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
Руководитель проекта
ООО «Максимал»

_____ Н.В. Бартая

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Предобработка аудио с извлечением метаданных для обучения нейросетевой модели по рекомендации музыки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.04.04.2024.308-1486.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.,
доцент
_____ И.И. Клебанов

Автор работы,
студент группы КЭ-229
_____ И.А. Березин

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
Березину Ивану Александрович,
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)

Предобработка аудио с извлечением метаданных для обучения нейросетевой модели по рекомендации музыки.

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

3. **Исходные данные к работе**

3.1. Необработанный набор аудио.

3.2. Модель для рекомендации музыки AudioCraft от Facebook. [Электронный ресурс]. URL: https://audiocraft.metademolab.com/ (дата обращения: 18.02.2024 г.).

3.3. API documentation для AudioCraft от Facebook. [Электронный ресурс]. URL: https://facebookresearch.github.io/audiocraft/api_docs/audiocraft/index.html (дата обращения: 18.02.2024 г.).

4. **Перечень подлежащих разработки вопросов**

4.1. Актуальность выбранной темы исследования.

4.2. Методы обработки и взаимодействия с аудио.

4.3. Нейросетевые модели и наборы данных по работе с аудио.

4.4. Обучение модели на обработанных данных.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н., доцент И.И. Клебанов

Задание принял к исполнению И.А. Березин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В эпоху развития технологий, связанных с искусственным интеллектом (ИИ), машинным обучением и нейронными сетями, очень важно не только соответствовать современным тенденциям в использовании, но также необходимо разбираться в реализации таких систем, их обучении и методах контроля.

В данной работе будет рассмотрен важнейший этап в создании любой аналогичной модели – этап обучения. Именно данные, подающиеся в модель на этом этапе, определяют суть модели. От полноты данных, от их структурированности и чистоты зависит не только то, как модель будет вести себя на этапе обучения, но и то, какой она будет выдавать результат.

Для достижения оптимальных результатов запланированного исследования или для будущего интегрирования модели в какую-либо систему, необходимо чтобы данные подаваемые на момент обучения в модель, были идеальными и соответствовали всем требованиям и ожиданиям. Именно это делает данную область актуальной в текущих реалиях, ведь данные никогда не бывают идеальными и без необходимой обработки, дальнейшая работа просто невозможна.

Постановка задачи

Целью выпускной квалификационной работы является предобработка аудио с извлечением метаданных для обучения нейросетевой модели по рекомендации музыки, благодаря этой системе можно будет запускать генератор наборов данных, для обучения модели на музыке по различной тематике.

Для достижения поставленной цели необходимо решить следующие задачи:

1) проанализировать предметную область;
2) определить структуру и необходимые преобразования данных;
3) наладить автоматизацию с базой данных;

4) реализовать генератор обучающего набора данных;

5) провести тестирование и обучить модель.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения, списка литературы и приложения.

Объем работы составляет 78 страниц, объем списка литературы – 26 источников.

В первой главе «Обзор литературы» содержится краткая сводка по источникам литературы, которые используются в качестве идеи реализации и теоретической основы работы.

Во второй главе «Анализ предметной области» описывается анализ исходных данных, а также инструменты и технологии определенные для разработки системы.

В третьей главе «Проектирование» представлена система, определены функциональные и нефункциональные требования, а также описана структура и содержание основных и дополнительных компонентов.

В четвертой главе «Реализация» описывается реализация разрабатываемой системы в соответствии с определенными требованиями и поставленными задачами.

В пятой главе «Тестирование» приведены результаты функционального тестирования системы и тестирования дополнительных компонентов, используемых для расширения вариативности применения.

В приложении А содержатся скриншоты расширения диаграммы компонентов для разрабатываемых компонентов системы.

В приложении Б содержатся скриншоты кода, который не вошел в основную часть работы.

В приложении В содержится листинг полной архитектуры нейросетевого кодека «EnCodec» от «facebookresearch».

1. ОБЗОР ЛИТЕРАТУРЫ

Обзор литературы является важнейшим этапом в любой исследовательской или проектной деятельности. Этот этап является определяющим в выборе методологии, проведения анализа и подчеркивания актуальности исследуемой области.

Важные аспекты обзора литературы, открывающиеся исследователю.

Определение текущего состояния знаний

Анализ существующих теорий, методов и результатов исследований позволяет выявить текущие тенденции и открытые проблемы в области предобработки аудио и создания наборов данных в целом.

Выбор подходящих методов и инструментов

Изучение литературы позволяет оценить эффективность различных методов предобработки и создания «датасетов», что помогает исследователю принять обоснованные решения при выборе инструментов и методологии.

Определение инноваций и тенденций

Следя за последними исследованиями, можно выявить инновации, которые могут внести вклад в развитие области, а также подхватить тенденции, которые будут важными для будущих исследовательских работ.

Обобщая сведения из различных источников, обзор литературы становится стратегическим инструментом, формирующим теоретическую основу для дальнейших исследований в области и написания работы по теме: «Обработка аудио с извлечением метаданных для обучения нейросетевой модели по рекомендации музыки».

Для дальнейшей работы были изучены следующие работы.

В работе «Speech and Audio Signal Processing: Processing and Perception of Speech and Music» by Ben Gold and Nelson Morgan [1] предоставляется введение в основные концепции обработки аудиосигналов, охватывая цифровую обработку сигналов, преобразования Фурье и основы восприя-

тия речи и музыки. Она является отличным ресурсом для понимания технических аспектов предобработки аудиоданных.

В книге «Python Machine Learning» by Sebastian Raschka and Vahid Mirjalili [2] содержится обширный обзор методов машинного обучения с использованием Python, включая главы о работе с данными. Особое внимание уделяется библиотекам, таким как «scikit-learn» и «librosa», что делает ее полезной для освоения техник предобработки и анализа аудиоданных.

В научном труде «Deep Learning for Audio, Image and Video Analysis» by Rajalingappaa Shanmugamani [3] представляется глубокое обучение в контексте аудиоанализа, затрагивая темы, такие как сверточные и рекуррентные нейронные сети. Она подробно рассматривает применение этих методов к обработке аудиосигналов, включая спектрограммы и временные ряды.

В данной статье «Urban Sound Classification, Using Convolutional Neural Networks» by Justin Salamon and Juan Pablo Bello [4] о фоновых звуках городской среды предлагаются конкретные примеры использования сверточных нейронных сетей для классификации аудиосигналов. Она также обсуждает вопросы сбора и предобработки данных для достижения высокой точности классификации.

Статья «Freesound Datasets: A Platform for the Creation of Open Audio Datasets» by Frederic Font, Gerard Roma, and Xavier Serra [5] представляет платформу Freesound и дает обзор создания открытых аудиодатасетов. Она включает в себя рекомендации по аннотации данных и методам, способствующим созданию качественных датасетов для обучения моделей аудиообработки.

Книга «Audio Signal Processing for Music Applications» by Joshua D. Reiss and Andrew McPherson [6] фокусируется на аудиосигналах в музыкальных приложениях. Она охватывает темы, такие как анализ тональности, извлечение ритма и многие другие, предоставляя практические при-

меры и алгоритмы, полезные для аудио-предобработки в музыкальном контексте.

В данном справочнике «Handbook of Signal Processing in Acoustics» edited by DavidHavelock [7] предлагается обширный обзор обработки сигналов в акустике. В частности, разделы о звуке и музыке содержат полезные материалы по анализу и обработке аудиоданных, а также техникам создания акустических датасетов.

Отдельно необходимо отменить, что в работе используется нейросетевая модель от компании Facebook «AudioCraft», для работы с ней необходимо ознакомиться с API документацией [8] предоставленной в репозитории на GitHub. Этот документ является основным в работе, так как правильное взаимодействие с моделью без ознакомления с документацией попросту невозможно.

Вывод по первой главе

В первой главе были рассмотрены основные источники литературы, которые используются для анализа и разработки искомой системы. Эти источники формируют комплексный обзор современных методов и техник, применяемых в предобработке аудиоданных и создании наборов данных. Их изучение поможет понять принцип работы с данными, их обработку и поможет в реализации собственного проекта.

2. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

2.1. Обработка исходных данных

Так как ключевой особенностью разрабатываемой системы является обработка аудиоданных, то подготовительным этапом при разработке генератора «датасетов» необходимо проанализировать исходные данные к работе, которые включают: необработанный набор аудио, модель и документацию по работе с моделью. В документации содержится необходимая информация по взаимодействию с предобученными моделями, спецификация использования различных модулей, в том числе и для обучения модели на своих данных.

2.1.1. Необработанный набор данных

Основным элементом в данной работе, являются поступающие на вход к генератору аудиофайлы. Основной объем данных для работы содержится на удаленном сервере и представляет собой хранилище аудиодорожек разного качества, длины, формата, от различных исполнителей и различного тематического направления. Единственным объединяющим фактором служит то, что все аудио являются различной музыкой для медитации. На основе этих данных, можно провести обучение соответствующих моделей, как для рекомендации, так и для генерации музыки. Образец исходных аудиофайлов, был предварительно загружен на локальную машину и представлен на рисунках 1 и 2.

На рисунке 1 представлена музыка для медитации, которая классифицирована по различным направлениям, тематикам или особенностями звучания. Общее количество различных направлений более 100 штук, поэтому на рисунке представлена лишь малая часть в качестве примера.

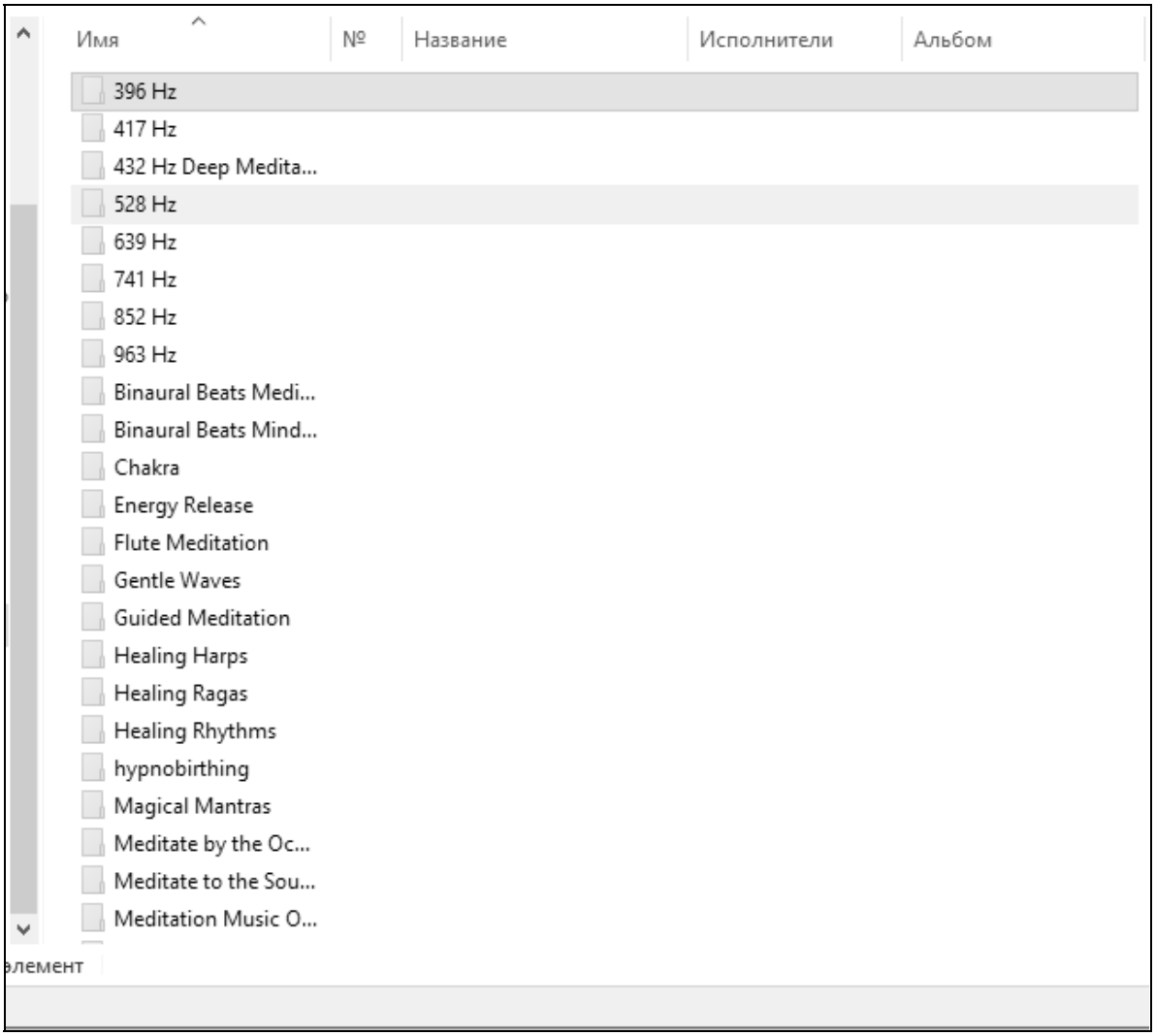


Рисунок 1 – Классификация музыки для медитации

На рисунке 2 представлен образец аудиодорожек от различных исполнителей из разных стран, классифицированных по частоте звучания в 963 Hz. Также можно наглядно убедиться, что некоторые аудио даже на первый взгляд уже не обладают всеми показателями валидности, например, название аудиофайла «Vi%U0301%U00F..» в списке на рисунке 2 не является информативным, что указывает на необходимость дополнительной проверки во время извлечения метаданных.

Имя	№	Название	Исполнители	Альбом
01 8 Chakras Bowls....	1	8 Chakras Bowls	Mudra Namaste	Salutation Bowls
01 963 Hz El Futuro....	1	963 Hz El Futuro	Paz Interna	963 Hz El Futuro
01 963 Solfeggio Att...	1	963 Solfeggio Attunement	Healing Solfeggio ...	963 Path to Awakening
01 963 Solfeggio.mp3	1	963 Solfeggio	Melatonement	963 Solfeggio
01 963hz Connect t...	1	963hz Connect to Source ...	Solfeggio Frequen...	963hz Connect to So...
01 Alpha bowls 10h...	1	Alpha bowls 10hz	Dream Wlzard	Alpha bowls 10hz
01 Awakenings 963 ...	1	Awakenings 963 Hz	Hertz-Hunters	Awakenings 936 & 43...
01 Chakra Healing....	1	Chakra Healing	Makalu	Chakra Healing
01 Clear Thoughts 9...	1	Clear Thoughts 963 Hz	Ozonezzz	Clear Thoughts 963 Hz
01 Crown Chakra - ...	1	Crown Chakra - Asian Flu...	Massage Tribe	Chakra Healing - Ma...
01 Divine Hemisphe...	1	Divine Hemispheres 963 Hz	Zephyrical	Divine Hemispheres ...
01 Golden Intuition ...	1	Golden Intuition 963Hz	Introspective Rele...	Golden Intuition 963Hz
01 Letting Go.mp3	1	Letting Go	Frozen Voices	Letting Go
01 Miracles 963 Hz....	1	Miracles 963 Hz	Drone-Dax	Miracles 963 Hz
01 Nagnallar.mp3	1	Nagnallar	Sahasrara Beats	Nagnallar
01 Orange Waves 9...	1	Orange Waves 963 hz	Afar Lux	Orange waves
01 Point Cabrillo.mp3	1	Point Cabrillo	Lundegard	Point Cabrillo
01 Solar Waves.mp3	1	Solar Waves	Tejal Yann	Solar Waves
01 The Sun Rises 96...	1	The Sun Rises 963 Hz	Tina Cirdan	The Sun Rises 963 Hz
01 Unrushed.mp3	1	Unrushed	Solfeggio Dreams	69,3 Hz Atmospheres...
01 Vast and Blue.mp3	1	Vast and Blue	Gloaming Days	Vast and Blue
01 Vi%U0301%U00F...	1	Víðsýni	Traum Fanger	Víðsýni
02 Chakra Healing ...	2	Chakra Healing Bowl (Sou...	EXOPIRS	Meditation Sounds

Рисунок 2 – Образец музыки для медитации из раздела 963 Hz

2.1.2. «AudioCraft» от Facebook

«AudioCraft» [8] – это программное обеспечение (ПО), созданное для генерации высококачественного аудио и музыки на основе текста. Этот инструмент представляет собой единый кодовый базис для различных потребностей в генерации аудио: музыки, звуковых эффектов и сжатия после обучения на сырых аудиоданных. В нем реализованы модели «MusicGen», «AudioGen» и «EnCodec».

Так как модели «MusicGen», «AudioGen» используются для генерации аудио, то рассматривать их подробно не будем. На рисунке 3 представлена схематичная архитектура обоих моделей.

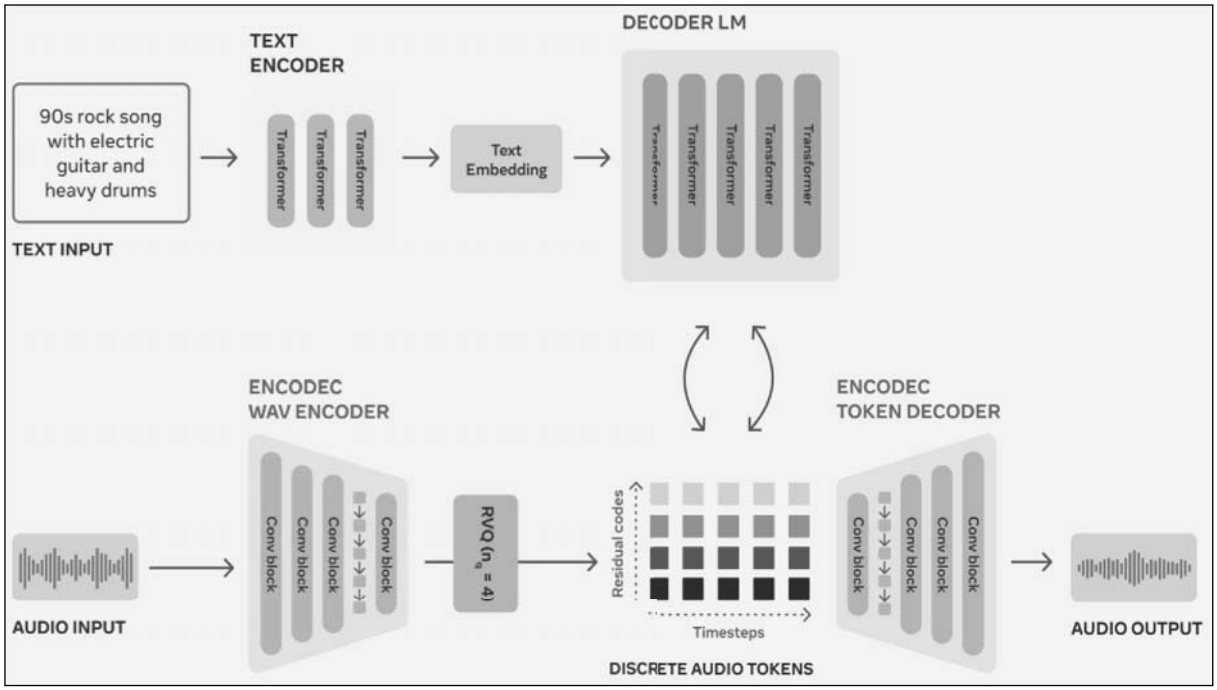


Рисунок 3 – Работа алгоритма «AudioGen» и «MusicGen»

«MusicGen»

«MusicGen» представляет собой значительное достижение в области условной генерации музыки. Алгоритм построен на одноступенчатой модели Language Model (LM), которая работает с сжатыми дискретными музыкальными токенами. Этот инновационный подход, вместе с эффективными шаблонами чередования токенов, оптимизирует процесс генерации, устраняя необходимость каскадирования нескольких моделей.

Одним из ключевых преимуществ «MusicGen» является его способность генерировать высококачественные образцы музыки, как моно, так и стерео, при условии подробных текстовых описаний или наличии мелодических особенностей.

«AudioGen»

«AudioGen» – авторегрессивная генеративная модель, которая создает аудиообразцы на основе текстовых входных данных. AudioGen оперирует на изученном дискретном аудиопредставлении.

Задача генерации текста в аудио включает в себя несколько сложностей:

– распознавание «объектов» (может быть сложной задачей из-за разделения нескольких одновременно говорящих людей);

– усложненные реальные условия записи (например, наличие фонового шума, эхо и прочее);

– недостаток текстовых аннотаций, который ограничивает возможность масштабирования моделей;

– моделирование аудио высокой точности, что требует кодирования аудио с высокой частотой дискретизации. Это приводит к крайне длинным последовательностям.

Чтобы смягчить вышеупомянутые трудности, используется техника аугментации, которая смешивает различные аудиообразцы, заставляя модель внутренне научиться разделять несколько источников.

«EnCodec»

«EnCodec» – современный аудио кодек, обеспечивающий высокую точность и работающий в реальном времени за счет использования нейронных сетей.

Этот кодек представляет собой стриминговую архитектуру энкодера-декодера с квантованным скрытым пространством, обученную в едином конвейерном процессе. Процесс обучения был улучшен и ускорен с помощью мультишкального адверсария спектрограммы, который эффективно уменьшает артефакты и генерирует высококачественные аудиообразцы. Механизм балансировки потерь, который способствует стабилизации обучения путем разрыва связи выбора гиперпараметров с типичной величиной потерь. Применение легких моделей «transformer» (трансформер) для дополнительного сжатия полученного представления до 40%, оставаясь при этом быстродействующими.

На рисунке 4 представлена архитектура «EnCodec». «Encoder» (энкодер) получает входные аудиоданные и преобразует их в компактное скрытое представление с помощью сверточных и рекуррентных нейронных сетей. Далее «Quantizer» (квантователь) преобразует непрерывные значения скрытого представления аудиоданных в дискретные, квантованные значения. Использование квантователя не обязательное, модель будет работать и без него, однако, модели трансформера, для дополнительного сжатия скрытого представления, позволяют улучшать производительность кодека при передаче данных. Полученное скрытое представление передается в «Decoder» (декодер), который восстанавливает исходный аудиосигнал.

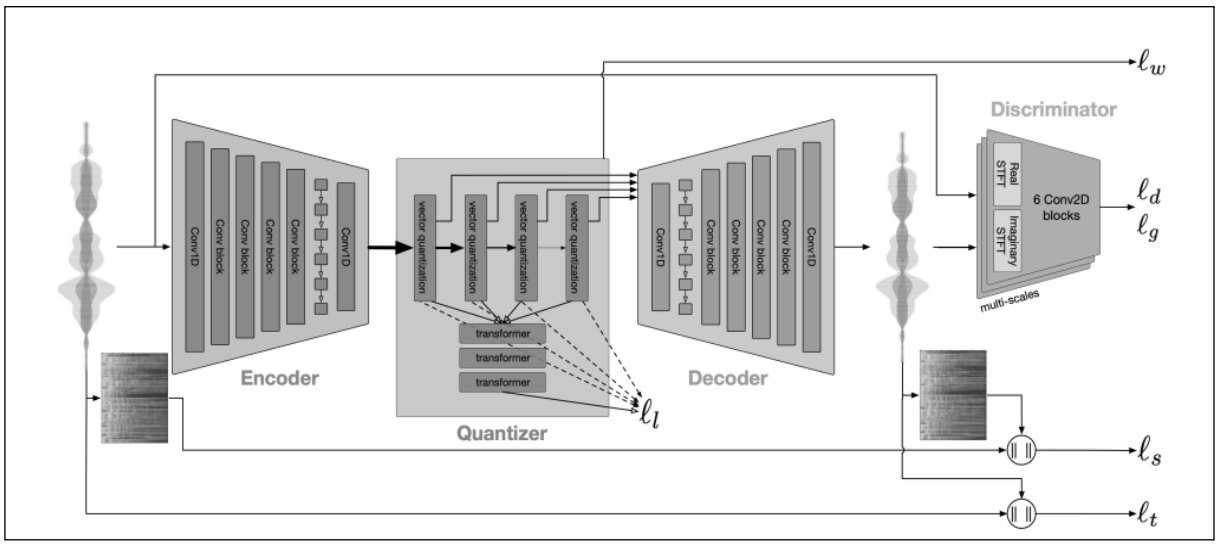


Рисунок 4 – Архитектура «EnCodec»

«Discriminator» (дискриминатор) может оценивать качество восстановленных аудиоданных после их сжатия и восстановления «EnCodec». Путем сравнения оригинальных аудиосигналов с их восстановленными версиями дискриминатор может определять, насколько точно кодек восстанавливает аудио. Это позволяет улучшать процесс сжатия и восстановления для достижения более высокого качества воспроизведения.

2.1.3. API documentation for «AudioCraft»

API documentation [9] включает в себя описание всех доступных функций, классов и методов, предоставляемых библиотекой «AudioCraft». Она предоставляет подробное описание синтаксиса, параметров и возвращаемых значений каждого элемента API, а также примеры использования.

Документация позволяет пользователям более эффективно использовать библиотеку, понимать ее функциональность и интегрировать ее в свои проекты. API documentation также может включать описание конфигурационных файлов, настройку среды и другие детали, необходимые для работы с библиотекой.

Для достижения поставленной задачи в API documentation AudioCraft необходимо было найти раздел по обучению модели «EnCodec», который содержит важную информацию о методах загрузки и использования модели, а также инструкции по обучению и инференции.

В результате анализа было установлено, что для реализации системы необходимо ознакомиться с архитектурой выбранной модели «EnCodec» библиотеки «AudioCraft», ее документацией и преобразовать исходные данные под формат, удовлетворяющий этой модели. Решение задачи автоматизации процесса преобразования данных, которые будут подаваться на обучение через генератор набора данных и является ключевым аспектом данной работы.

2.2. Используемые инструменты и технологии

В качестве основной среды разработки использован PyCharm [10]. PyCharm – это интегрированная среда разработки (IDE) для языка программирования Python [11], разработанная компанией JetBrains. Она предоставляет широкий набор инструментов и функций, облегчающих процесс разработки программного обеспечения. Все основные компоненты системы соответственно реализованы с помощью языка программирования Python.

Основные особенности PyCharm необходимые для разработки, представлены ниже.

1. Редактор кода. Предоставляет удобный и мощный редактор кода с подсветкой синтаксиса, автодополнением, быстрым переходом к определению функций и переменных, поддержкой рефакторинга и многими другими функциями.

2. Интеграция с системами контроля версий. PyCharm интегрирован с системами контроля версий, такими как Git, что облегчает работу с проектами, управление изменениями и совместную работу.

3. Отладка. Позволяет проводить отладку кода Python, устанавливать точки останова, просматривать значения переменных и следить за выполнением программы.

4. Интеграция с другими инструментами и сервисами. PyCharm интегрирован с различными инструментами и сервисами разработки, такими как Docker, SQL, Flask, SQLAlchemy, некоторые из которых будут использованы в работе.

Для решения задач, связанных с операциями с файлами, хранящимися удаленно, были использованы технологии SSH, чтобы безопасно устанавливать соединение между удаленными компьютерами, и распределенная база данных, которая относится к классу NoSQL – MongoDB [12], где данные хранятся в виде документов, напоминающих формат JSON. Каждый документ представляет собой набор пар ключ-значение, где значения могут быть простыми типами данных, массивами или вложенными документами.

Основные особенности MongoDB, представлены ниже.

1. Гибкая схема данных. MongoDB не требует строгой предварительной схемы данных, что означает, что каждый документ в коллекции может иметь различную структуру. Это облегчает разработку и обновление приложений без необходимости изменения схемы базы данных.

2. Масштабируемость. MongoDB поддерживает горизонтальное масштабирование, что позволяет добавлять новые узлы в кластер для увеличения производительности и емкости базы данных.

3. Высокая доступность. MongoDB обеспечивает высокую доступность данных путем репликации данных на несколько узлов кластера. Это позволяет обеспечить непрерывную работу приложений даже в случае отказа одного или нескольких узлов.

4. Мощный язык запросов. MongoDB предоставляет богатый набор операторов запросов для извлечения и манипулирования данными. Он поддерживает индексы для оптимизации запросов и агрегационные операции для выполнения сложных аналитических запросов.

5. Поддержка адаптивного сжатия данных. MongoDB позволяет сжимать данные на лету для экономии места на диске и увеличения скорости передачи данных.

Вывод по второй главе

Во второй главе были проанализированы исходные данные к работе, которые включают в себя исходный набор данных, библиотеку «AudioCraft» от Facebook, которая содержит необходимые модели, а также соответствующую API документацию для работы с моделью «EnCodec». Также были определены инструменты и технологии, которые будут использованы в ходе реализации искомой системы.

3. ПРОЕКТИРОВАНИЕ

При разработке системы, особо важную часть занимает этап проектирования системы. Он включает в себя несколько основных подэтапов, которые могут варьироваться в зависимости от предназначения системы.

3.1. Представление системы

Начальный подэтап проектирования системы – это представление того, как готовая система должна выглядеть, что включать и какой структурой обладать. Для систем, обладающих графическим интерфейсом, этот раздел может включать разработку макета, однако для системы, разрабатываемой в контексте данной работы, наглядным представлением будет служить составленная диаграмма компонентов. Диаграмма компонентов – статическая структурная диаграмма, которая является элементом языка моделирования UML [13]. Диаграмма отображает разбиение системы на компоненты, а также установленную между ними зависимость. С помощью данной диаграммы, можно получить общее представление системы, а также понять какие компоненты будут использованы при разработке, и какие компоненты необходимо разработать для обеспечения функциональности. Для создания диаграмм использовался бесплатный онлайн редактор PlantUML [14].

На рисунке 5 изображена основная диаграмма компонентов, включающая идею реализации системы.

Разрабатываемые компоненты основной диаграммы представлены в приложении А.

На диаграммах отображено, что для реализации системы необходимо разработать несколько основных и несколько вспомогательных компонентов, каждый из которых обладает определенным набором методов.

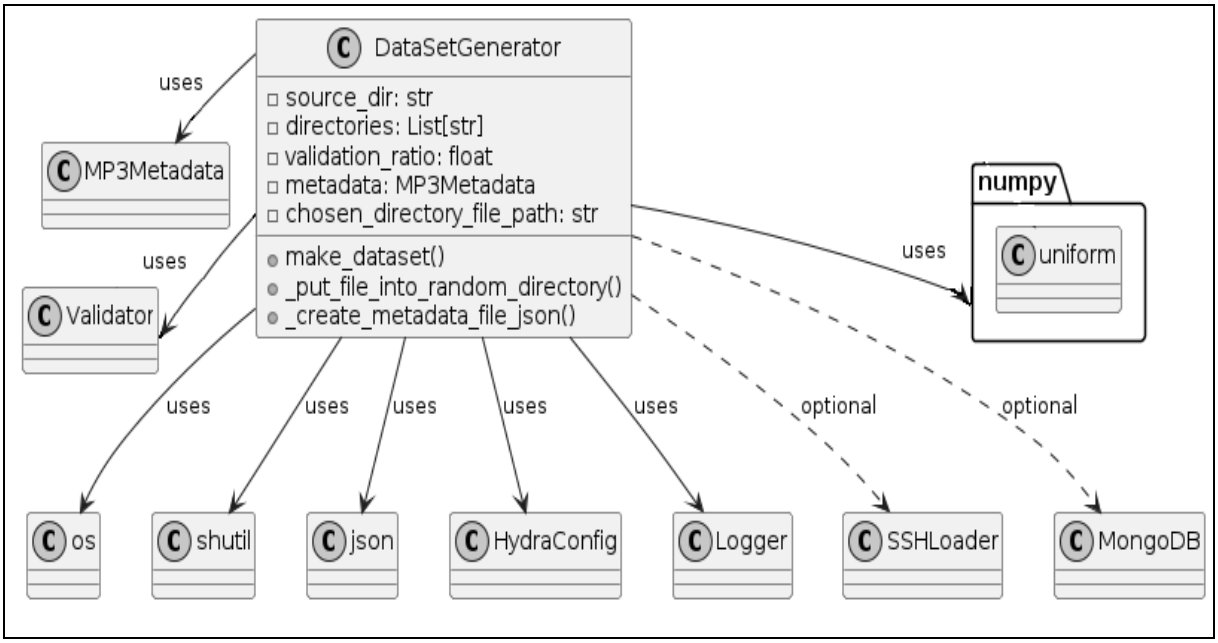


Рисунок 5 – Диаграмма компонентов

Также для работы системы необходимо использовать компоненты для конфигурирования и логирования системы и компоненты для обработки аудиофайлов, и работу с файлами в операционной системе. Вспомогательные компоненты, такие как MongoDB и SShLoader, необходимы в работе для увеличения вариативности функциональности.

Ниже представлено описание для основных компонентов системы.

1. «DataSetGenerator» – основной компонент в работе, который объединяет работу других компонентов. Он принимает на вход директории, которые заданы посредством конфигурационного компонента, коэффициент разбиения, который используется для нормального распределения файлов по выборкам, а также предоставляет постоянный публичный доступ к извлекаемой метадате и выбранной директории после «валидации». Основная задача данного компонента создать результирующий набор данных, в процессе чего файлы проходят проверку, извлекается их метадата, после чего они разделяются на подвыборки.

2. «BaseMetadata» – компонент, представляющий собой базовый «датакласс», на основе которого переопределяются наследуемые классы

под каждый тип аудиофайла. «Датакласс» определяет поля данных, используемых при заполнении и абстракцию метода заполнения этих полей.

3. «MP3Metadata» – компонент, являющийся наследуемым классом от «BaseMetadata», который переопределяет метод заполнения полей в зависимости от типа файла, в данном случае mp3. Компонент взаимодействует с другими компонентами для обработки аудио.

4. «Validator» – компонент, который представляет собой набор функций, отвечающих за определение «валидности» файла. Все функции возвращают «bool» значение, для определения прохождения той или иной проверки. Если хоть одно значение отрицательное, то файл проверку не прошел. Данный компонент также работает с компонентами конфигурирования, логирования и взаимодействия с ОС.

5. «MongoDB» – компонент для взаимодействия с одноименной базой данных, представляющий собой наследуемый класс от «AbstractBase», определяющий методы для отправки, получения и агрегации документов «MongoDB» через компонент «MongoClient». Для хранения файлов не в виде документа mongo подключен компонент «GridFS».

6. «AbstractBase» – компонент абстрактного класса, для переопределения метода преобразования в словарь. Так как документы в «MongoDB» представляют собой структуру аналогичную стандарту «json» и чаще всего «сериализуются» именно как формат словаря с парами ключ-значение. Для удобства использования в методах наследуемого класса «MongoDB», данный метод переопределен.

7. «SShLoader» – компонент для загрузки аудио с удаленного сервера. Не является обязательным для работы, может использоваться перед запуском для загрузки недостающих файлов. При небольшой доработке может быть использован для создания набора данных удаленно.

3.2. Функциональные и нефункциональные требования

Функциональные требования

Данные требования необходимы для определения процессов, действий и операций, которые способна выполнить система. Часто функциональные требования равносильны понятию вариантов использования [15]. Исходя из поставленной задачи на разработку и анализа исходных данных, разрабатываемая система должна соответствовать требованиям, которые представлены ниже.

1. Система должна иметь возможность взаимодействовать с аудио файлам, как на уровне ОС, так и через БД и удаленный сервер.
2. Система должна извлекать метаданные с аудио и предоставлять их в виде удобной структуры.
3. Система должна распределять пары аудиофайл-метаданные по соответствующему алгоритму нормального распределения, где через конфигурацию указывается коэффициент распределения, будь то 30 на 70 или 50 на 50.
4. Система должна проверять файлы на допустимость их использования при создании наборов данных, для чего должен быть реализован сложный компонент валидации, проверяющий не только аудио файл, но и метаданные, которые он содержит.
5. Система должна обладать компонентом логирования, который будет уведомлять о прогрессе загрузки файлов, формирования набора данных, включающего извлечение метаданных и прохождение «валидации».
6. Система должна извлекать метаданные разными способами, чтобы соответствовать требованиям качества подаваемых впоследствии данных на обучение модели.

Нефункциональные требования

Эти требования накладывают свойства и ограничения на разрабатываемую систему [16]. Нефункциональные требования служат для создания системы с приемлемым качеством.

В дополнение к выполнению поставленных перед системой задач, а также рассмотрения инструментов для реализации системы, представленных во второй главе, к ней были сформированы следующие нефункциональные требования.

1. Система должна обладать простой и удобной формой конфигурации, запуска и развертывания.
2. Основная программная реализация осуществляется с использованием языка программирования Python и адаптирована для интеграции под модель «EnCodec» входящую в «AudioCraft» от facebook.
3. Система должна иметь возможность работы с базой данных MongoDB.
4. Система должна иметь возможность для расширения взаимодействия с файлами других форматов и типов.

3.3. Структура метаданных

Так как частью основной задачи разрабатываемой системы является извлечение метаданных из аудио файлов, то необходимо спроектировать структуру, которой данные будут следовать. Для осуществления этой идеи необходимо из документации определить данные какого типа и формата подаются на обучение. В репозитории «audiocraft» на «GitHub» от «facebookresearch» [17] в разделе «dataset/example» представлены образцы файлов подающихся на обучение моделей. Скриншоты образцов и репозитория представлены на рисунках 6 и 7.

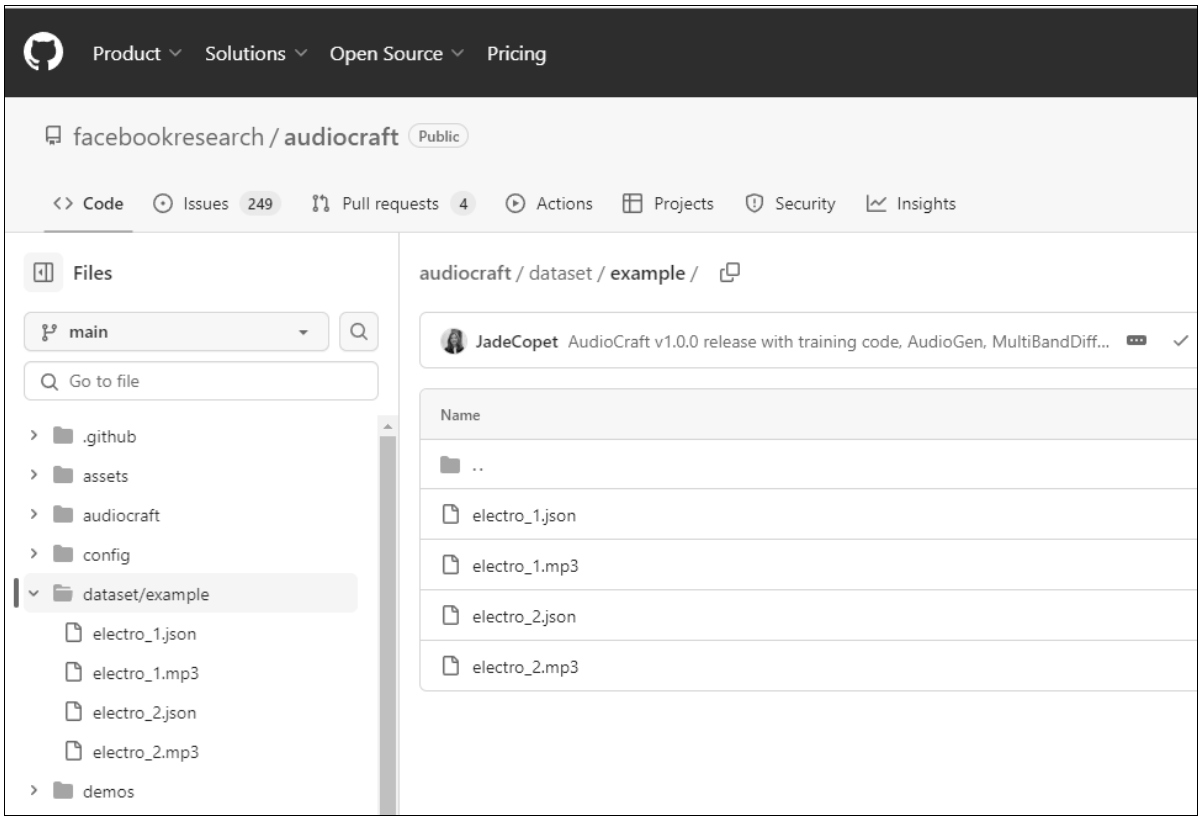


Рисунок 6 – Скриншот образцов «dataset/example»

На рисунке 6 представлен фрагмент набора данных, которые должны подаваться на обучение моделей в «audiocraft». Набор данных представляет собой пары метаданные-аудиофайл, где метаданные представлены в виде «json» файлов и аудиофайлы в формате mp3.

Рисунок 7 – Образец «electro_1.json»

На рисунке 7 представлено содержание «.json» файла, который содержит метаданные аудио с одноименным названием. Полное содержание словаря, включающее все пары ключ-значение, содержится в листинге 1.

Листинг 1 – Образец метаданных

```
{
“key”: “”,
“artist”: “Voyager I”,
“sample_rate”: 48000,
“file_extension”: “mp3”,
“description”: “A cool song from Voyager. ”,
“keywords”: “bright, pulsing, cool”,
“duration”: 15.0,
“bpm”: “”,
“genre”: “electronic”,
“title”: “Enracinement”,
“name”: “electro_1”,
“instrument”: “Mix”,
}
```

Исходя из представленной информации, можно определить какие поля необходимо извлечь из аудио, так как они подаются на обучение и имеют определенный вес для модели.

Поля метаданных представлены ниже:

1) «key» – ключ (тональность);
2) «artist» – исполнитель;
3) «sample_rate» – частота;
4) «file_extension» – расширение файла;
5) «description» – текстовое описание;
6) «keywords» – ключевые слова;
7) «duration» – продолжительность;
8) «bpm» – бит;
9) «genre» – жанр;
10) «title» – заголовок;
11) «name» – название;
12) «instrument» – инструменты;

Большинство полей, например, «artist» или «name», очевидны и понятны для заполнения, они в свою очередь имеют наименьший вес или не

имеют его совсем. Наибольший вес для моделей по генерации музыки имеет поле «description» – текстовое описание. Для модели «EnCodec» наибольший вес представляют поля технического характера, такие как: «key», «sample_rate», «file_extension», «bpm» и «duration».

3.4. Дополнительные компоненты

Исходя из поставленной задачи, для системы также необходимо спроектировать и реализовать дополнительные компоненты, которые будут использоваться вне работы основной системы для решения вспомогательных задач. Основные компоненты системы также будут рассмотрены более подробно в главе 4 «Реализация».

Компонент «SShLoader»

«SShLoader» будет использоваться для загрузки аудио с сервера. В основе данного компонента необходимо использовать протоколы SSH и SFTP, а также настроен криптографический алгоритм шифрования RSA.

SSH – сетевой протокол, позволяющий устанавливать безопасное соединение и управлять операционной системой, а также открывать TCP туннели. SSH полностью зашифровывает трафик посредством различных алгоритмов шифрования, которые можно подключить по своему усмотрению.

RSA – алгоритм шифрования, в основе которого лежит задача вычислительной сложности факторизации больших чисел. В алгоритме RSA используется 2 ключа – публичный и приватный. Публичный ключ необходим для передачи по открытым каналам связи, а приватный необходим для расшифровки сообщения.

SFTP канал – это специальный протокол передачи файлов, который работает только поверх установленного безопасного соединения, например, SSH. Протокол SFTP позволяет удаленно работать с файлами, включая операцию удаления, а также ряд особенностей, например, возобновление прерванной передачи.

Для реализации компонента необходимо установить защищенное соединение SSH, посредством подключения RSA ключей. При успешном подключении необходимо открыть SFTP канал, через который будет осуществляться загрузка файлов по определенному пути, который задается через конфигурационный файл.

Так как процесс выполнения загрузки файлов не моментальный и ограничен пропускной способностью, то необходимо реализовать функцию, которая будет отслеживать процесс загрузки аудио и отображать данную информацию в виде информативной панели. Самый распространенный и удобный для использования вариант – это библиотека «tqdm». Она обладает всем необходимым функционалом для реализации данной подзадачи.

Компонент «MongoDB»

«MongoDB» одноименный компонент для работы с базой данных MongoDB. Для работы с этой базой данных необходимо использование библиотеки «pymongo». Она обеспечивает всем необходимым функционалом для установки соединения с «MongoClient», посредством amqp-строки, которую можно задать через конфигурационный файл.

Конечно, как и у многих современных систем у MongoDB есть графический интерфейс, через который удобно осуществлять все необходимые операции. Однако, для того чтобы сделать это методами Python, необходимо определить операции, которые будут использованы для взаимодействия с базой данных.

Необходимые функции для работы с документами в базе данных:

1) get – получить документ;
2) get_items – получить множество документов;
3) update – обновить документ (добавить новые данные);
4) push – добавить документ;
5) aggregate – сложный запрос (осуществляется через «pipeline» с использованием запросов).

Для обеспечения корректной работы с файлами, в MongoDB встроена файловая система GridFS. Она необходима для хранения и извлечения больших файлов (размером большим, чем стандартный документ на 255 КБ), таких как изображения, аудиофайлы, видеофайлы. Файловая система работает на основе технологии разделения больших файлов на части. GridFS по умолчанию использует две коллекции «fs.files» для хранения метаданных и «fs.chunks» для хранения чанков (частей) файла. Файл внутри коллекции fs.files является родительским документом и содержит поле «_id». Поле «files_id» в документе «fs.chunks» устанавливает связь между чанком и его родителем.

Необходимые функции для работы с файлами в базе данных:

1) save – для сохранения файлов;
2) load – для загрузки.

Благодаря реализации всех запланированных функций, можно будет легко оперировать как документами, содержащими метаданные аудио, так и самими аудиофайлами, которые будут храниться в той же базе с использованием GridFS.

Вывод по третьей главе

Третья глава посвящена проектированию системы. В ней были определены функциональные и нефункциональные требования к системе. В качестве представления системы была выбрана и реализована упрощенная диаграмма компонентов с краткими описаниями каждого компонента, и чем он будет представлен в системе. Отдельно были рассмотрены демообразцы обучающего набора данных для определения структуры, которую необходимо получить при реализации генератора, а также структуры метаданных, которые подаются вместе с аудио на обучение в виде «.json» файла. В завершающем этапе проектирования были рассмотрены технологии, которые будут использованы при реализации дополнительных компонентов системы, а также определены соответствующие методы для взаимодействия с базой данных.

4. РЕАЛИЗАЦИЯ

4.1. Структура проекта. Основные компоненты

Первым этапом реализации системы является инициализация проекта. Для этого был создан пустой проект в IDE PyCharm. К пустому проекту была добавлена виртуальная среда выполнения, с подключенным к ней интерпретатором языка Python версии 3.10. Для удобства работы со сложными системами, часто прибегают к разделению проекта на различные модули, которые могут быть представлены в разных документах и классифицированы по директориям с определенными интуитивно понятными названиями. Директории и их назначение описано ниже.

1. Директория «config» – директория необходимая для структурирования и хранения конфигурационных файлов. Директория содержит основной файл конфигурации «config.yaml», инициализирующий маршруты к конфигурационным файлам всего проекта. Файлы конфигурации также содержатся в поддиректориях, классифицированных по различным задачам.

2. Директория «lib» – директория, которая чаще всего используется для хранения не исполняемых структурных файлов. В директории расположен файл «meta.py», в котором содержится наследуемый класс, переопределяющий метод подготовки метаданных по соответствующей структуре. Также содержится файл «tonalities.json», который необходим для определения тональности.

3. Директория «models» – директория, инициализирующая базовые модели классов, по которым определяются наследуемые классы. Файл «base.py» содержит два датакласса: «AbstractBase» и «BaseMetadata».

4. Директория «service» – основная директория всего проекта, которая содержит два микросервиса – «downloader.py» (загрузчик) и «generator.py» (генератор).

5. Директория «utils» – директория для хранения модулей системы, которые используются для выполнения мелких задач в сервисных файлах.

Директория содержит модуль «валидации» аудио-файлов «check_mp3.py», модуль для работы с базой данных «mongo.py» и разные вспомогательные функции в файле «helpers.py».

В проекте существует файл «test.py» для проверки работы системы и выполнения различных тестовых сценариев, а также файл «requirements.txt», необходимый для развертывания системы и файл «readme.txt», который содержит всю необходимую документацию по работе и установке системы. Файл «setup.py» необходим для запуска установки и содержит установочные параметры, такие как название, версия и библиотеки, с которыми система будет отображаться как единый целый модуль. Итоговая структура проекта представлена на рисунке 8.

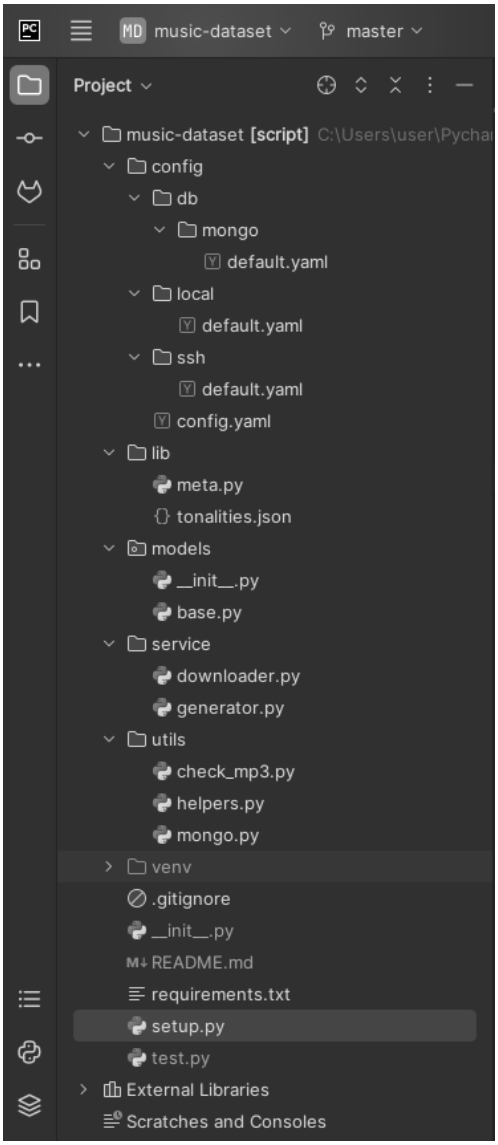


Рисунок 8 – Структура проекта

4.2. Конфигурирование и логирование

Для того чтобы система отвечала всем современным стандартам разработки, а также удовлетворяла требованиям, необходимо использование конфигурирования. Конфигурирование системы, обладает широким спектром преимуществ, которые перечислены ниже.

1. Параметры приложения. Посредством использования конфигурационных файлов можно передавать различные параметры, такие как адреса серверов, настройки баз данных, пути к файлам и другую информацию.

2. Удобство настройки. Настройки приложения можно отделить от его исходного кода. Это делает процесс настройки более гибким, так как настройки могут быть изменены без изменения самого кода.

3. Поддержка различных сред. Для различных сред выполнения можно создавать отдельные наборы параметров (например, разработка, тестирование, производство). Это упрощает развертывание и управление приложением в разных окружениях.

4. Безопасность. Ключи API или пароли, можно сохранить в защищенных конфигурационных файлах с помощью ограничений доступа к файлам или шифрования.

5. Поддержка множества форматов. Широкий выбор различных форматов конфигурационных файлов, таких как JSON, YAML, INI, XML и других.

В качестве инструмента конфигурирования в системе используется библиотека Hydra [18]. «Hydra» – это открытый фреймворк на языке Python, который упрощает разработку исследований и других сложных приложений. Основной особенностью является возможность динамического создания иерархической конфигурации путем композиции и переопределения ее через файлы конфигурации и командную строку. Иерархическая конфигурация представлена на рисунках 9, 10.

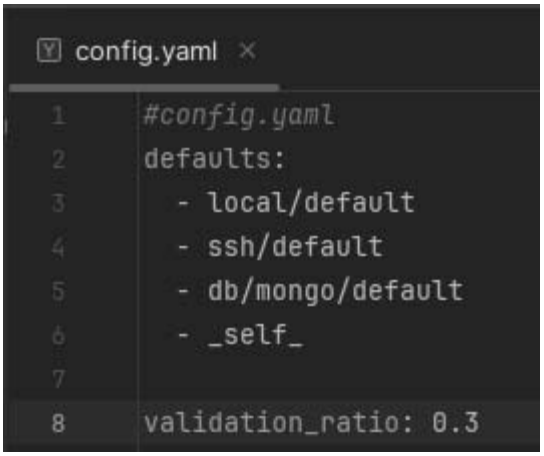


Рисунок 9 – Конфигурация «config.yaml»

На рисунке 9 представлен основной файл конфигурации, который содержит пути инициализации к другим конфигурационным файлам, а также параметры, которые не были никак классифицированы.

Рисунок 10 – Конфигурация «default.yaml» каталога local

На рисунке 10 представлен файл конфигурации «default.yaml» используемый генератором, который содержит пути к директории с аудиофайлами, а также директории со сгенерированными метаданными. Конфигурационные файлы других модулей содержатся в приложении Б.

Для использования Hydra в системе необходима аннотация `@hydra.main()`, которая указывает, что данная функция или класс является точкой входа для запуска приложения, и использует конфигурационные файлы, расположенные в определенном каталоге.

Пример тестовой `main()` функции представлен на рисунке 11.

Рисунок 11 – Аннотация `@hydra.main()`

На рисунке 11 в аннотации `@hydra.main()` используются параметры `config_path` и `config_name`, которые указывают путь и название к основному файлу конфигурации. В `main()` функцию передается зарезервированный библиотекой аргумент `cfg`, который и является объектом конфигурации. Можно передавать как всю конфигурацию целиком, так и указывать конкретный раздел конфигурации, файл или даже строку внутри файла, например, `cfg.local.local_tracks`, будет ссылаться на конфигурационный файл «default.yaml» раздела «local» и строку `local_tracks`. Такая структура конфигурации очень удобна в использовании при работе с разными функциями, когда нет необходимости вызывать объект конфигурации целиком.

В качестве компонента логирования используется стандартная библиотека «logging» [11]. Логирование – это процесс записи сообщений об операциях, событиях, или о состоянии программного обеспечения в хронологическом порядке. Библиотека «logging» предоставляет гибкие средства для создания логеров, уровней логирования, форматирования сообще-

ний и настройки потоков вывода для логов. Основное преимущество стандартной библиотеки в том, что все модули Python могут участвовать в регистрации событий, что позволяет комбинировать как сообщения модулей, так и собственные логи. Пример инициализации и использования логирования представлен на рисунках 12 и 13. Также на рисунке 12 отображается наглядный пример использования объекта конфигурации, описанный выше.

```
import logging


2 usages  ivan_b +1
class DataSetGenerator:
    ivan_b +1
    def __init__(self, cfg):
        self.cfg = cfg
        self.logger = logging.getLogger(__name__)
        self.source_dir = cfg.local.local_tracks
```

Рисунок 12 – Создание объекта `logger`

Для создания и оперирования с логами используется объект логирования `logger`. На рисунке 12 логер инициализируется с именем текущего модуля `__name__`. В процессе вывода лога сообщения буду выводиться от имени заданного модуля. Это помогает в организации и отслеживании сообщений, специфичных для каждого модуля в приложении.

```
                except Exception as e:
                    self.logger.error(f"Ошибка при обработке файла {file_name}: {str(e)}")

        else:
            self.logger.info("Finish")
```

Рисунок 13 – Уровни логирования

На рисунке 13 представлены используемые уровни логирования в текущем модуле.

Ниже представлен список стандартных уровней логирования.

1. DEBUG. Используется для отладочной информации.
2. INFO. Используется для информационных сообщений.
3. WARNING. Используется для сообщений предупреждений.
4. ERROR. Используется для сообщений об ошибках.
5. CRITICAL. Используется для сообщений о критических ошибках.

При инициализации логера можно установить стандартный уровень логирования, сообщения ниже которого не будут выводиться. Например, если стоит уровень логирования `logger.info()` (второй уровень), то сообщения `logger.debug()` (первый уровень) выводиться не будут. Существуют различные настраиваемые потоки вывода, такие как в файл, в консоль или через сеть.

4.3. Модели

Четкая структуризация необходима не только для удобства работы, повышения читабельности и структуры кода, но и для расширения системы. Для этого в работе реализованы модели, которые определяют базовую структуру используемых при разработке как дополнительных, так и основных компонентов.

Модели содержатся в файле «base.py» директории «models». Для работы с данными используется библиотека «dataclasses» – это компонент в стандартной библиотеке Python, который предоставляет декоратор `dataclass`, упрощающий создание классов, предназначенных для хранения и структуризации данных.

В модуле «base.py» содержатся два класса для работы с данными `BaseMetadata` и `AbstractBase`. Для обоих классов объявлен декоратор `dataclass`. Он автоматически добавляет стандартные методы, такие как `__init__()`, `__repr__()`, `__eq__()` на основе полей, которые были определены в классе. Это сокращает объем написанного кода и делает его более понятным и лаконичным.

Также для обоих классов определен метод `dict()`, которая преобразует объект класса в словарь используя для этого вызов стандартной функции из библиотеки «dataclasses» – `asdict()`. В работе это необходимо для взаимодействия с базой данных MongoDB, которая ожидает словарь в качестве входных данных. Дополнительно данный метод можно использовать при сериализации объектов в JSON или при передаче данных в другие части программы. Код базовых классов представлен ниже на рисунках 14 и 15.

Рисунок 14 –Класс `BaseMetadata`

На рисунке 14 атрибуты класса `BaseMetadata` задают различные метаданные (`key`, `artist`, `sample_rate` и другие), которые были определе-

ны в соответствии с документацией на этапе проектирования. Для всех атрибутов указаны значения по умолчанию, чтобы в случае отсутствия данных при заполнении, поле не принимало значение None.

Метод `prepare()` определен для подготовки метаданных на основе пути к MP3 файлу, но в базовом классе он не содержит никакой логики.

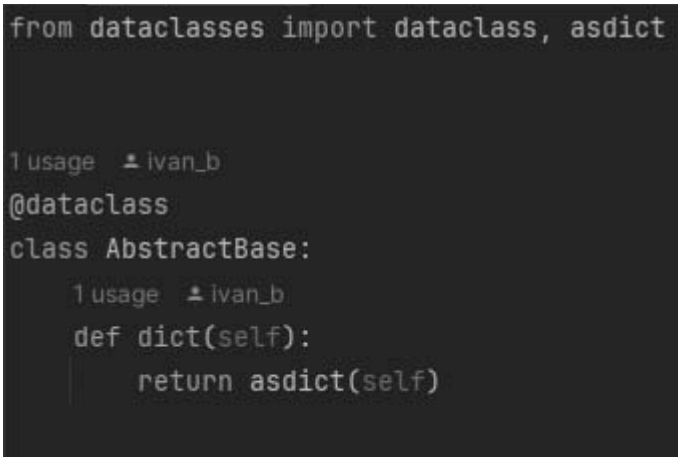


Рисунок 15 – Класс `AbstractBase`

На рисунке 15 представлен класс `AbstractBase`, в котором определен метод `dict()`, данный метод используется в методе другого класса для добавления данных, представленных в виде словаря, в базу данных.

4.4. Переопределение метода «prepare()» класса «MP3Metadata»

Основной задачей генератора набора данных, помимо разбиения исходного набора на подвыборки, является извлечение метаданных. Метаданные – это информация, хранящаяся внутри файла и описывающая его содержание, свойства и другие характеристики.

Структура метаданных была определена на этапе проектирования и реализована в базовом «датаклассе» в моделях. Для заполнения соответствующих аргументов класса необходимо извлечь метаданные из аудио файлов. Так как метаданные извлекаются уже после прохождения «валидации», то все аудио будут содержать максимальный набор тегов, хранящихся внутри файла.

Существует множество различных библиотек для извлечения метаданных из аудио. Из документации по работе с моделью «EnCodec» следует, что наиболее подходящими являются библиотеки «audio_metadata»[19] и «librosa»[20].

Библиотека «audio_metadata» – это библиотека Python, предназначенная для извлечения метаданных из аудиофайлов различных форматов, таких как MP3, WAV, FLAC, OGG, AAC. Она обеспечивает простой интерфейс для извлечения информации из аудиофайлов. Библиотека может прочитать метаданные как основных тегов, так и дополнительных, в том числе заголовок, артист, альбом, жанр, формат файла, длительность, частота дискретизации, количество каналов и другие аудио-атрибуты.

Библиотека «librosa» – это библиотека для анализа и обработки звуковых сигналов в Python, которая широко используется в области обработки аудиоданных и аудио-аналитики. Она предоставляет множество функций и инструментов для работы с аудиофайлами, включая загрузку, обработку, извлечение признаков и визуализацию звуковых данных.

На рисунках 16–17 представлен класс `MP3Metadata`, который наследуется от `BaseMetadata` и использует библиотеки «audio_metadata», «librosa» и стандартные библиотеки «os» и «json».

На рисунке 16 определены скрытые методы `_analyze_audio` и `_get_description`. Метод `_analyze_audio` необходим для заполнения атрибутов `bpm` и `key`. Для этого используется библиотека «librosa», с помощью которой читается аудио-файл и используются соответствующие методы `librosa.beat.beat_track()` и `librosa.effects.feature.tonnetz()`. Вызов этих методов анализирует загруженный трек, представленный в виде двух значений `y` – амплитудный временной ряд, `sr` – частота дискретизации.

На выходе первого метода можно получить два значения – это искомый нам `bpm` и неиспользуемый в работе массив временных точек.

Рисунок 16 – Методы `_analyze_audio` и `_get_description`

С полем тональности (ключа) немного сложнее. Метод `librosa.effects.feature.tonnetz()`вычисляет тоновые характеристики аудио-файла. Результат сохраняется в виде матрицы `tone`. `tone.sum(axis=1)` – этот вызов суммирует значения по строкам матрицы, возвращая вектор сумм для каждого столбца. `Numpy.argmax()` – этот метод находит индекс максимального элемента в полученном векторе сумм. Полученный ключ `key` вычисляется по соответствующему индексу списка тональностей, который хранится в отдельном «json»-сериализованном файле «tonalities.json». Список тональностей обычно состоит из 12 элементов, представляющих 12 различных нот и их соответствующих мажорных и минорных тональностей. Список всех тональностей представлен ниже в листинге 2.

Листинг 2 – Список тональностей

```
[
"C", "C#",
"D", "D#",
"E",
"F", "F#",
"G", "G#",
"A", "A#",
"B"
]
```

Метод `_get_description()` предназначен для получения описания MP3 файла. Он строит путь к файлу описания, предполагая, что он находится в том же каталоге, что и MP3 файл, и имеет имя «description.txt». Если файл описания существует, его содержимое читается и сохраняется в атрибут `description`. Если файл описания не найден, атрибут `description` остается пустым. Файлы описаний заранее сгенерированы различными методами, в том числе вручную и с помощью нейросетевых чат моделей. Пример описания: «A positive Solfeggio Frequency, used by many for meditation, manifestation and healing».

На рисунке 17 представлен основной метод класса `MP3Metadata` – метод `prepare()`. Метод определен декоратором `@classmethod`, что обозначает, что метод является классовым методом, а не методом экземпляра. Классовые методы принимают первым аргумент `cls`, который ссылается на сам класс, а не на экземпляр.

Основная задача метода `prepare()` это заполнение атрибутов класса. Для этого используется библиотека `audio_metadata`, которая открывает доступ к тегам метаданных, после прочтения аудиофайла. Загрузка тегов происходит через метод `load()` в конструкции `try–except audio_metadata.UnsupportedFormat()`, которая позволяет обрабатывать исключения, если поступил файл неизвестного формата. Далее создается экземпляр класса `MP3Metadata`, внутри которого для каждого атрибута вызывается соответствующий метод прочтения тега, например, для атрибута

sample_rate, осуществляется вызов file_audio_metadata.streaminfo.sample_rate, который возвращает значение частоты дискретизации.

Для атрибута исполнителя artist – это будет file_audio_metadata.tags.get("artist", [""])[0], где вызов метода get() пытается получить значение метаданных для ключа artist из словаря tags. Если ключ artist присутствует в словаре, метод get() вернет его значение. Если ключ отсутствует, будет возвращено значение по умолчанию. Индекс [0] используется для получения первого элемента списка значений метаданных словаря tags.

```python
import os
import json
import librosa
import audio_metadata

from models.base import BaseMetadata


2 usages  ivan_b +2 *
class MP3Metadata(BaseMetadata):
    1 usage  ivan_b +2 *
    @classmethod
    def prepare(cls, mp3_path):
        try:
            file_audio_metadata = audio_metadata.load(mp3_path)
        except audio_metadata.UnsupportedFormat:
            return None

        instance = cls(key="",
                       artist=file_audio_metadata.tags.get("artist", [""])[0],
                       sample_rate=file_audio_metadata.streaminfo.sample_rate,
                       file_extension=os.path.splitext(mp3_path)[-1:][0].lstrip('.'),
                       description='',
                       keywords="",
                       duration=file_audio_metadata.streaminfo.duration,
                       bpm=0,
                       genre=file_audio_metadata.tags.get("genre", [""])[0],
                       title=file_audio_metadata.tags.get("title", [""])[0],
                       name=os.path.splitext(os.path.basename(mp3_path))[0],
                       instrument=file_audio_metadata.tags.get("album", [""])[0],
                       )

        instance._analyze_audio(mp3_path)
        instance._get_description(mp3_path)
        return instance

```

Рисунок 17 – Переопределения метода prepare() класса MP3Metadata

4.5. Генератор «DataSetGenerator» и загрузчик «SShLoader»

«DataSetGenerator»

Основной компонент в работе, который объединяет в себе все этапы создания набора данных – это «DataSetGenerator». Модуль использует 4 стандартных компонента, представленных в Python.

1. Библиотека «os» – используется для работы с операционной системой.
2. Библиотека «json» – используется для работы с объектами формата json.
3. Библиотека «shutil» – используется для операции с файловой системой.
4. Библиотека «logging» – используется для логирования.

Исходя из заданных функциональных требований, система должна распределять данные по подвыборкам, используя равномерное нормальное распределение. Для реализации данного требования в работе используется функция библиотеки «numpy» – «uniform» [21]. Функция «uniform» содержится в модуле «numpy.random» и используется для генерации случайных чисел с нормальным равномерным распределением. Данная функция полезна для решения задач, связанных с моделированием, статистическим анализом и машинного обучения.

Функция принимает на вход три значения.

1. Параметр «low» – нижняя граница интервала. Генерируемые числа будут больше или равны этому значению. Стандартное значение – 0.
2. Параметр «high» – верхняя граница интервала. Генерируемые числа будут меньше этого значения. Стандартное значение – 1.
3. Параметр «size» – форма возвращаемого массива. Если не указано, возвращается одно случайное число. Может быть целым числом, кортежем целых чисел или None.

Генератор использует «датакласс» `MP3Metadata` для формирования метаданных, а также модуль для прохождения «валидации» `check_mp3`.

Импорт соответствующих компонентов и инициализация класса представлена на рисунке 18.

```python
import os
import json
import shutil
import logging
from lib.meta import MP3Metadata
from numpy.random import uniform
from utils.check_mp3 import is_valid


2 usages  ivan_b +1 *
class DataSetGenerator:
    ivan_b +1
    def __init__(self, cfg):
        self.cfg = cfg
        self.logger = logging.getLogger(__name__)
        self.source_dir = cfg.local.local_tracks
        self.directories = [os.path.join(self.source_dir, "validation"),
                            os.path.join(self.source_dir, "train"),
                            os.path.join(self.source_dir, "removed")]
        for directory in self.directories:
            os.makedirs(directory, exist_ok=True)

        self.validation_ration = cfg.local.validation_ratio

        self.metadata = None
        self.chosen_directory_file_path = None

```

Рисунок 18 – Импорт библиотек и инициализация `DataSetGenerator`

На рисунке 18 отображены все используемые библиотеки и компоненты при реализации `DataSetGenerator`. Класс принимает на вход объект конфигурации `cfg`, который содержит один путь до директории с аудиофайлами и специальный коэффициент `validation_ratio`, в соотношении которого будет производиться разбиение на подвыборки.

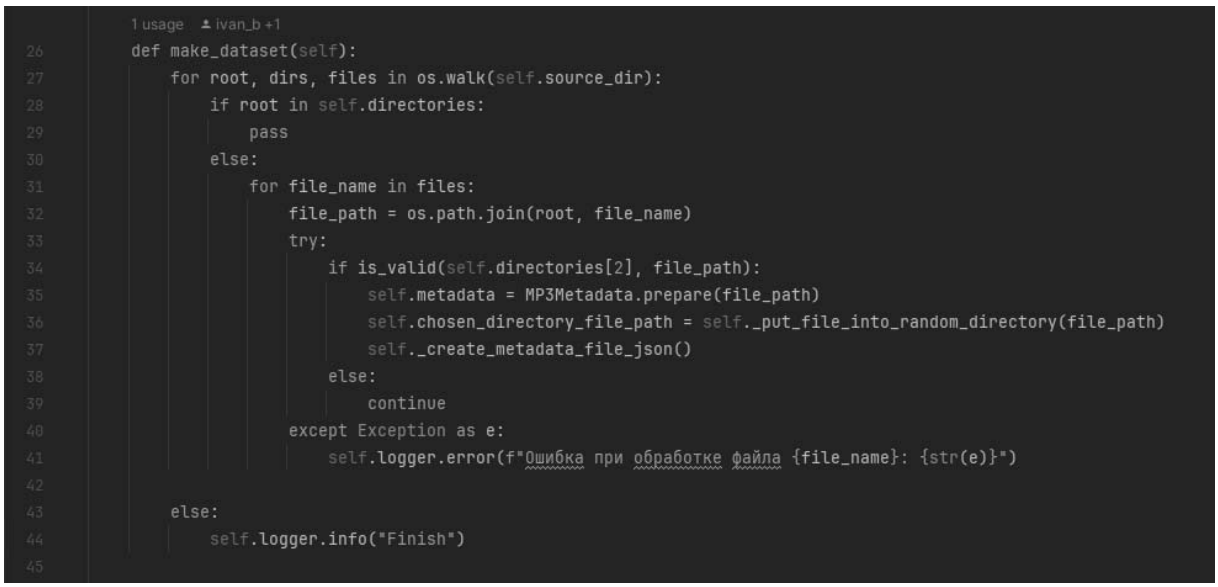
На основе объекта конфигурации производиться инициализация класса. В вызове метода `__init__()` производиться инициализация компонента логирования, директорий валидации, обучения и файлов, не прошедших проверку, а также коэффициента разделения и служебные атрибуты `metadata` и `chosen_directory_file_path`, чтобы при тестировании

можно было обратиться и получить метаданные или посмотреть сгенерированный путь до выбранной директории для пары аудио-метаданные.

У класса существует три реализованных метода, два из которых закрытые и один открытый для вызова процесса создания.

Рассмотрим открытый метод `make_dataset()` и закрытый `_put_file_into_random_directory()`. Скриншоты этих методов представлены на рисунках 19 и 20.

Скриншот метода `_create_metadata_file_json()` по отдельному сохранению метаданных в формате «json» находится в приложении Б.



```
1 usage   ivan_b +1
def make_dataset(self):
    for root, dirs, files in os.walk(self.source_dir):
        if root in self.directories:
            pass
        else:
            for file_name in files:
                file_path = os.path.join(root, file_name)
                try:
                    if is_valid(self.directories[2], file_path):
                        self.metadata = MP3Metadata.prepare(file_path)
                        self.chosen_directory_file_path = self._put_file_into_random_directory(file_path)
                        self._create_metadata_file_json()
                    else:
                        continue
                except Exception as e:
                    self.logger.error(f"Ошибка при обработке файла {file_name}: {str(e)}")

    else:
        self.logger.info("Finish")

```

Рисунок 19 – Метод `make_dataset()`

На рисунке 19 представлен метод `make_dataset()`. Данный метод вызывается после инициализации объекта класса и является своеобразной стартовой командой для выполнения генерации набора данных. В цикле вызывается проход с помощью метода `os.walk()` по всем файлам для всех директорий в корневом каталоге, исключая директории, созданные под выборки «validation», «train» «removed».

Далее в конструкции `try–except` вызываются различные методы и функции, в результате которых мы получаем метаданные аудио-файла в атрибуте `self.metadata` и путь до выбранной директории в атрибуте

self.chosen_directory_file_path. Ниже представлено краткое описание вызываемых функций и методов в теле конструкции.

1. Функция is_valid() – это основная функция для проверки файла на соответствие критериям набора данных. Она принимает два параметра – это путь до файла, который нужно проверить и директорию, в которую его переместить, если он не отвечает требованиям.

2. Метод класса MP3Metadata.prepare() – это вызов метода извлечения метаданных из файла, путь к которому передается в качестве аргумента.

3. Закрытый метод класса _put_file_into_random_directory() – это вызов метода текущего класса DataSetGenerator для генерации случайной выборки, который аналогично принимает путь до файла в качестве аргумента на вход. Метод представлен на рисунке 20.

```
1 usage   ivan_b
def _put_file_into_random_directory(self, file_path):
    if uniform() > self.validation_ration:
        chosen_directory = self.directories[1]
    else:
        chosen_directory = self.directories[0]

    file_name = os.path.basename(file_path)
    destination_path = os.path.join(chosen_directory, file_name)

    try:
        shutil.move(file_path, destination_path)
        return destination_path
    except PermissionError as e:
        self.logger.error(f"Ошибка доступа: {e}")

```

Рисунок 20 – Метод _put_file_into_random_directory()

На рисунке 20 представлен код метода _put_file_into_random_directory(). Метод использует три вспомогательных компонента для работы: «numpy.random.uniform», «os», «shutil». Если сгенерированная случайная нормальная величина в границах от 0 до 1 больше, чем порог, то файл отправляется в директорию «train», иначе в директорию «validation».

Таким образом, при любом количестве элементов, подающихся на формирование набора данных, разбиение на выборки всегда будет случайным, и соответствовать процентному соотношению, заданному через конфигурацию. В результате файл перемещается в указанную ему директорию и возвращается путь к этой директории.

«SShLoader»

При необходимости создания набора данных локально, можно использовать дополнительный модуль загрузки файлов с удаленного сервера – `SShLoader`. Для работы с технологиями безопасного удаленного соединения SSH и SFTP, используется специальная библиотека «paramiko» [22]. Она содержит необходимые классы и методы для установления соединения и открытия sftp-канала. Так же для реализации метода прогрессии загрузки, как и планировалось, была использована библиотека «tqdm» [23]. Импорты соответствующих библиотек и инициализация модуля представлены на рисунке 21.

```
import os
import logging
import paramiko
from tqdm import tqdm


1 usage  ivan_b +1
class SShLoader:
    ivan_b
    def __init__(self, cfg):
        self.cfg = cfg
        self.logger = logging.getLogger(__name__)
        self.ssh = paramiko.SSHClient()
        self.ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        self.downloaded_files = []
```

Рисунок 21 – Инициализация класса `SShLoader` и импорты библиотек

Как для других компонентов системы, при инициализации используется только передаваемый объект конфигурации `cfg`. Аналогично другим компонентам, инициализируется объект логирования – `logger`.

При вызове конструктора класса создается объект SSH-клиента `self.ssh = paramiko.SSHClient()`. Этот объект представляет собой клиент SSH, который позволяет установить соединение с удаленным сервером и выполнить на нем различные команды. Также для того чтобы устанавливать соединения с множеством различных хостов, используется автоматическая политика обработки отсутствующих ключей. Код `self.ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())`, позволяет автоматически добавлять ключи новых хостов в список доверенных при подключении к хосту в первый раз.

Атрибут `self.downloaded_files` – служебный и необходим в тестовых сценариях. Он содержит список путей к загруженным файлам.

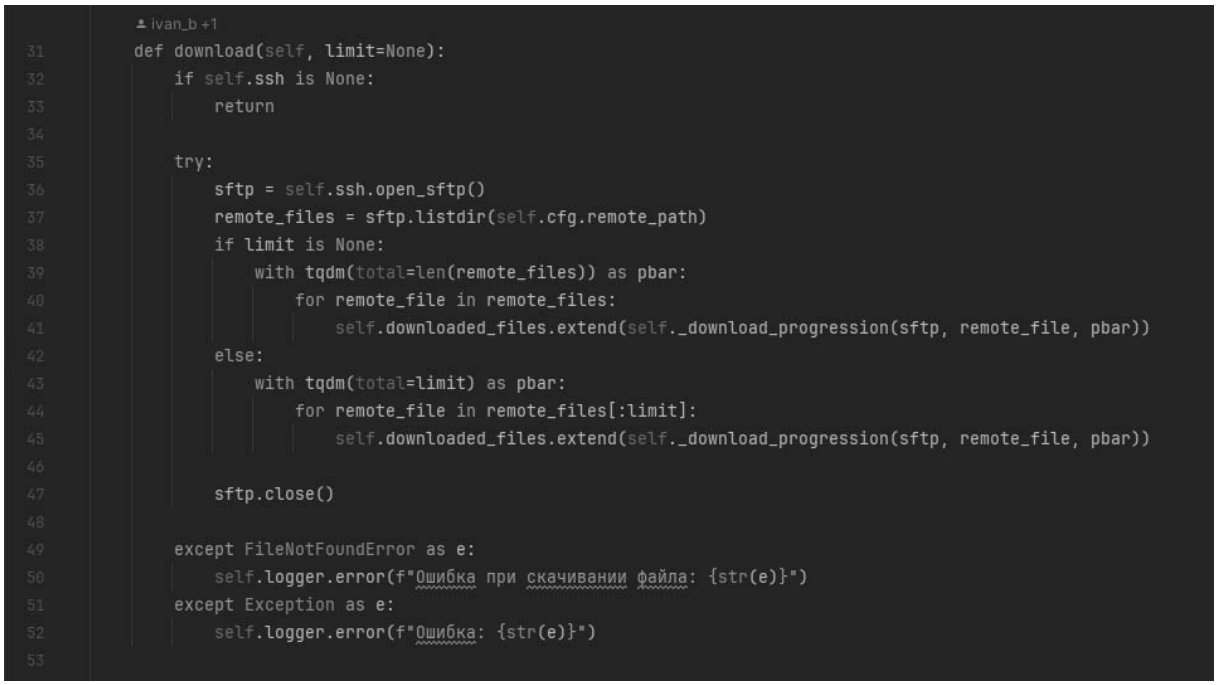
У класса три основных метода, два открытых и один закрытый. Рассмотрим открытые методы `connect()` и `download()`, которые представлены на рисунках 22 и 23. Код метода `_download_progression()` находится в приложении Б.

```
ivan_b +1
def connect(self):
    private_key = paramiko.RSAKey.from_private_key_file(self.cfg.private_key_path)
    try:
        self.ssh.connect(self.cfg.host, port=self.cfg.port, username=self.cfg.user, pkey=private_key)
        self.logger.info("SSH подключение установлено.")
        return self.ssh
    except paramiko.AuthenticationException:
        self.logger.error("Ошибка аутентификации. Проверьте ваш приватный ключ или пароль.")
        return None
    except paramiko.SSHException as e:
        self.logger.error(f"Ошибка SSH: {str(e)}")
        return None
    except Exception as e:
        self.logger.error(f"Ошибка: {str(e)}")
        return None
```

Рисунок 22 – Метод `connect()`

На рисунке 22 отображено, что при вызове метода `connect()`, создается приватный ключ из файла с ключом, путь к которому задан посредством конфигурации. С помощью метода `connect()` объекта `SSHClient`, в который передаются аргументы `host`, `port`, `username` и `private_key`,

устанавливается соединение с сервером. Весь процесс обернут в конструкцию `try–except` для обработки исключений во время установления соединения.



```
ivan_b +1
def download(self, limit=None):
    if self.ssh is None:
        return

    try:
        sftp = self.ssh.open_sftp()
        remote_files = sftp.listdir(self.cfg.remote_path)
        if limit is None:
            with tqdm(total=len(remote_files)) as pbar:
                for remote_file in remote_files:
                    self.downloaded_files.extend(self._download_progression(sftp, remote_file, pbar))
        else:
            with tqdm(total=limit) as pbar:
                for remote_file in remote_files[:limit]:
                    self.downloaded_files.extend(self._download_progression(sftp, remote_file, pbar))

        sftp.close()

    except FileNotFoundError as e:
        self.logger.error(f"Ошибка при скачивании файла: {str(e)}")
    except Exception as e:
        self.logger.error(f"Ошибка: {str(e)}")

```

Рисунок 23 – Метод `download()`

На рисунке 23 отображен код метода `download()`. В качестве параметра, можно передать значение `limit`, по умолчанию, которое равно «None». Если параметр `limit` остается по умолчанию, он загружает все файлы из удаленной директории, в противном случае загружает только указанное количество файлов.

Блок `if self.ssh is None` проверяет, инициализирован ли объект SSH-клиента. Если нет, метод завершает свое выполнение, поскольку без подключения к удаленному серверу загрузка файлов невозможна.

SFTP-соединение открывается с удаленным сервером с использованием объекта SSH-клиента `sftp = self.ssh.open_sftp()` и получает список файлов на удаленном сервере в указанной директории из объекта конфигурации `remote_files = sftp.listdir(self.cfg.remote_path)`.

Для загрузки файлов в режиме прогрессии вызывается внутренний метод `_download_progression()`, который загружает файл с удаленного

сервера с помощью метода `sftp.get()` и обновляет прогресс-бар. Загруженные файлы добавляются в список `downloaded_file`, после чего sftp-канал закрывается вызовом метода `sftp.close()`. Обработка ошибок производится аналогично методу `connect()`.

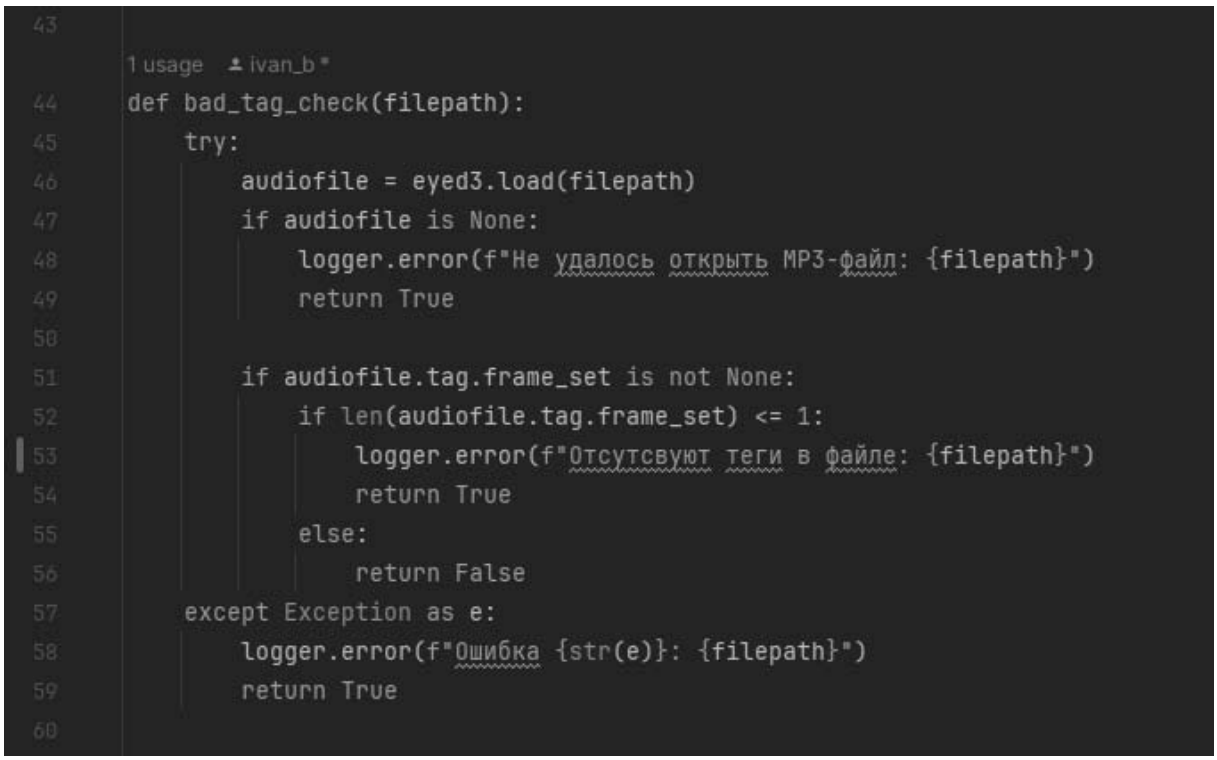
4.6. Валидация файлов и работа с «MongoDB»

Функция «is_valid()»

Один из ключевых моментов формирования набора данных – это проверка данных на соответствие заданным требованиям. Для обучения нейросетевого кодека «EnCodec», большинство полей технического характера из структуры метаданных имеют вес для модели. Однако, даже при таких условиях, чтобы система была максимально универсальной и подходила к другим моделям «AudioCraft», необходимо чтобы все поля были максимально заполнены.

В ходе тестовых запусков системы без функции предварительной проверки был выявлен ряд ошибок, возникающих как при извлечении метаданных, так и при обучении модели.

Самая распространенная ошибка, которая возникала на этапе извлечения метаданных, это то, что многие поля оставались незаполненными в связи с отсутствием тегов, в метаданных. Для решения данной проблемы, была реализована функция `bad_tag_check()`. Она использует мощный инструмент для работы с аудиофайлами – библиотеку «eyed3» [24]. Данная библиотека способна не только прочитать информацию из существующей меты файлов, но и реализовано специальное средство «eyeD3», которое позволяет работать с командной строкой и вшивать метаданные в файл. Функция `bad_tag_check()` представлена на рисунке 24.



```
1 usage  ivan_b*
def bad_tag_check(filepath):
    try:
        audiofile = eyed3.load(filepath)
        if audiofile is None:
            logger.error(f"Не удалось открыть MP3-файл: {filepath}")
            return True

        if audiofile.tag.frame_set is not None:
            if len(audiofile.tag.frame_set) <= 1:
                logger.error(f"Отсутсвуют теги в файле: {filepath}")
                return True
            else:
                return False
    except Exception as e:
        logger.error(f"Ошибка {str(e)}: {filepath}")
        return True
```

Рисунок 24 – Функция `bad_tag_check()`

На рисунке 24 отображена функция `bad_tag_check()`. Для загрузки аудио-файла на проверку используется функция `load()`, которая принимает путь до файла. Далее выполняется двухуровневая проверка файла.

Первый этап `if audiofile is None` – это проверка открытия, потому что иногда при больших объемах данных, встречаются битые файлы.

Второй этап `if audiofile.tag.frame_set is not None` – это проверка на наличие тегов. Для этого читается отдельный тег «frame_set», который содержит информацию обо всех содержащихся тегах в метаданных. Если данный тег отсутствует, то тегов в мете нет.

Финальная проверка это количество элементов в теге «frame_set», данный тег содержит либо все теги, которые есть в метаданных, либо содержит один, который говорит что тегов нет, поэтому проверка `if len(audiofile.tag.frame_set <= 1)`.

В результате, если файл читается и содержит хотя бы несколько читабельных тегов, то возвращается значение «False», что значит, что файл прошел проверку.

Альтернативная и тоже довольно распространенная ошибка возникала на этапе обучения моделей. При работе с «audiocraft» для обучения моделей используется инструмент «ffprobe» [25] – это консольная утилита, позволяющая собирать и отображать информацию о медиафайлах и мультимедиапотоках, кодеках, форматах и прочего. Поэтому на этапе обучения не редко возникала ошибка «Incorrect BOM value. Error reading comment frame», которая приводила к критической ошибке и завершению выполнения.

Так как «ffprobe» работает с файлами из набора данных, а не с самой моделью, была реализована функция `check_comment_frame()`, с использованием встроенной библиотеки для работы с консолью «subprocess». Данная функция представлена на рисунке 25.

```
1 usage   ivan_b
def check_comment_frame(filepath):
    command = ['ffprobe', '-v', 'error', '-select_streams', 'a:0',
               '-show_entries', 'stream_tags=comment', '-of',
               'default.yaml=noprint_wrappers=1', filepath]

    result = subprocess.run(command, stderr=subprocess.PIPE, text=True)
    if "Incorrect BOM value\nError reading comment frame, skipped\n" in result.stderr:
        logger.error(f"Incorrect BOM value. Error reading comment frame: {filepath}")
        return False

    return True
```

Рисунок 25 – Функция `check_comment_frame()`

На рисунке 25 отображена функция `check_comment_frame()`. Она принимает на вход путь к аудиофайлу. Затем строится «pipeline», для передачи его в консоль как команды на выполнение. Данная команда использует предустановленный заранее «ffprobe», для вывода ошибок при прочтении соответствующего потока медиаданных и тега комментариев в виде строки.

Если в сформированной строке ошибок, возникает ошибка, которая до этого возникала на этапе обучения, то возвращалось значение «False», иначе «True», что значит, что файл проверку прошел.

Самая малораспространенная ошибка, которая возникала на этапе чтения, это ошибка кодировки «unicode», то есть при декодировании или кодировании названия аудиофайла при прочтении или записи, возникала ошибка, при которой невозможно было прочесть или записать файл. Скриншот кода данной проверки находится в приложении.

Все виды проверок были объединены в одну логическую функцию `is_valid()`. Данная функция представлена на рисунке 26.

```
2 usages  ivan_b *
def is_valid(removed_folder, filepath):
    filename = os.path.basename(filepath)

    if filename.endswith(".mp3"):

        if not is_unicode_encoded(filename):
            move_to_removed_folder(filepath, removed_folder)
            return False

        if not check_comment_frame(filepath):
            move_to_removed_folder(filepath, removed_folder)
            return False

        if bad_tag_check(filepath):
            move_to_removed_folder(filepath, removed_folder)
            return False

        return True

```

Рисунок 26 – Функция `is_valid()`

На рисунке 26 отображено, что необходимо соблюдение всех проверок, чтобы MP3 файл прошел проверку. Функция также принимает путь к папке, где будут храниться файлы не прошедшие проверку и перемещает их туда с помощью функции `move_to_removed_folder()`, в основе которой лежит библиотека «shutil». Данная функция также представлена в приложении Б.

«MongoDB»

Компонент для работы с базой данных MongoDB, структура и необходимые методы которого были определены в разделе «Проектирование». Для реализации класса, как и планировалось, была использована библиотека «pymongo», при установке которой с использованием команды `pip install pymongo`, автоматически установится также библиотека «gridfs», которая необходима для реализации хранения файлов. Инициализация класса и используемые библиотеки представлены на рисунке 27.

```
1   from functools import singledispatchmethod
2   from pymongo import MongoClient
3   from pymongo.results import InsertOneResult
4   from models import Base
5   from gridfs import GridFS
6   import pickle
7
8
    ivan_b *
9   class MongoDB:
        ivan_b *
10      def __init__(self, cfg):
11          self.mongo_client = MongoClient(cfg.uri)
12          self.db = self.mongo_client[cfg.database_name]
13          self.fs = GridFS(self.db)
14
```

Рисунок 27 – Инициализация класса и импорт библиотек и компонентов

На рисунке 27 представлен код, который используется при инициализации класса и импортирования библиотек и компонентов. Для хранения данных в байтовом представлении используется стандартная библиотека сериализации «pickle». Для создания универсальной функции `push()`, которая сможет отправлять данные представленные разными типами, используется декоратор `@singledispatchmethod` из стандартной библиотеки «functools».

Определяются аргументы `mongo_client`, который представляет собой объект клиента `MongoClient()`, устанавливающий соединение с базой данных через строку `uri`, заданную в конфигурации, а также аргументы `db` и `fs`, которые представляют конкретную базу данных и файловую систему «GridFS» внутри нее соответственно.

Основные методы необходимые для работы системы, согласно заданной структуре, представлены на рисунках 28 и 29.

```
5 usages (5 dynamic)  ivan_b
31  def get(self, query: dict, collection: str, **kwargs):
32      return self.db[collection].find_one(query, **kwargs)
33
    ivan_b
34  def get_iter(self, query: dict, collection: str, **kwargs):
35      return self.db[collection].find( *args: query, **kwargs)
36
    ivan_b *
37  def update_document(self, query: dict, data: dict, collection: str, **kwargs):
38      return self.db[collection].update_one(query, update: {"$set": data}, **kwargs)
39
    ivan_b
40  def aggregate(self, pipeline: list, collection: str, **kwargs):
41      return self.db[collection].aggregate(pipeline, **kwargs)
42
    2 usages  ivan_b
43  @singledispatchmethod
44  def push(self, **kwargs) -> InsertOneResult:
45      raise NotImplementedError("Unknown Type: rejected")
46
    ivan_b
47  @push.register
48  def _(self, item: dict, collection: str):
49      return self.db[collection].insert_one(item)
50
    ivan_b
51  @push.register
52  def _(self, item: Base, collection: str):
53      return self.db[collection].insert_one(item.dict())
```

Рисунок 28 – Методы для работы с документами MongoDB

На рисунке 28 определены следующие методы, описанные ниже.

1. Метод `get(self, query: dict, collection: str, **kwargs)` – получает документ на основе запроса `query` из коллекции «collection» с возможностью передать другие аргументы через `**kwargs`.

2. Метод `get_iter(self, query: dict, collection: str, **kwargs)` – получить множество документов на основе запроса `query` из коллекции `collection` с возможностью передать другие аргументы через `**kwargs`, например, `limit`.

3. Метод `update_document(self, query: dict, data: dict, collection: str, **kwargs)` – обновить документ на основе запроса `query` с добавлением новых данных `{"$set": data}` в коллекцию `collection` с возможностью передать другие аргументы через `**kwargs`.

4. Универсальный метод с декоратором `@singledispatchmethod` `push(self, **kwargs)` – добавляет данные `item` в виде документа в коллекцию `collection`. Данные могут быть двух типов – это либо `item: dict`, то есть в виде словаря, либо `item: Base`, то есть экземпляр датакласса.

5. Метод `aggregate(self, pipeline: list, collection: str, **kwargs)` – сложный запрос, осуществляется через запросы на языке SQL, представленные в виде `pipeline` в коллекции `collection` с возможностью передать другие аргументы через `**kwargs`.

```
ivan_b
15  def load_binary(self, file_name):
16      m = self.fs.get_last_version(filename=file_name)
17      m_data = m.read()
18      return pickle.loads(m_data)
19
    ivan_b
20  def save_binary(self, file, file_name):
21      item = pickle.dumps(file)
22      with self.fs.new_file(filename=file_name) as f:
23          f.write(item)
24
```

Рисунок 29 – Методы для работы с файлами через «GridFS»

На рисунке 29 определены следующие методы, описанные ниже.

1. Метод `load_binary(self, file_name)` – находит в документах базы данных и загружает сериализованный файл `file_name` в байтовом представлении и возвращает его в десериализованном виде.

2. Метод `save_binary(self, file, file_name)` – сериализует файл «file» в байтовом представлении и сохраняет его с использованием файловой системы «fs» под именем `file_name` в виде документов.

4.7. Обучение модели «EnCodec»

Финальным этапом разработки является обучение нейронной сети, а точнее аудиокодека, основанного на нейросетевых технологиях «EnCodec», входящего в пакет «AudioCraft» от «facebookresearch». Код загрузки модели и обучения не нуждается в реализации и представлен в открытом доступе. Код находится в репозитории «facebookresearch/audiocraft» [17], который был рассмотрен в главах «Анализ предметной области» и «Проектирование». Фрагмент кода представлен ниже на рисунке 30.

На рисунке 30 представлен код класса `CompressionSolver` от «facebookresearch». Можно наглядно убедиться, что инициализация класса стандартна, а также используются стандартные библиотеки для решения задач обучения нейросети.

Класс инициализируется аналогично моей системе, используя объект конфигурации «cfg», который представлен в виде словаря типа «omegaconf». Данная библиотека лежит в основе библиотеки «hydra», которая была использована в моей системе. Логирование осуществляется посредством логера из библиотеки «logging», а также в работе используются библиотека для многопоточной работы «multiprocessing» и различные компоненты собственной разработки, такие как «models», «quantization», «checkpoint», «get_pool_executor» и другие.

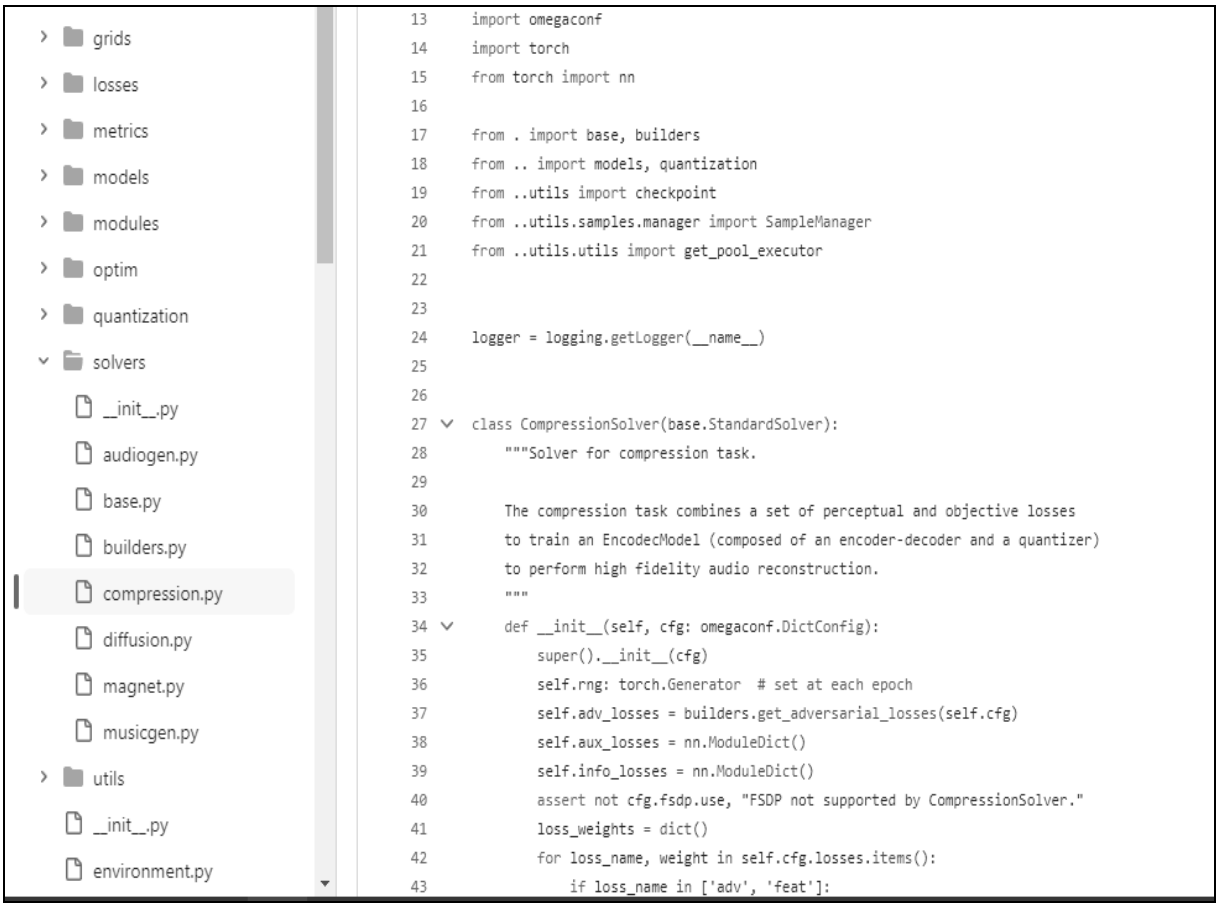


Рисунок 30 – Инициализация класса `CompressionSolver`
от «facebookresearch»

Класс `CompressionSolver` наследуется от базового класса `StandartSolver` и переопределяет стандартные методы для обучения модели, которые представлены ниже на рисунке 31.

На рисунке представлены методы для обучения модели такие как:

1) `build_model()` – метод построения модели;
2) `build_dataloaders()` – метод построения загрузчиков данных;
3) `run_step()` – метод запуска одного шага обучения;
4) `run_epoch()` – метод запуска эпохи обучения из шагов;
5) `evaluate()` – метод оценки эффективности обучения;
6) `generate()` – метод генерации новых параметров обучения.

Также в классе «CompressionSolver» содержатся методы для обучения предобученной модели и метод оценки восстановления аудио.

Рисунок 31 – Методы класса «CompressionSolver» от «facebookresearch»

Архитектура модели нейросетевого кодека «EnCodec», которая была использована для обучения, представлена в листинге 1 приложения В.

Архитектура модели содержит три основных блока – это энкодер (encoder), декодер (decoder) и квантователь (quantizer). Энкодер и декодер состоят из нескольких блоков, где каждый блок состоит из нескольких

слоев. Например, для энкодера тип модели последовательный `sequential`, нулевой блок – `StreamableConv1d` , состоящий из трех слоев `(conv): NormConv1d`, `(conv): Conv1d(1, 64, kernel_size=(7,), stride=(1,))`, `(norm): Identity()`. Первый блок `SEANetResnetBlock` тоже является последовательным (sequential) и состоит из двух пар: функции активации `ELU(alpha=1.0)` и блока `StreamableConv1d`», который описан выше. Полная архитектура содержится в листинге в приложении В.

Модель обучалась в несколько этапов по 150-200 эпох, что заняло довольно продолжительное время. Во время обучения выводились логи, отображающие процесс обучения и различные метрики. Логи за начальные и конечные эпохи представлены ниже в листингах 3–4.

Листинг 3 – Лог обучения эпохи 1

```
[10-03 09:33:27][flashy.solver][INFO] - Train Summary
Epoch 2
bandwidth=2.200
d_loss=1.923
d_msstftd=1.923
adv_msstftd=0.489
feat_msstftd=0.019
l1=0.053
msspec=0.154
penalty=0.423
ratio1=0.444
g_loss=10.429
ratio2=132.702
mel=0.776
sisnr=4.991
adv=0.489
feat=0.019
duration=31226.479
```

Листинг 4 – Лог обучения эпохи 90

```
[10-03 09:33:27][flashy.solver][INFO] - Train Summary
Epoch 90
bandwidth=2.200
d_loss=1.395
d_msstftd=1.395
adv_msstftd=0.804
feat_msstftd=0.046
l1=0.025
msspec=0.052
penalty=2.161
ratio1=0.617
g_loss=1.957
ratio2=22.191
mel=0.362
sisnr=-6.549
```

```
adv=0.804
feat=0.046
duration=3014.021
```

На примере сравнения двух эпох (2 и 90), можно сделать следующие выводы, представленные ниже.

1. Значение «d_loss» уменьшилось с 1,923 на 1,395, что указывает на улучшение качества модели в ходе обучения.

2. Значение «sisnr» сначала было положительным 4,991, а затем стало отрицательным -6,549, что указывает о том, что модель научилась лучше распознавать шум.

3. Значение «g_loss» сначала был высоким 10,429, а затем уменьшился до 1,957, что указывает на улучшение модели в ходе обучения.

4. Значение «duration» уменьшилось с 31226,479 на 3014,021 секунд, что означает лучшую оптимизацию процесса обучения.

5. Значение «adv_msstftd» уменьшилось с 0,489 до 0,804. Это указывает на то, что модель стала более устойчивой к атакам на аудио-функции.

6. Значение «feat_msstftd» увеличилось с 0,019 до 0,046. Это означает, что модель стала более точно извлекать признаки из аудио-файлов.

7. Значение «l1» уменьшилось с 0,053 до 0,025. Это указывает на снижение средней абсолютной ошибки в регрессии.

8. Значение «msspec» увеличилось с 0,154 до 0,052. Это может указывать на то, что среднеквадратичное отклонение мел-спектрограммы уменьшилось.

После завершения процесса обучения, модель была использована, для компрессии и декомпрессии файлов. Эксперименты проводились как на аудио образцах, так и на полноценных треках. Ключевая особенность сжатого образца, в его размере.

На рисунках 32 и 33 представлено сравнение размеров образца прошедшего через копрессию и оригинальной песни. Как можно увидеть, разница колоссальна и составляет 120 Кб против 7,32 Мб у оригинала (122880 байт против 7684096 байта).

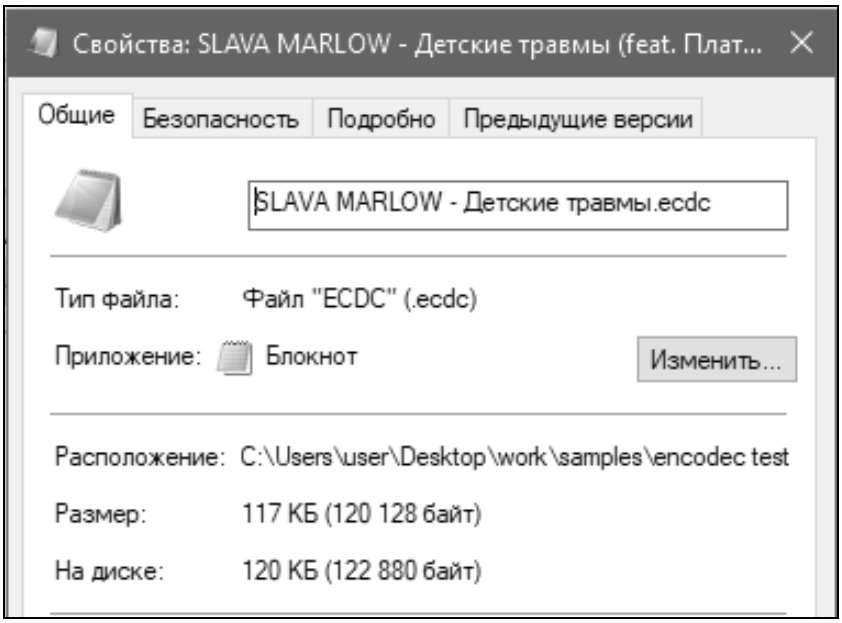


Рисунок 32 – Образец песни после применения «EnCodec»

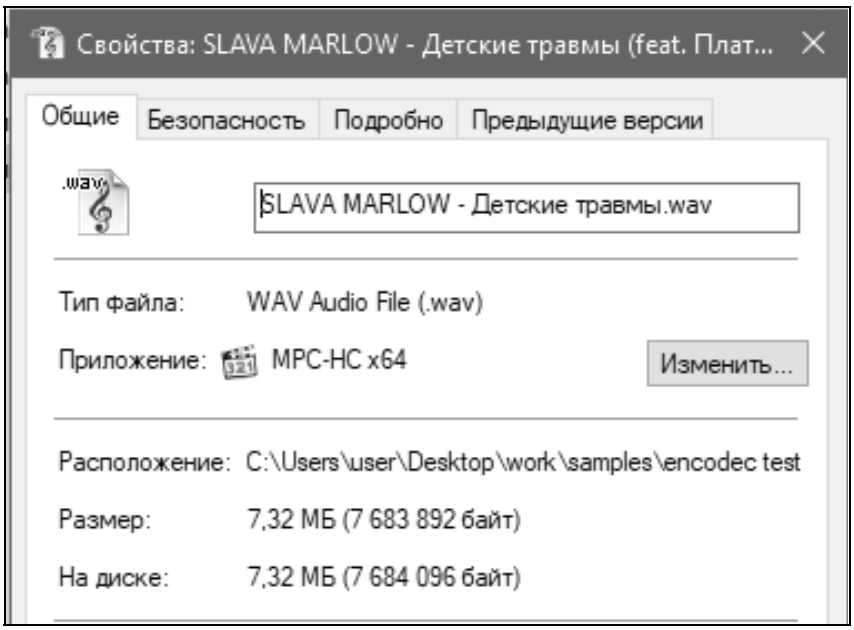


Рисунок 33 – Оригинал песни

Другой ключевой особеннсотью «EnCodec» является то, что при восстановлении файла обратно в оригинальной состояние, то есть пройдя

процесс декомпрессии с помощью той же модели, качество звука сохраняется, и разницу услышать в мелодии невозможно, а в песне, только на самых высоких частотах голоса исполнтеля.

Возможно, если продолжать обучать модель на еще более широкой выборке аудио, то и этого деффекта можно будет избежать.

Вывод по четвертой главе

В четвертой главе «Реализация» была описана разработка искомой системы согласно задуманному проекту и выявленным функциональным требованиям.

Реализация включает в себя разработку основных и дополнительных компонентов системы, подключение компонентов логирования и конфигурирования системы.

Были инициализированы классы и разработаны методы для извлечения метаданных из аудиофайлов с помощью различных библиотек, таких как «audio_metadata» и «librosa».

Был разработан класс генератор, который формирует набор данных, используя метаданные, функцию «валидации» и нормальное распределение, представленное функцией «numpy.random.uniform». Функция проверки (валидации) для генератора была разработана с использованием мощных инструментов для работы с аудио – библиотекой «eyed3» и инструментом для чтения потоковых данных «ffprobe».

Были разработаны дополнительные компоненты «SShLoader» и «MongoDB», которые позволяют работать с аудиофайлами на удаленном сервере и сохранять, как метаданные, так и сами файлы в виде документов для базы данных mongoDB.

В качестве финального результата, была обучена модель новейшего нейросетевого аудиокодека «EnCodec» от компании «facebookresearch». Работа данной модели была проверена на аудио-образцах и предоставлены результаты обучения, компрессии и декомпрессии образца.

5. ТЕСТИРОВАНИЕ

Функциональное тестирование

Тестирование системы проводилось по методу функционального тестирования. Функциональное тестирование предназначено для определения соответствия разработанного программного обеспечения заявленным требованиям [26]. Результаты тестирования представлены в таблице 1.

Таблица 1 – Функциональное тестирование

№	Тест	Ожидаемый результат	Полученный результат	Прохождение теста
1	Тестирование взаимодействия с аудио файлами средствами ОС	Система читает, перемещает и удаляет аудио-файлы средствами ОС	Результат соответствует ожиданиям	Пройден
2	Тестирование извлечения метаданных с аудио	Система извлекает метаданные с аудио	Результат соответствует ожиданиям	Пройден
3	Тестирование распределения пар аудиофайл-метаданные по алгоритму нормального распределения	Система распределяет пары аудиофайл-метаданные по алгоритму нормального распределения	Результат соответствует ожиданиям	Пройден
4	Тестирование компонента «валидации», который проверяетфайлы на допустимость их использования.	Компонент «валидации» успешно определяет плохие файлы и перемещает их в специальную директорию	Результат соответствует ожиданиям	Пройден
5	Тестирование компонента логирования и уровней логирования «debug», «info», «error»	Выводятся сообщения соответствующих уровней согласно компонентам	Результат соответствует ожиданиям	Пройден
6	Тестирование компонента конфигурации, чтобы определенные модули ожидали объект конфигурации на вход	Без заполнения конфигурационных файлов до запуска программы, вызывается ошибка конфигурирования «hydra»	Результат соответствует ожиданиям	Пройден
7	Тестирование на возникновение ошибок при выполнении программы	Ошибки при выполнении генерации и извлечения метаданных не возникают	Результат соответствует ожиданиям	Пройден

Окончание таблицы 1

№	Тест	Ожидаемый результат	Полученный результат	Прохождение теста
8	Тестирование создания необходимых директорий при генерации набора данных	Создаются необходимые директории «train», «validation» и «removed»	Результат соответствует ожиданиям	Пройден
9	Тестирование удобства развертки системы командой «pip install git+» и файла «setup.py»	Система устанавливается посредством команды «pip install git+» и модули доступны для импортирования	Результат соответствует ожиданиям	Пройден
10	Тестирование установки необходимых пакетов из файла «requirements.txt»	Все пакеты актуальных версий успешно устанавливаются	Результат соответствует ожиданиям	Пройден

Тестирование компонентов

Для тестирования компонентов, не используемых во время работы основного модуля, таких как загрузчик файлов с сервера или компонента для работы с файлами в базе данных, были проведены отдельные тестовые сценарии с подключением соответствующих компонентов: «SShLoader» и «MongoDB». Тестирование проводилось аналогично тестовым сценариям функционального тестирования, включая загрузку файлов на локальную машину или взаимодействие с файлами, которые хранятся в базе данных. Тесты пройдены успешно.

Вывод по пятой главе

В пятой главе было описано функциональное тестирование, включающее в себя 10 вариативных тестов функциональности. Также проведено тестирование компонентов на их работоспособность, что позволяет расширить вариативность использования основного модуля с файлами, не находящимися в локальном хранилище. Тестирование прошло успешно.

ЗАКЛЮЧЕНИЕ

В данной работе была реализована система, представляющая собой генератор набора данных, который преобразует необработанные аудио файлы под формат необходимый для обучения различных моделей рекомендации или генерации музыки. В процессе генерации формируется отдельный метафайл для каждого аудио, содержащий метаданные в соответствии с заявленной структурой. Генератор поддерживает работу как с файлами, хранящимися локально, так и с файлами, хранящимися на удаленном сервере, а также реализован отдельный модуль для взаимодействия с базой данных, что позволяет хранить при необходимости как аудиофайлы, так и метаданные в виде документов mongoDB. На основе сгенерированных данных была обучена и протестирована модель нейросетевого кодека «EnCodec».

Для достижения поставленной цели работы были выполнены следующие шаги:

1) проанализирована предметная область;
2) определена структура и необходимые преобразования данных;
3) налажена автоматизация с базой данных;
4) реализован генератор обучающего набора данных;
5) проведено тестирование и обучение модели на обработанных данных.

На текущий момент система успешно эксплуатируется, были получены различные наборы данных, которые впоследствии были использованы для обучения различных моделей. Акт о введении в научно-техническую эксплуатацию предоставлен по запросу.

Дальнейшая работа над системой включает расширение ее возможностей и вариативности использования с другими данными.

ЛИТЕРАТУРА

1. Gold B., Morgan N., Ellis D. Speech and Audio Signal Processing: Processing and Perception of Speech and Music. – Wiley-Interscience; 15 August 2011. – 688 с.

2. Raschka S., Mirjalili V. Python Machine Learning. – Packt Publishing, December 12, 2019. – 772 с.

3. Shanmugamani R. Deep Learning for Audio, Image and Video Analysis. – Packt Publishing, January 2018. – 310 с.

4. Salamon J., Bello J. P. Urban Sound Classification, Using Convolutional Neural Networks. – Packt Publishing, 2021. – 1099 с.

5. Freesound Datasets: A Platform for the Creation of Open Audio Datasets. [Электронный ресурс] URL: https://labs.freesound.org/datasets/ (дата обращения: 18.02.2024 г.).

6. Reiss J. D., McPherson A. Audio Effects: Theory, Implementation and Application. – CRC Press, – October 23, 2014. – 368 с.

7. Havelock D., Kuwano S., Vorländer M. Handbook of Signal Processing in Acoustics. – Softcover reprint of the original 1st ed. 2008 изд. – Springer. – August 23, 2016. – 2000 с.

8. Модель для рекомендации музыки AudioCraft от Facebook. [Электронный ресурс] URL: https://audiocraft.metademolab.com/ (дата обращения: 18.02.2024 г.).

9. API documentation for AudioCraft by facebook. [Электронный ресурс] URL: https://facebookresearch.github.io/audiocraft/api_docs /audiocraft/index.html (дата обращения: 18.02.2024 г.).

10. PyCharm: the Python IDE for Professional Developers by JetBrains. [Электронный ресурс] URL: https://www.jetbrains.com/pycharm/ (дата обращения: 18.02.2024 г.).

11. Python 3.10 documentation. [Электронный ресурс] URL: https://docs.python.org/3.10/ (дата обращения: 18.02.2024 г.).

12. MongoDB Documentation. [Электронный ресурс] URL: https://www.mongodb.com/docs/ (дата обращения: 18.02.2024 г.).

13. Буч Г., Рамбо Д., Якобсон И. Язык UML. Руководство пользователя / Пер. с англ. Мухин Н. – М.: ДМК Пресс, 2006. – 496 с.

14. UML code online. [Электронный ресурс] URL: https://www.plantuml.com/plantuml/uml/ (дата обращения: 18.02.2024 г.).

15. Маглинец Ю. А. Анализ требований к информационным системам. – 2007. – 100 с.

16. Howden W. E. Functional program testing and analysis. – New York, NY: McGraw-Hill, 1987. – Т. 2. – С. 997–1005.

17. GitHub repository for «Audiocraft» by facebookresearch. [Электронный ресурс] URL: https://github.com/facebookresearch/audiocraft (дата обращения: 18.02.2024 г.).

18. Hydra. [Электронный ресурс] URL: https://hydra.cc/docs/intro/ (дата обращения: 18.02.2024 г.).

19. Audio-metadata. [Электронный ресурс] URL: https://audio-metadata.readthedocs.io/ (дата обращения: 18.02.2024 г.).

20. Librosa. [Электронный ресурс] URL: https://librosa.org/doc/latest/index.html (дата обращения: 18.02.2024 г.).

21. Numpy. Uniform. [Электронный ресурс] URL:https://numpy.org/doc/stable/reference/random/generated/numpy.random.uniform.html (дата обращения: 18.02.2024 г.).

22. Paramiko API documentation for SSH connection. [Электронный ресурс] URL: https://docs.paramiko.org/en/latest/ (дата обращения: 18.02.2024 г.).

23. TQDM. [Электронный ресурс] URL: https://tqdm.github.io/ (дата обращения: 18.02.2024 г.).

24. Eyed3. [Электронный ресурс] URL: https://eyed3.readthedocs.io/ (дата обращения: 18.02.2024 г.).

25. Ffprobe. [Электронный ресурс] URL: https://ffmpeg.org/ (дата обращения: 18.02.2024 г.).

26. Руководство к Своду знаний по управлению проектами (Руководство PMBOK) – Пятое издание. – Project Management Institute, Inc., 2013. – 762 с.

ПРИЛОЖЕНИЯ

Приложение А. Дополнения основной диаграммы компонентов

Разрабатываемые компоненты для дополнения основной диаграммы компонентов представлены на рисунках 1–3.

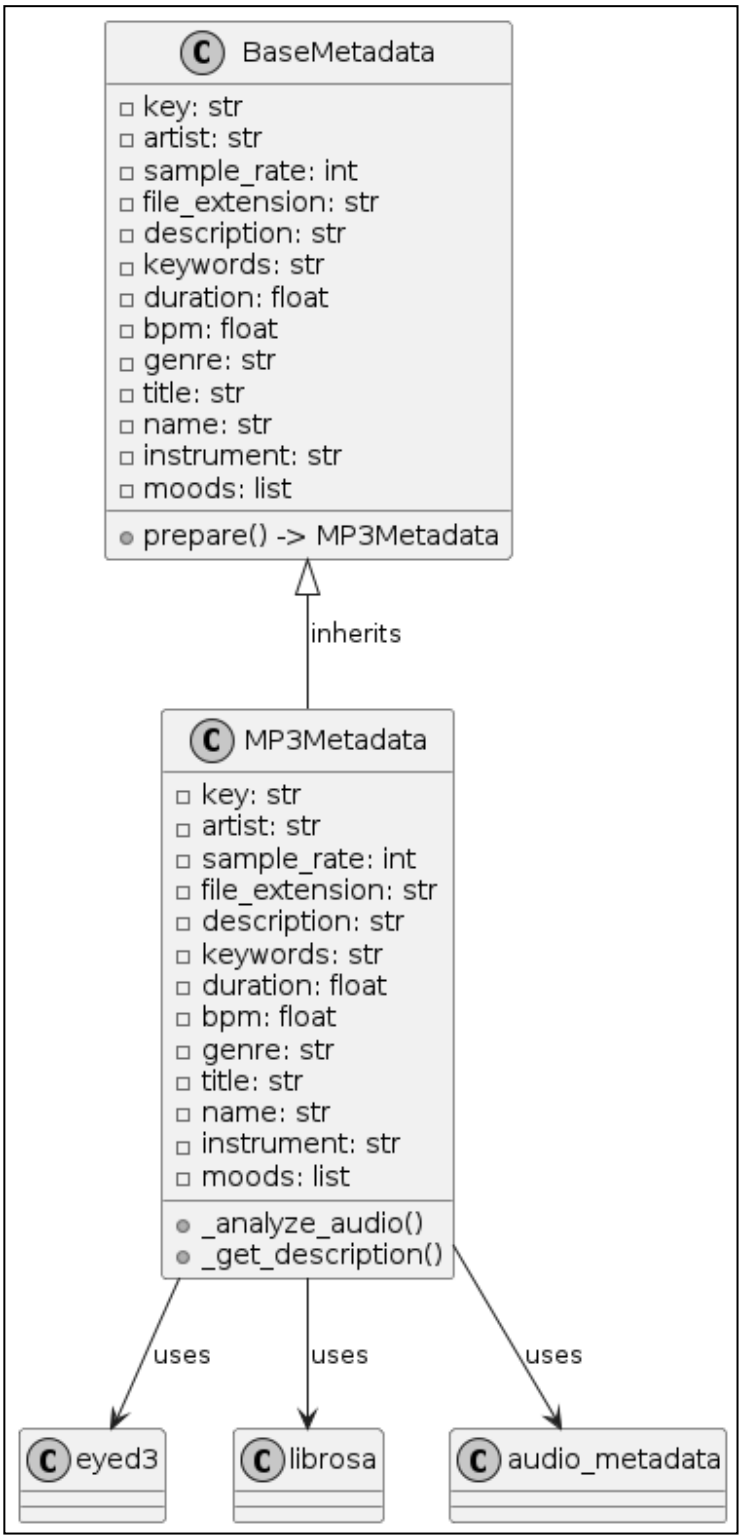


Рисунок 1 – Компонент «MP3Metadata»

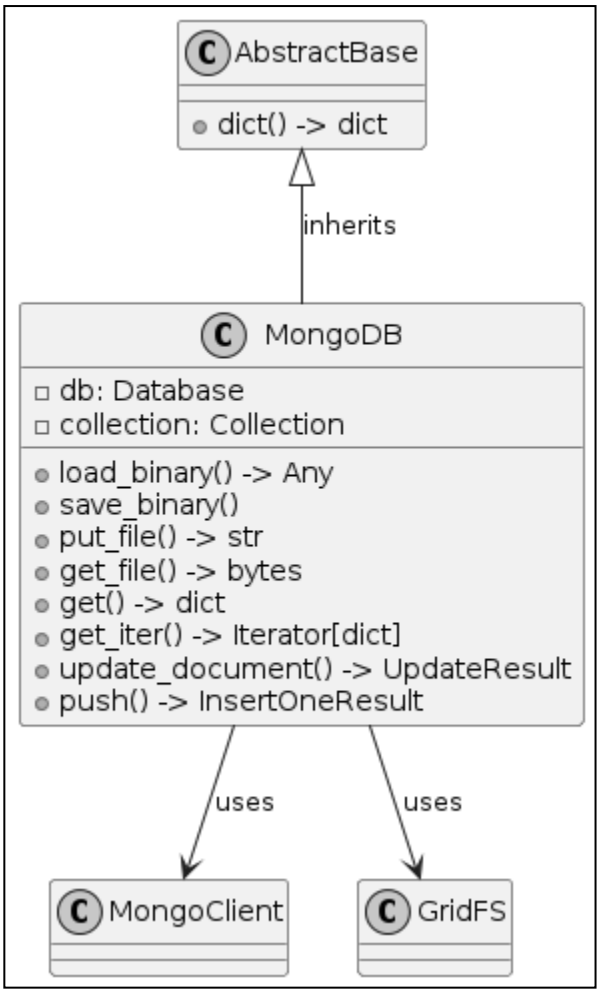


Рисунок 2 – Компонент «MongoDB»

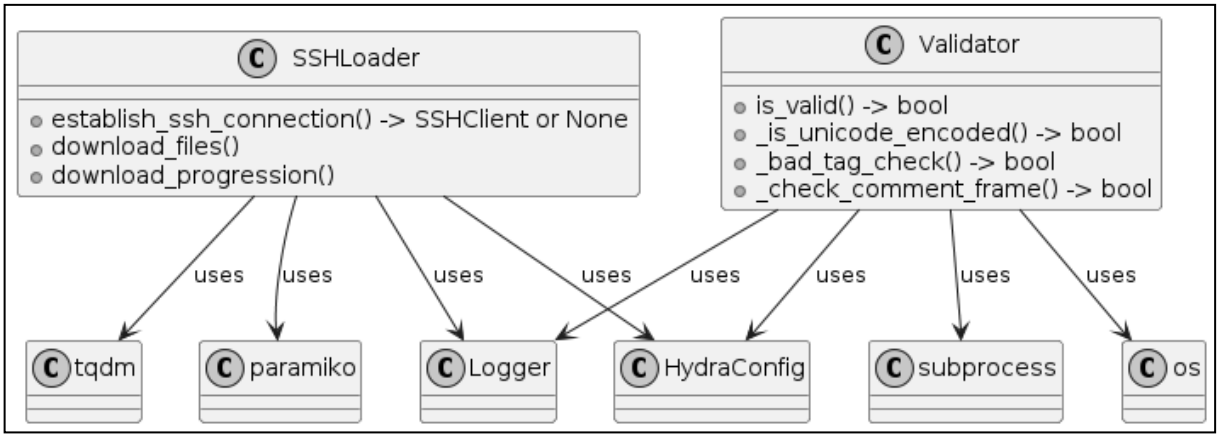


Рисунок 3 – Компоненты «Validator» и «SShLoader»

Приложение Б. Дополнительные фрагменты кода

Фрагменты кода, на которые имеются ссылки в основном тексте работы, представлены на рисунках 4–9.

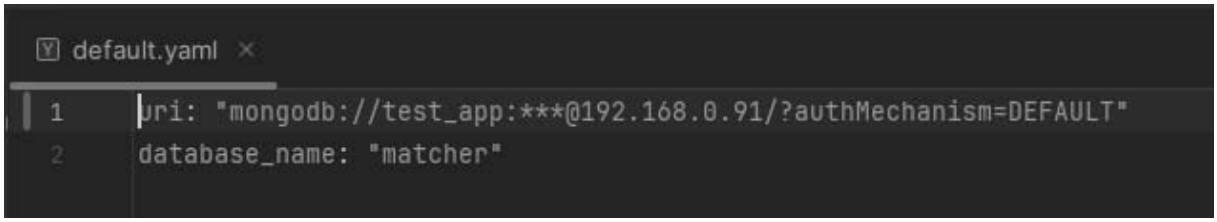


Рисунок 4 – Конфигурация «default.yaml» компонента «MongoDB»

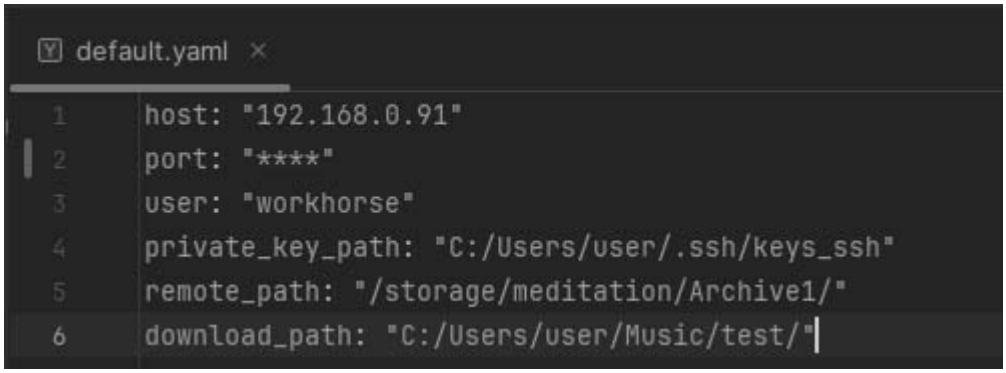


Рисунок 5 – Конфигурация «default.yaml» компонента «SShLoader»

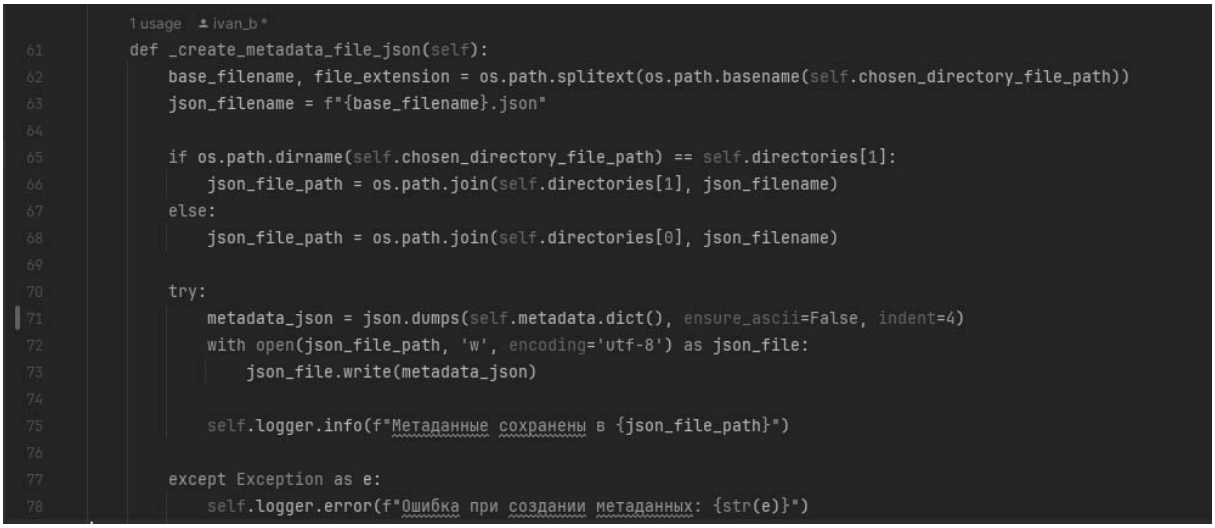


Рисунок 6 – Метод `_create_metadata_file_json()` компонента `DataSetGenerator`

```
    2 usages  ivan_b
54      def _download_progression(self, sftp, remote_file, pbar):
55          remote_file_path = os.path.join(self.cfg.remote_path, remote_file)
56          local_file_path = os.path.join(self.cfg.download_path, remote_file)
57          sftp.get(remote_file_path, local_file_path)
58          pbar.update(1)
59          return [local_file_path]
60
```

Рисунок 7 – Метод «_download_progression()» компонента «SShLoader»

```
    1 usage  ivan_b *
31  def is_unicode_encoded(filename):
32      try:
33          filename.decode('utf-8')
34          filename.encode('utf-8')
35          return True
36      except UnicodeDecodeError:
37          logger.error(f"Найден файл с недопустимой кодировкой: {filepath}")
38          return False
39      except UnicodeEncodeError:
40          logger.error(f"Найден файл с недопустимой кодировкой: {filepath}")
41          return False
```

Рисунок 8 – Функция «is_unicode_encoded()»

```
    3 usages  ivan_b
75  def move_to_removed_folder(mp3_filepath, removed_folder):
76      shutil.move(mp3_filepath, removed_folder)
77
```

Рисунок 9 – Функция «move_to_removed_folder()»

Приложение В. Полный листинг архитектуры «EnCodec»

Полный листинг архитектуры «EnCodec» представлен в листинге 1.

Листинг 1 – Архитектура «EnCodec»

```
EncodecModel(
  (encoder): SEANetEncoder(
    (model): Sequential(
      (0): StreamableConv1d(
        (conv): NormConv1d(
          (conv): Conv1d(1, 64, kernel_size=(7,), stride=(1,))
          (norm): Identity()
        )
      )
      (1): SEANetResnetBlock(
        (block): Sequential(
          (0): ELU(alpha=1.0)
          (1): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(64, 32, kernel_size=(3,), stride=(1,))
              (norm): Identity()
            )
          )
          (2): ELU(alpha=1.0)
          (3): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(32, 64, kernel_size=(1,), stride=(1,))
              (norm): Identity()
            )
          )
        )
        (shortcut): Identity()
      )
      (2): ELU(alpha=1.0)
      (3): StreamableConv1d(
        (conv): NormConv1d(
          (conv): Conv1d(64, 128, kernel_size=(8,), stride=(4,))
          (norm): Identity()
        )
      )
      (4): SEANetResnetBlock(
        (block): Sequential(
          (0): ELU(alpha=1.0)
          (1): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(128, 64, kernel_size=(3,), stride=(1,))
              (norm): Identity()
            )
          )
          (2): ELU(alpha=1.0)
          (3): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
              (norm): Identity()
            )
          )
        )
        (shortcut): Identity()
      )
      (5): ELU(alpha=1.0)
      (6): StreamableConv1d(
        (conv): NormConv1d(
```

```
    (conv): Conv1d(128, 256, kernel_size=(8,), stride=(4,))
    (norm): Identity()
  )
)
(7): SEANetResnetBlock(
  (block): Sequential(
    (0): ELU(alpha=1.0)
    (1): StreamableConv1d(
      (conv): NormConv1d(
        (conv): Conv1d(256, 128, kernel_size=(3,), stride=(1,))
        (norm): Identity()
      )
    )
    (2): ELU(alpha=1.0)
    (3): StreamableConv1d(
      (conv): NormConv1d(
        (conv): Conv1d(128, 256, kernel_size=(1,), stride=(1,))
        (norm): Identity()
      )
    )
  )
  (shortcut): Identity()
)
(8): ELU(alpha=1.0)
(9): StreamableConv1d(
  (conv): NormConv1d(
    (conv): Conv1d(256, 512, kernel_size=(10,), stride=(5,))
    (norm): Identity()
  )
)
(10): SEANetResnetBlock(
  (block): Sequential(
    (0): ELU(alpha=1.0)
    (1): StreamableConv1d(
      (conv): NormConv1d(
        (conv): Conv1d(512, 256, kernel_size=(3,), stride=(1,))
        (norm): Identity()
      )
    )
    (2): ELU(alpha=1.0)
    (3): StreamableConv1d(
      (conv): NormConv1d(
        (conv): Conv1d(256, 512, kernel_size=(1,), stride=(1,))
        (norm): Identity()
      )
    )
  )
  (shortcut): Identity()
)
(11): ELU(alpha=1.0)
(12): StreamableConv1d(
  (conv): NormConv1d(
    (conv): Conv1d(512, 1024, kernel_size=(16,), stride=(8,))
    (norm): Identity()
  )
)
(13): StreamableLSTM(
  (lstm): LSTM(1024, 1024, num_layers=2)
)
(14): ELU(alpha=1.0)
(15): StreamableConv1d(
```

```
        (conv): NormConv1d(
          (conv): Conv1d(1024, 128, kernel_size=(7,), stride=(1,))
          (norm): Identity()
        )
      )
    )
  )
  (decoder): SEANetDecoder(
    (model): Sequential(
      (0): StreamableConv1d(
        (conv): NormConv1d(
          (conv): Conv1d(128, 1024, kernel_size=(7,), stride=(1,))
          (norm): Identity()
        )
      )
      (1): StreamableLSTM(
        (lstm): LSTM(1024, 1024, num_layers=2)
      )
      (2): ELU(alpha=1.0)
      (3): StreamableConvTranspose1d(
        (convtr): NormConvTranspose1d(
          (convtr): ConvTranspose1d(1024, 512, kernel_size=(16,),
stride=(8,))
          (norm): Identity()
        )
      )
      (4): SEANetResnetBlock(
        (block): Sequential(
          (0): ELU(alpha=1.0)
          (1): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(512, 256, kernel_size=(3,), stride=(1,))
              (norm): Identity()
            )
          )
          (2): ELU(alpha=1.0)
          (3): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(256, 512, kernel_size=(1,), stride=(1,))
              (norm): Identity()
            )
          )
        )
        (shortcut): Identity()
      )
      (5): ELU(alpha=1.0)
      (6): StreamableConvTranspose1d(
        (convtr): NormConvTranspose1d(
          (convtr): ConvTranspose1d(512, 256, kernel_size=(10,),
stride=(5,))
          (norm): Identity()
        )
      )
      (7): SEANetResnetBlock(
        (block): Sequential(
          (0): ELU(alpha=1.0)
          (1): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(256, 128, kernel_size=(3,), stride=(1,))
              (norm): Identity()
            )
```

```
          )
          (2): ELU(alpha=1.0)
          (3): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(128, 256, kernel_size=(1,), stride=(1,))
              (norm): Identity()
            )
          )
        )
        (shortcut): Identity()
      )
      (8): ELU(alpha=1.0)
      (9): StreamableConvTranspose1d(
        (convtr): NormConvTranspose1d(
          (convtr): ConvTranspose1d(256, 128, kernel_size=(8,),
stride=(4,))
          (norm): Identity()
        )
      )
      (10): SEANetResnetBlock(
        (block): Sequential(
          (0): ELU(alpha=1.0)
          (1): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(128, 64, kernel_size=(3,), stride=(1,))
              (norm): Identity()
            )
          )
          (2): ELU(alpha=1.0)
          (3): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(64, 128, kernel_size=(1,), stride=(1,))
              (norm): Identity()
            )
          )
        )
        (shortcut): Identity()
      )
      (11): ELU(alpha=1.0)
      (12): StreamableConvTranspose1d(
        (convtr): NormConvTranspose1d(
          (convtr): ConvTranspose1d(128, 64, kernel_size=(8,), stride=(4,))
          (norm): Identity()
        )
      )
      (13): SEANetResnetBlock(
        (block): Sequential(
          (0): ELU(alpha=1.0)
          (1): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(64, 32, kernel_size=(3,), stride=(1,))
              (norm): Identity()
            )
          )
          (2): ELU(alpha=1.0)
          (3): StreamableConv1d(
            (conv): NormConv1d(
              (conv): Conv1d(32, 64, kernel_size=(1,), stride=(1,))
              (norm): Identity()
            )
          )
```

```
        )
        (shortcut): Identity()
      )
      (14): ELU(alpha=1.0)
      (15): StreamableConv1d(
        (conv): NormConv1d(
          (conv): Conv1d(64, 1, kernel_size=(7,), stride=(1,))
          (norm): Identity()
        )
      )
    )
  )
  (quantizer): ResidualVectorQuantizer(
    (vq): ResidualVectorQuantization(
      (layers): ModuleList(
        (0-3): 4 x VectorQuantization(
          (project_in): Identity()
          (project_out): Identity()
          (_codebook): EuclideanCodebook()
        )
      )
    )
  )
)
```