block3 | Поле | Сверточный блок сети архитектуры WaveNet |
| wave\_block4 | Поле | Сверточный блок сети архитектуры WaveNet |
| fc | Поле | Полносвязный линейный слой |
| relu | Поле | Функция активации ReLU |
| flatten | Поле | Объект класса Flatten для склейки векторов одного потока данных в один вектор |
| lin\_final | Поле | Финальный полносвязный линейный слой |

**Описание серверных составляющих**

|  |  |  |
| --- | --- | --- |
| **Название** | **Тип** | **Назначение** |
| app | Глобальная переменная | Переменная для запуска API |
| api | Глобальная переменная | Переменная для поддержки API на сервере |
| generate\_music | Функция | Функция для обработки запроса для генерации музыки |
| post\_playlist | Функция | Функция для обработки запроса для записи плейлиста в базу данных |
| post\_track | Функция | Функция для обработки запроса для записи трека в базу данных |
| get\_user\_one\_playlist | Функция | Функция для обработки запроса для поиска конкретного плейлиста конкретного пользователя |
| get\_user\_playlists | Функция | Функция для обработки запроса для вывода всех плейлистов пользователя |
| get\_all\_networks | Функция | Функция для обработки запроса для возврата информации о всех существующих генеративных сетях |
| get\_track | Функция | Функция для обработки запроса для добавления трека в базу данных |
| get\_tracks | Функция | Функция для обработки запроса для вывода всех треков из базы данных |
| get\_user | Функция | Функция для обработки запроса для возврата информации о пользователе |
| get\_user\_by\_id | Функция | Функция для обработки запроса для возврата информации о пользователе по его id |
| get\_user\_by\_nickname | Функция | Функция для обработки запроса для возврата информации о пользователе по его никнейму |
| register\_user | Функция | Функция для обработки запроса на регистрацию нового пользователя |
| get\_all\_users | Функция | Функция для обработки запроса для возврата информации о всех существующих пользователях |

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в документе | № документа | Входящий № сопроводительного документа и дата | Подпись | Дата |
| измененных | замененных | новых | аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |

1. https://ru.wikipedia.org/wiki/MIDI [↑](#footnote-ref-1)
2. https://openai.com/blog/musenet/ [↑](#footnote-ref-2)
3. https://en.wikipedia.org/wiki/GPT-2 [↑](#footnote-ref-3)
4. https://openai.com/blog/sparse-transformer/ [↑](#footnote-ref-4)
5. https://arxiv.org/abs/1609.03499 [↑](#footnote-ref-5)
6. https://neerc.ifmo.ru/wiki/index.php?title=Сверточные\_нейронные\_сети [↑](#footnote-ref-6)
7. https://habr.com/ru/company/Voximplant/blog/309648/ [↑](#footnote-ref-7)
8. https://ru.wikipedia.org/wiki/Генеративно-состязательная\_сеть [↑](#footnote-ref-8)
9. https://arxiv.org/abs/2005.00341 [↑](#footnote-ref-9)
10. https://arxiv.org/abs/1706.03762 [↑](#footnote-ref-10)
11. https://www.nkj.ru/open/36052/ [↑](#footnote-ref-11)