Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Метод автоматического определения ритмического рисунка и темпа цифровой музыкальной записи на основе байесовского иерархического моделирования»

Студент ИУ7-86Б _____ А. А. Петрова
(Группа) (Подпись, дата) (И.О. Фамилия)

Руководитель _____ К. А. Кивва
(Подпись, дата) (И.О. Фамилия)

2023 г.

РЕФЕРАТ

Расчетно-пояснительная записка 65 с., 29 рис., 12 табл., 21 ист., 3 прил.

В работе представлена разработка метода автоматического определения ритмического рисунка и темпа цифровой музыкальной записи на основе байесовского иерархического моделирования.

Рассмотрена задача определения темпа и ритма музыки. Рассмотрен метод ее решения на основе байесовского иерархического моделирования, его применимость при работе с музыкой разных жанров и разным составом инструментов. Представлена реализация метод оценки переменных темпа и ритма музыки на основе байесовского иерархического моделирования.

КЛЮЧЕВЫЕ СЛОВА

темп музыки, ритм музыки, bpm, вейвлет, марковская модель, байесовская модель, нейросети.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Автоматическая транскрипция музыки (АТМ) — это процесс преобразования акустического музыкального сигнала в ту или иную форму нотной записи [1]. АТМ чаще всего используется музыкантами для получения нот или табулатур по аудиофайлу с целью дальнейшего изучения этой композиции. Данную задачу можно разделить на несколько подзадач, к которым в том числе относятся задачи выделения информации о ритме и темпе музыки. Несмотря на то, что задачу АТМ для монофонических сигналов можно считать решенной [1], проблема создания автоматизированной системы, способной транскрибировать полифоническую (многоголосую) музыку без ограничений по степени полифонии или типу инструмента, остается открытой.

Цель данной работы – реализовать метод автоматического определения темпа и ритма музыки на основе байесовского иерархического моделирования.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

— провести анализ предметной области и сформулировать проблему;
— проанализировать и сравнить основные методы определения темпа и ритма;
— разработать метод решения поставленной задачи;
— спроектировать архитектуру разрабатываемого программного обеспечения;
— реализовать разработанный метод;
— протестировать и сравнить результаты работы реализованного метода с результатами, полученными с помощью известных аналогов.

1 Аналитический раздел

1.1 Темп, ритм и метр

Темп – мера времени в музыке, упрощенно – «скорость исполнения музыки» [2].

Существует несколько способов измерения темпа. В классической музыке чаще всего используется словесное описание (как правило, на итальянском). Этот метод является неточным и дает лишь примерное представление о «скорости» исполнения музыкального произведения. Примеры такого описания: адажио, ленто (медленные темпы); анданте, модерато (средние темпы); аллегро, виво (быстрые темпы).

Второй, более точный способ измерения темпа – это число ударов в минуту (beats per minute, сокращенно bpm). Данный метод напрямую связан с частотой колебания маятника в метрономе (устройстве, предназначенном для точного ориентира темпа при исполнении музыки). Стандартным темпом считается 120 bpm, т. е. 2 Гц.

В данной работе будет использоваться второй способ измерения темпа (в bpm).

Ритм – организация музыки во времени [3]. Ритмическую структуру музыки образует последовательность длительностей – звуков и пауз.

Ритм в музыке принадлежит к числу терминов, дискуссии о которых ведутся в науке последние два столетия. Единого мнения по вопросу его определения нет [4]. Чаще всего ритм определяется как регулярная, периодическая последовательность акцентов. Такое понимание ритма фактически идентично метру.

Метр в музыке – это чередование сильных и слабых долей в определенном темпе [2]. Обычно метр фиксируется с помощью тактового размера и тактовой черты (рис. 1.1).

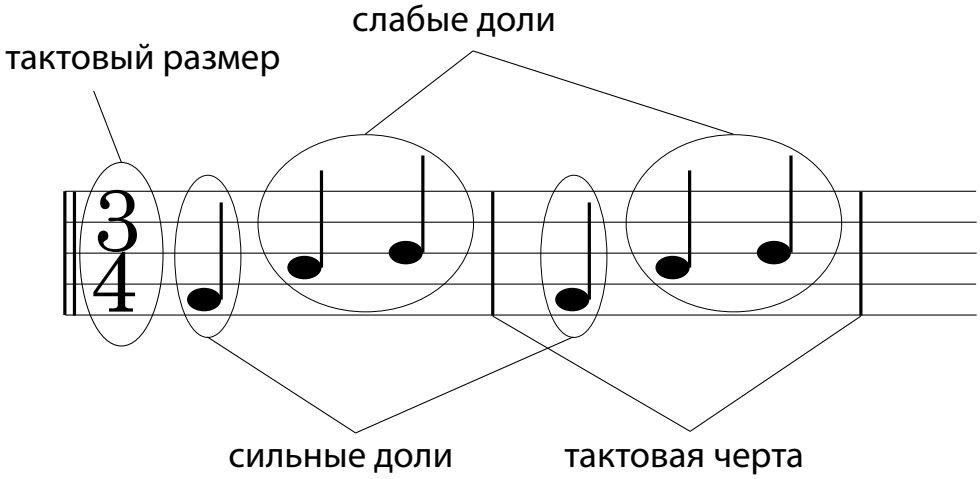

Рисунок 1.1 – Обозначение метра

Размер задает относительную длительность каждой доли. Например, размер «3/4» говорит о том, что в такте 3 доли, каждая из которых представлена четвертной нотой. Можно сказать, что размер – числовое представление метра с указанием длительности каждой доли. Такт в свою очередь – единица метра, начинающаяся с наиболее сильной доли и заканчивающаяся перед следующей равной ей по силе (рис. 1.1).

В данной работе не будут учитываться тонкости различия ритма и метра. Соответственно, для измерения ритма будет использоваться числовое представление метра в виде тактового размера.

1.2 Проблема определения ритма и темпа

Основной проблемой автоматического определения ритма и темпа музыки является наличие некоторых особенностей в музыкальных записях с живыми инструментами, затрудняющих это определение. Одна из таких особенностей – это нечеткое попадание инструмента в ритмическую сетку. Такие небольшие отклонения на живых записях присутствуют всегда [5]. Они не заметны для уха человека, но могут осложнять автоматическое распознавание.

Также в некоторых случаях темп и ритм может изменяться в течение музыкального произведения. Пример переменного темпа приведен на рис. 1.2 (темп

обозначается числами сверху в bpm). На рис. 1.3 приведен пример переменного ритма (размера).

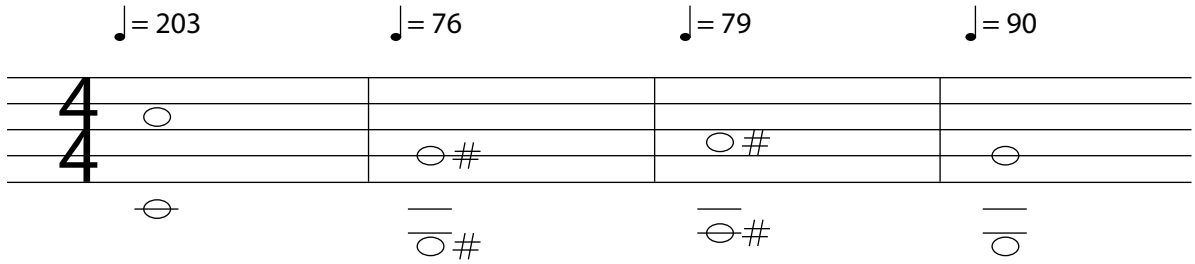

Рисунок 1.2 – Пример переменного темпа (System of a down «Aerials»)

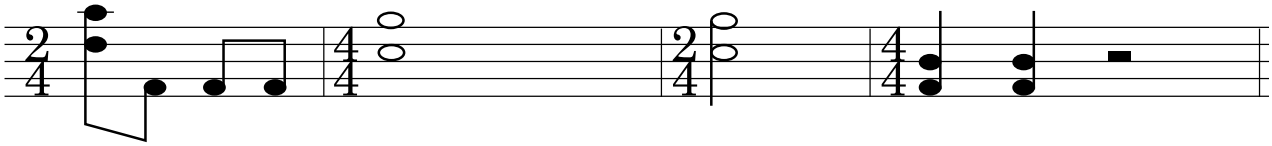

Рисунок 1.3 – Пример переменного размера (Metallica «Master of puppets»)

В качестве критериев сравнения рассматриваемых далее методов выделены следующие:

— точность результатов применения метода;
— возможность определения переменного темпа и ритма;
— ограничения на формат входного аудиофайла;
— размер использовавшегося для обучения датасета (если обучение необходимо).

1.3 Дискретное вейвлет-преобразование

1.3.1 Общие сведения

Так как преобразование Фурье не позволяет получить частотно-временное представление сигнала, оно подходит только для стационарных сигналов (т. е. сигналов, частотное наполнение которых не меняется во времени). Большинство же реальных аудио-сигналов являются нестационарными. Основная же проблема оконного преобразования Фурье (ОПФ) заключается в невозможно-

сти получить произвольно точное частотно-временное представление сигнала, то есть нельзя определить для какого-то момента времени, какие спектральные компоненты присутствуют в сигнале. Эта проблема называется проблемой разрешения.

В качестве альтернативы ОПФ было разработано вейвлет-преобразование.

Основная идея вейвлет-преобразования – это разделение сигнала на высокие и низкие частоты с помощью фильтров [6]. После применения фильтров полученные низкие частоты снова пропускаются через два фильтра и т. д. При этом высокие частоты остаются неизменными. Эта операция называется декомпозицией.

На высоких частотах лучше разрешение по времени, а на низких – по частоте.

Фильтры для высоких и низких частот определяются следующими уравнениями [7]:

$$y_{high}[k] = \sum_{n=-\infty}^{\infty} x[n]g[2k-n], \tag{1}$$

$$y_{low}[k] = \sum_{n=-\infty}^{\infty} x[n]h[2k-n], \tag{2}$$

где $x[n]$ – пропускаемый через фильтр сигнал (последовательность), $h[n]$ и $g[n]$ – импульсные характеристики (отклик на единичный импульс) низкочастотного и высокочастотного фильтров соответственно, k и n – целые числа, соответствующие отсчетам (теорема отсчетов [8]).

Выражение $2k-n$ в формулах 1 и 2 позволяет обрезать сигнал, тем самым увеличив его масштаб в два раза (т. к. половина частот удаляется в результате фильтрации) [6].

Само ДВП (дискретное вейвлет-преобразование) описывается формулой:

$$W(j,k) = \sum_{j}\sum_{k} x(k)2^{-j/2}\psi(2^{-j}n-k), \tag{3}$$

где $\psi(t)$ – функция преобразования, называемая материнским вейвлетом, j и k связаны с параметрами сдвига τ (местоположение окна) и масштаба s (величина, обратная частоте). $s = s_0^j$, $\tau = ks_0^j\tau_0$. В данном случае $s_0 = 2$, $\tau_0 = 1$.

1.3.2 Определение ритма и темпа

Алгоритм определения ритма с помощью ДВП основан на обнаружении наиболее заметных периодов сигнала.

Сигнал сначала раскладывается на несколько частотных полос с помощью ДВП. Для этого сигнал «делится» пополам на высокие и низкие частоты, после чего низкие частоты снова разделяются пополам и т. д. Так продолжается до тех пор, пока не останутся два отсчета. Эта операция необходима, т. к. для высоких частот можно точнее указать их временную позицию, а для низких – их значение частоты [6]. После этого огибающая амплитуды во временной области каждой полосы извлекается отдельно. Это достигается за счет фильтрации нижних частот каждой полосы, применения полноволнового выпрямления и понижения частоты дискретизации [7]. Затем огибающие каждой полосы суммируются и вычисляется автокорреляционная функция. Пики автокорреляционной функции соответствуют различным периодам огибающей сигнала.

Фильтрация нижних частот:

$$y[n] = (1 - \alpha)x[n] - \alpha y[n], \tag{4}$$

где $\alpha = 0,99$.

Полноволновое выпрямление:

$$y[n] = abs(x[n]). \tag{5}$$

Понижение частоты дискретизации:

$$y[n] = x[kn]. \tag{6}$$

Нормализация в каждой полосе (удаление среднего значения) для исключения аномальных данных:

$$y[n] = x[n] - E[x[n]], \tag{7}$$

где $E[x[n]]$ – среднее значение последовательности $x[n]$.

Автокорреляция:

$$y[k] = \frac{1}{N} \sum_{n=0}^{N-1} x[n]x[n+k]. \tag{8}$$

Из результата берутся первые пять пиков автокорреляционной функции, после чего рассчитываются и добавляются в гистограмму соответствующие им периодичности в bpm. Этот процесс повторяется в процессе прохождения по сигналу. Периодичность, соответствующая наиболее заметному пику конечной гистограммы, является предполагаемым темпом аудиофайла в bpm.

Основными недостатками рассмотренного метода определения темпа являются неточные (в некоторых случаях даже ошибочные) результаты на музыке определенных жанров (например, на классической музыке), а также невозможность определить переменный темп.

1.4 Скрытые модели Маркова

1.4.1 Стохастическое моделирование

Как уже было упомянуто выше, практически во всех музыкальных записях имеет место небольшое отклонение нот от ритмической сетки. Рассматриваемый метод рассчитан именно на работу с такими случаями. Также в данном методе подразумевается, что входные данные представлены в формате MIDI (Musical Instrument Digital Interface, стандарт обмена данными между цифровыми музыкальными инструментами). В MIDI файлах указывается информация о высоте ноты, ее длительности и силе нажатия [9].

Исследования показывают, что отклонения нот можно смоделировать с помощью распределения Гаусса относительно их идеальной длительности [10].

Тогда, если i – идеальная длительность ноты («намерение») в момент времени t, то ее исполненная длительность x_t моделируется функцией плотности вероятности $f_i(x_t)$.

Пусть $Q = \{q_1, q_2, ..., q_N\}$ – последовательность «намерений» в соответствующие моменты времени. Тогда наблюдаемая последовательность длительностей $X = \{x_1, x_2, ..., x_N\}$ определяется как:

$$P(X|Q) = \prod_{t=1}^{N} f_{q_t}(x_t). \tag{9}$$

В данном методе используются два типа моделей генерации ритмических рисунков для получения возможных ритмов:

— n-граммная модель (длина ноты предсказывается исходя из предыдущих n-1 нот в вероятностном смысле. Эта модель охватывает любые ритмические рисунки и может выдавать точную вероятность);

— «ритмический словарь» (состоит из всех известных ритмических рисунков за единицу времени. Хорошо представляет известные ритмические рисунки, в то время как неизвестные заменяются аналогичными существующими ритмами).

Обе модели можно представить в виде вероятностных сетей перехода состояний, где каждое состояние связано с предполагаемой длительностью ноты. Вероятность того, что номер состояния изменится в последовательности $Q = \{q_1, q_2, ..., q_N\}$ определяется как $P(Q) = p_{q_0} \prod_{t=1}^{N} a_{q_{t-1}q_t}$, где p_i – вероятность изначального нахождения в состоянии i, а a_{ij} – вероятность перехода из состояния i в состояние j.

Колеблющиеся длительности и возможные последовательности нот могут быть объединены в рамках скрытой модели Маркова как вероятности перехода $A = \{a_{ij}\}$ и наблюдаемые вероятности $B = \{b_i(x_t)\}$ соответственно. В таком случае вероятность наблюдения последовательности длительностей X определяется как:

$$P(X) = P(X|Q)P(Q). \tag{10}$$

1.4.2 Определение ритма

Задача заключается в том, чтобы найти временную последовательность Q номеров состояний, которая дает максимальную апостериорную вероятность $P(Q|X)$ при заданной последовательности наблюдаемых длительностей X [10].

По теореме Байеса:

$$P(Q|X) = \frac{P(X|Q)P(Q)}{P(X)}. \tag{11}$$

Значит, максимизация апостериорной вероятности эквивалентна нахождению $argmax P(X|Q)P(Q)$ среди всех возможных Q.

Оптимальная последовательность состояний находится с помощью алгоритма Витерби для поиска наилучшего пути в вероятностной сети переходов.

Основной недостаток представленного метода заключается в необходимости входных данных быть в формате MIDI. Также к недостаткам можно отнести периодические неточности в результатах. Например, музыкальные фрагменты с разным темпом (к примеру, 116 bpm и 127 bpm) могут быть определены как имеющие одинаковый темп (в данном случае 120 bpm [10]).

1.5 Байесовское иерархическое моделирование

1.5.1 Общие сведения

Байесовская иерархическая модель - это метод статистического анализа, который использует байесовский подход для оценки параметров. Она состоит из нескольких уровней, где каждый уровень описывает определенный аспект данных [11].

Для каждого уровня иерархии определяются соответствующие параметры, которые описывают данные. Для оценки параметров используются априорные распределения, которые описывают предположения об этих параметрах до получения данных. Затем с помощью формулы 11 оцениваются парамет-

ры модели, учитывая как данные ($P(X|Q)$), так и априорные распределения ($P(Q)$). Таким образом, байесовская иерархическая модель позволяет учитывать неопределенность в данных и параметрах модели, а также учитывать различные уровни влияния на данные.

Термин $P(X|Q)$ играет две роли в статистике, в зависимости от контекста [12]. Когда она рассматривается как функция от X для известного Q, то она называется **функцией массы вероятности**. Функция массы вероятности описывает вероятность любого исхода X при заданном значении Q. Но этот термин также можно рассматривать как функцию параметра Q для фиксированного X. В этом случае она называется **функцией правдоподобия** и описывает вероятность определенного значения параметра Q при заданном фиксированном значении X. В теореме Байеса используется апостериорное распределение ($P(Q|X)$) как функция параметра Q. Следовательно, $P(X|Q)$ рассматривается как правдоподобие параметра Q, а не масса вероятности X.

Член $P(X)$ является константой по отношению к Q и служит нормирующим числом, которое гарантирует, что апостериорная вероятность не превышает 1.

Чтобы выполнить байесовский анализ, необходимо выбрать априорное распределение $P(Q)$. Этот выбор отражает мнение о возможных значениях Q до сбора данных.

Когда априорная и апостериорная вероятность относятся к одному и тому же распределению, они называются **сопряженными** [12]. Сопряженные значения удобны, они часто облегчают получение апостериорных распределений. Однако сопряженность вовсе не обязательна для байесовского анализа.

1.5.2 Определение темпа и ритма

В случае с темпом музыки формула 11 принимает вид:

$$P(t|d) = \frac{P(d|t)P(t)}{P(d)}, \tag{12}$$

где d – это собранные данные с известным темпом (датасет), а t – искомый темп.

Аналогично и с ритмом (тактовым размером).

Однако темп музыки имеет также некоторую корреляцию с жанром. Поэтому в таком случае лучше всего применять иерархический вариант рассмотренной выше байесовской модели. Тогда на первом уровне будут располагаться так называемые **гипераприорные** распределения [11] для темпа и для жанра ($P(t)$ и $P(g)$ соответственно). На основе этих распределений вычисляются параметры, располагающиеся на втором уровне и необходимые для задания распределения функции правдоподобия (likelihood_params) (см. рис. 1.4). На третьем же уровне находится функция правдоподобия, которая задается на основе вычисленных ранее параметров.

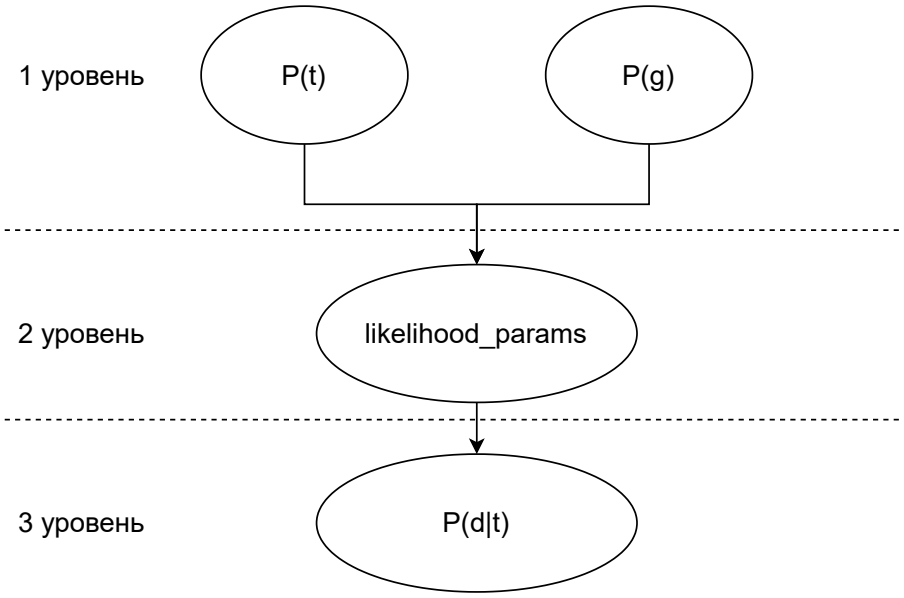

Рисунок 1.4 – Схема иерархической байесовской модели

Таким образом, байесовское иерархическое моделирование позволяет немного увеличить точность определения ритма по сравнению с марковскими моделями (примерно на 2% [13]). Однако главным недостатком байесовского моделирования является возможное неправильное задание априорного распределения. Также к недостаткам можно отнести определение только постоянного темпа и ритма.

1.6 Использование сверточных нейросетей

1.6.1 Представление сигнала

Сигнал представляется в виде спектрограммы по шкале мела, чтобы снизить объем данных, который должен быть обработан нейросетью (мел, от слова «мелодия», - психофизическая (субъективная) единица высоты звука [14]). Шкала мела выбрана вместо линейной шкалы из-за ее связи с человеческим восприятием и диапазонами частот инструментов.

Чтобы создать спектрограмму, сигнал конвертируется в моно, его дискретизация понижается до 11025 Гц, после чего используются полуперекрывающиеся окна из 1024 отсчетов [15]. Это эквивалентно частоте кадров 21,5 Гц, что (согласно теореме отсчетов) достаточно для представления темпа до 646 bpm. Каждое окно преобразуется в 40-полосный спектр в шкале мел, охватывающий диапазон от 20 до 5000 Гц. В качестве длины спектрограммы выбрано 256 кадров, что примерно равняется 11,9 с.

1.6.2 Архитектура сети

Архитектура рассматриваемой сети представлена на рис. 1.5.

Сначала входные данные обрабатываются тремя сверточными слоями, каждый из которых состоит из 16 фильтров размера 1x5. С помощью этих фильтров сопоставляется ритмическая структура сигнала.

После этого идут четыре модуля с несколькими фильтрами. Каждый из модулей состоит из среднего слоя пулинга («avg pooling»), шести параллельных сверточных слоев с фильтрами разных размеров (от 1x32 до 1x256), слоя конкатенации и т. н. «узкого» («bottle-neck») слоя, предназначенного для уменьшения размерности. С помощью этих модулей достигаются две цели:

1) Пулинг по оси частот для суммирования диапазонов мел.
2) Сопоставление сигнала с различными фильтрами, способными обнаруживать длительные временные зависимости.

Чтобы классифицировать свойства, полученные из сверточных слоев, до-

бавляются два полносвязных слоя (по 64 единицы каждый), за которыми следует выходной слой с 256 единицами. Выходной слой использует softmax в качестве функции активации, а все остальные слои используют ELU [16]. Каждому сверточному или полносвязному слою предшествует пакетная нормализация [17]. Первому полносвязному слою также предшествует слой отсева с $p = 0,5$ («dropout») для противодействия переобучению.

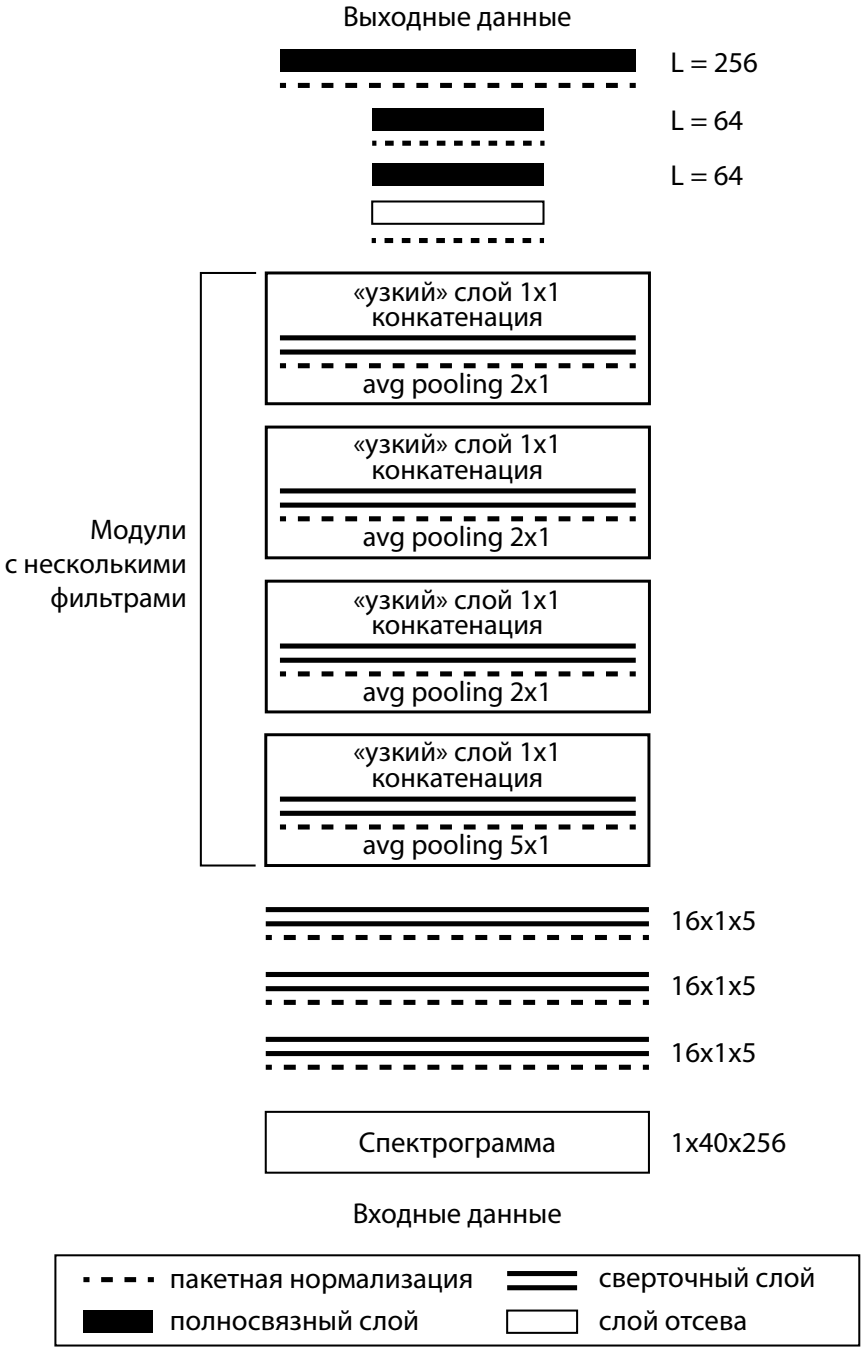

Рисунок 1.5 – Схема архитектуры нейросети

Всего сеть имеет 2921042 обучаемых параметра.

В результате выбирается один из 256 вариантов темпа от 30 до 285 bpm.

Таким образом, сверточные нейросети позволяют определять темп с достаточно высокой точностью (процент правильных оценок с допустимой погрешностью в 4%) (до 92% на основе комбинированной выборки, состоящей из аудиофайлов различных жанров с темпом от 44 до 216 bpm [15]). Также данный метод можно использовать и при определении глобального темпа, не только для фрагментов.

Но он по-прежнему не позволяет определить переменный темп, а также не предназначен для определения ритма. Помимо этого нейросетевые методы имеют такие недостатки, как необходимость обучающих датасетов больших объемов, зависимость от исходных данных и долгое время обучения.

1.7 Сравнение методов

По результатам рассмотрения перечисленных выше методов была составлена таблица 1.1.

Как видно из таблицы, ни один метод в своем изначальном варианте не предполагает определение переменного темпа и ритма. Однако метод скрытых марковских моделей при небольшой модификации может позволить определить переменный темп и ритм [10].

Также стоит заметить, что все методы, кроме ДВП, содержат обучаемые параметры. В скрытых марковских моделях – это множество $\{a_{ij}\}$, а в байесовском иерархическом моделировании – множество $\{\pi_{kk'}\}$. В обоих методах обучение происходит с помощью статистической оценки. Размеры датасетов в таблице были указаны исходя из данных, использовавшихся для обучения соответствующих моделей в исследованиях.

Таблица 1.1 – Сравнение рассмотренных методов

Метод	Точность результатов	Переменный темп и ритм	Формат входного аудиофайла	Размер использовавшегося датасета (кол-во аудиофайлов)
ДВП	Примерно 65% (13 верных из 20) [7]	Не определяются	Нет ограничений	Обучение не требуется
Скрытые марковские модели	Примерно 80% (при допустимой погрешности 4%)	Могут определяться при модификации метода	MIDI	88 [10]
На основе БИМ*	Примерно 82%	Не определяются	Нет ограничений	100 [13]
Сверточная нейросеть	До 92%	Не определяются	Нет ограничений	8596 [15]

*БИМ – байесовское иерархическое моделирование.

Выводы

В этом разделе была проанализирована предметная область и сформулирована проблема. А также была проведена классификация и сравнение основных существующих методов решения поставленной задачи, которое показало, что метод на основе байесовского иерархического моделирования дает достаточно высокую точность результатов, при этом не имея ограничений на формат входного аудиофайла и не требуя много времени на обучение.

Таким образом, необходимо разработать метод на основе байесовского

иерархического моделирования, способный определять переменные темп и ритм музыки. Для этого требуется составить модели, которые будут использовать статистические данные о музыке, такие как темп и тактовый размер. Модель для определения темпа должна также учитывать жанр анализируемой музыки. Составленные модели должны быть обучены на наборе данных, включающих в том числе различные жанры [18]. После обучения моделей, они должны быть протестированы на новых данных, чтобы оценить их точность и эффективность.

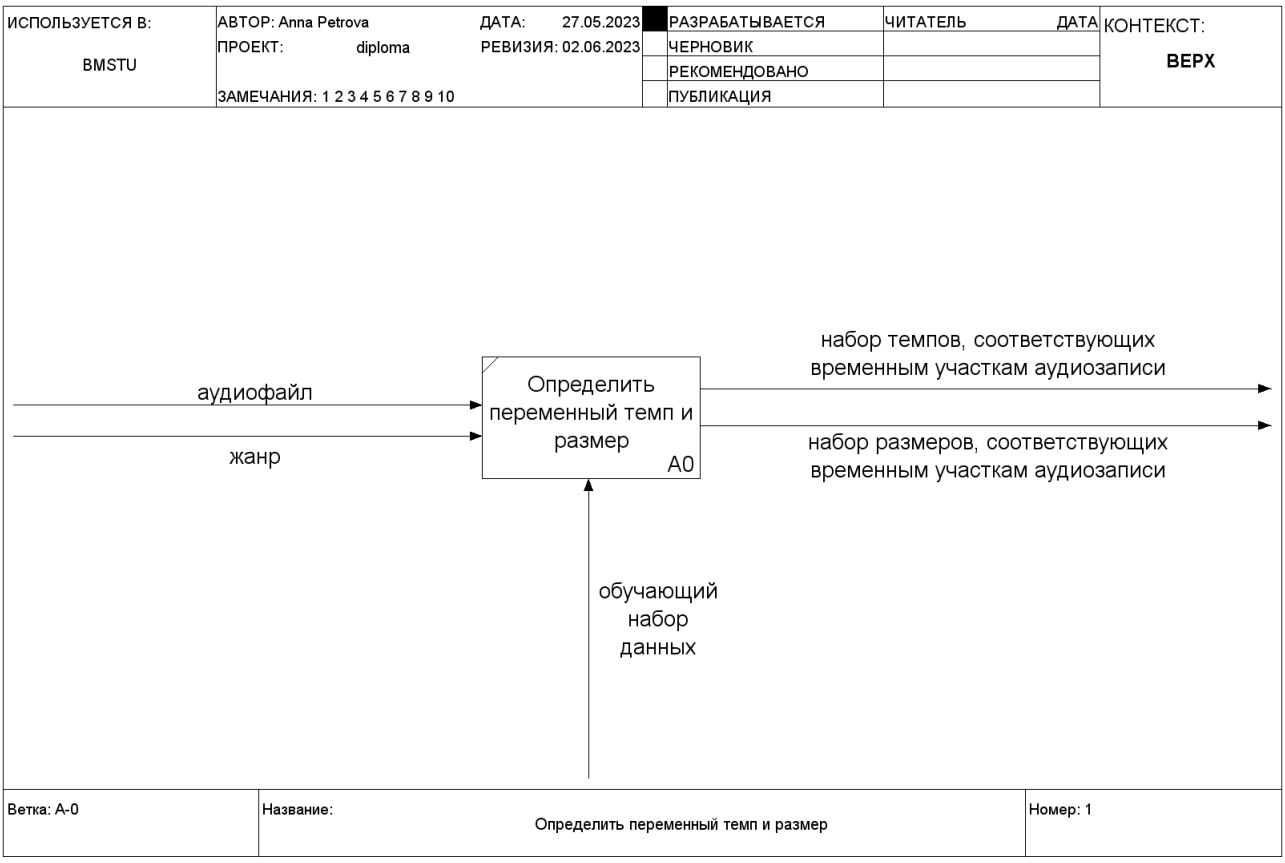

Рисунок 1.6 – IDEF0 нулевого уровня для поставленной задачи

Входные данные:

— аудиофайл;
— жанр музыки (строка).

Выходные данные:

— набор темпов, соответствующих временным участкам аудиозаписи (целые числа, в BPM);

— набор тактовых размеров, соответствующих временным участкам аудиозаписи (обыкновенные дроби в формате «a/b»).

Ограничения:

— загружаемые аудиофайлы должны быть в формате mp3;

— знаменатель тактового размера принимается равным 4.

2 Конструкторский раздел

2.1 Определение переменного темпа

На рисунках 2.7 – 2.11 приведены IDEF0-диаграммы для алгоритма определения переменного темпа музыки.

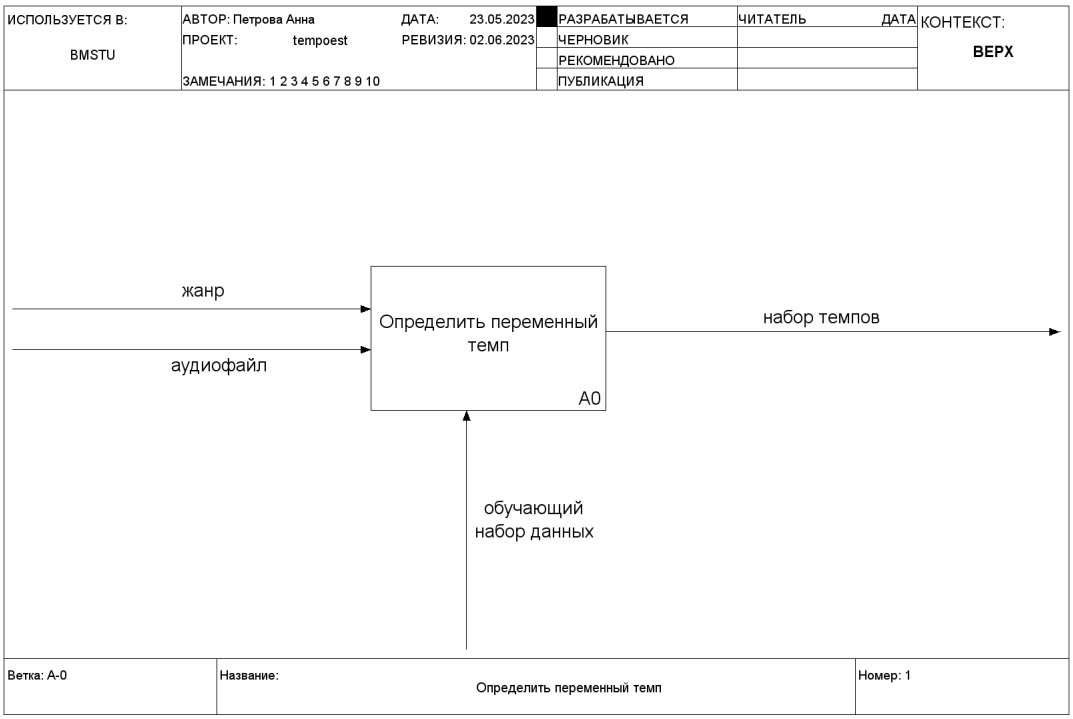

Рисунок 2.7 – IDEF0 нулевого уровня

Для адаптации используемого метода к определению переменного темпа музыки анализируемый аудиофайл разделяется на равные фрагменты. Опытным путем было выявлено, что оптимальной длиной фрагментов является 5 секунд. Если взять более короткие фрагменты, то методу начинает не хватать данных для определения темпа из-за слишком маленькой длины аудио. При этом если разделять на более крупные фрагменты, то увеличивается риск пропуска изменений темпа.

При реализации байесовской иерархической модели в качестве априорного распределения темпа анализируемой музыки было выбрано равномерное распределение с границами от минимального возможного темпа до максимального. Границы темпа выбираются на основе указанного жанра музыки. Такое

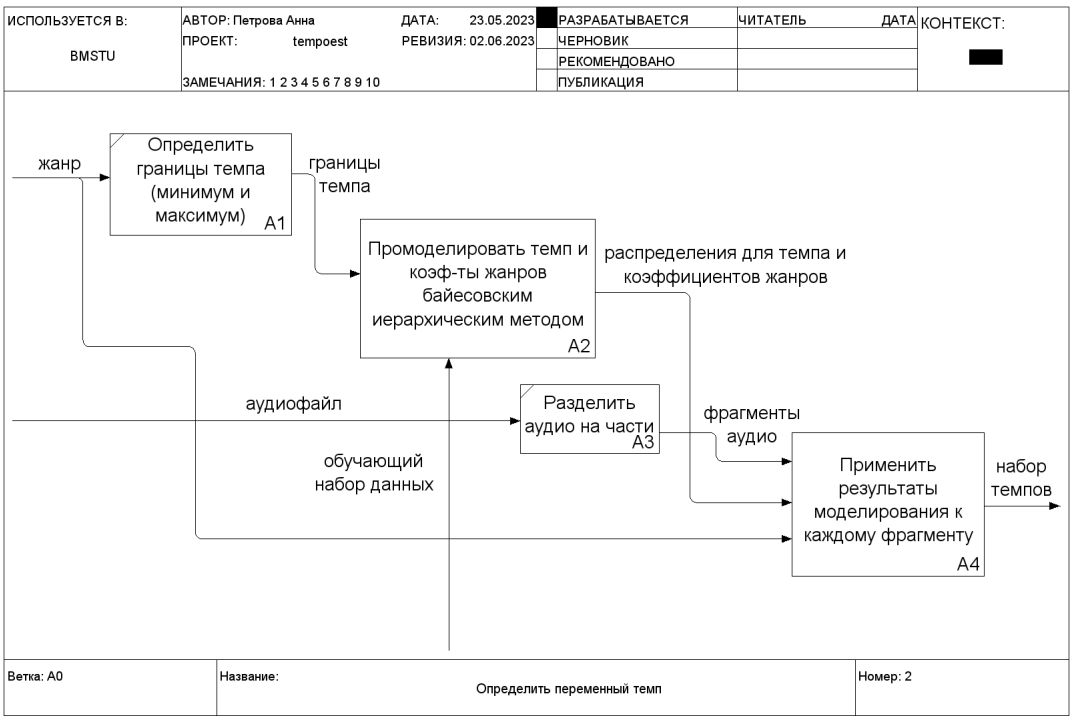

Рисунок 2.8 – Определение переменного темпа

распределение было выбрано, так как на данном этапе, за неимением каких-либо других характеристик указанного аудиофайла, предполагается, что все темпы для него в заданных пределах равновероятны.

Коэффициенты жанров необходимы для корректировки получаемой оценки темпа в зависимости от жанра музыки. Если коэффициент отрицательный, то темп должен быть уменьшен, если положительный – увеличен. В качестве априорного распределения коэффициентов для всех жанров было выбрано нормальное распределение с математическим ожиданием, равным 0, и дисперсией, равной 1. Такой выбор связан с тем, что чаще всего разница между темпами в зависимости от жанра не сильно большая. Это означает, что корректирующие коэффициенты с большей вероятностью находятся в «районе» нуля. При этом 0 является «нейтральным» коэффициентом, т. е. не влияет на полученную оценку темпа.

На основе информации из датасета было выявлено, что распределение темпов различной музыки близко к нормальному (рис. 2.9).

Поэтому в качестве распределения функции правдоподобия темпа было

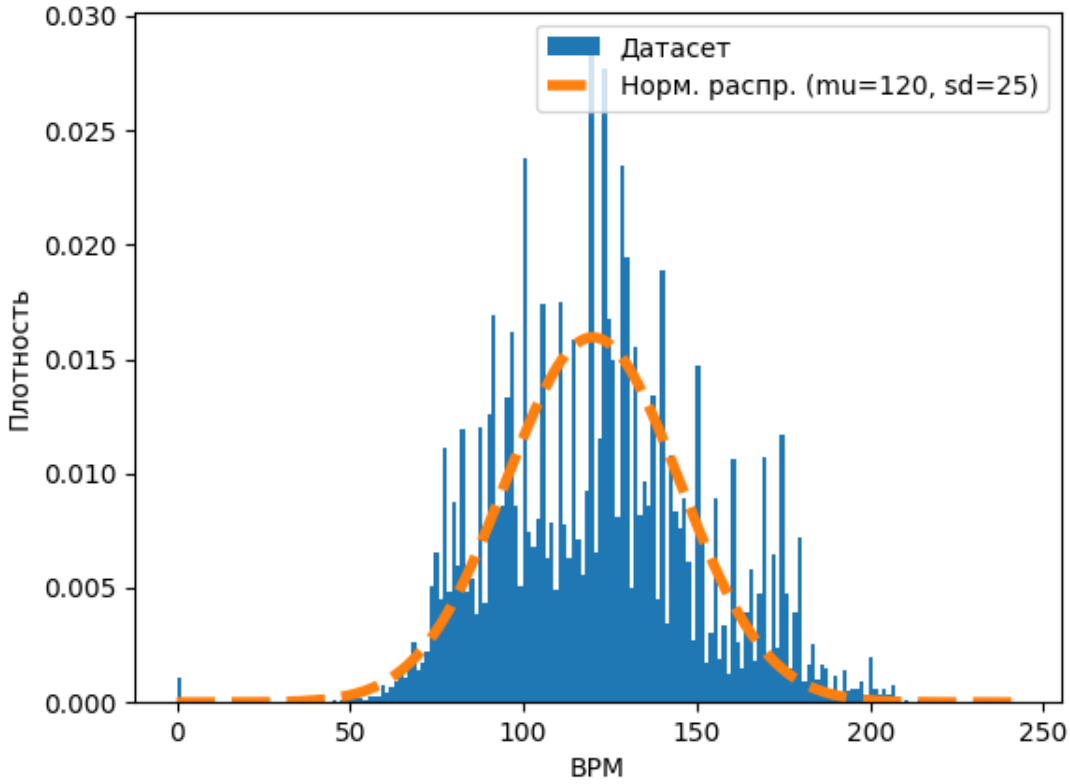

Рисунок 2.9 – Распределение темпов музыки из датасета

выбрано нормальное распределение. Математическое ожидание для каждого жанра для этого распределения рассчитывается на основе мат. ожидания и дисперсии априорного распределения темпа и априорных распределений коэффициентов жанров по формуле:

$$\mu = \mu_{\text{prior}} + \text{coef} * \sigma_{\text{prior}}, \tag{13}$$

где μ_{prior} – мат. ожидание априорного распределения темпа, σ_{prior} – дисперсия априорного распределения темпа, coef – априорный коэффициент текущего жанра.

В качестве дисперсии при этом используется дисперсия априорного распределения темпа.

Важно заметить, что байесовская модель считается до разделения аудиофайла на фрагменты. В результате получаются апостериорные распределения

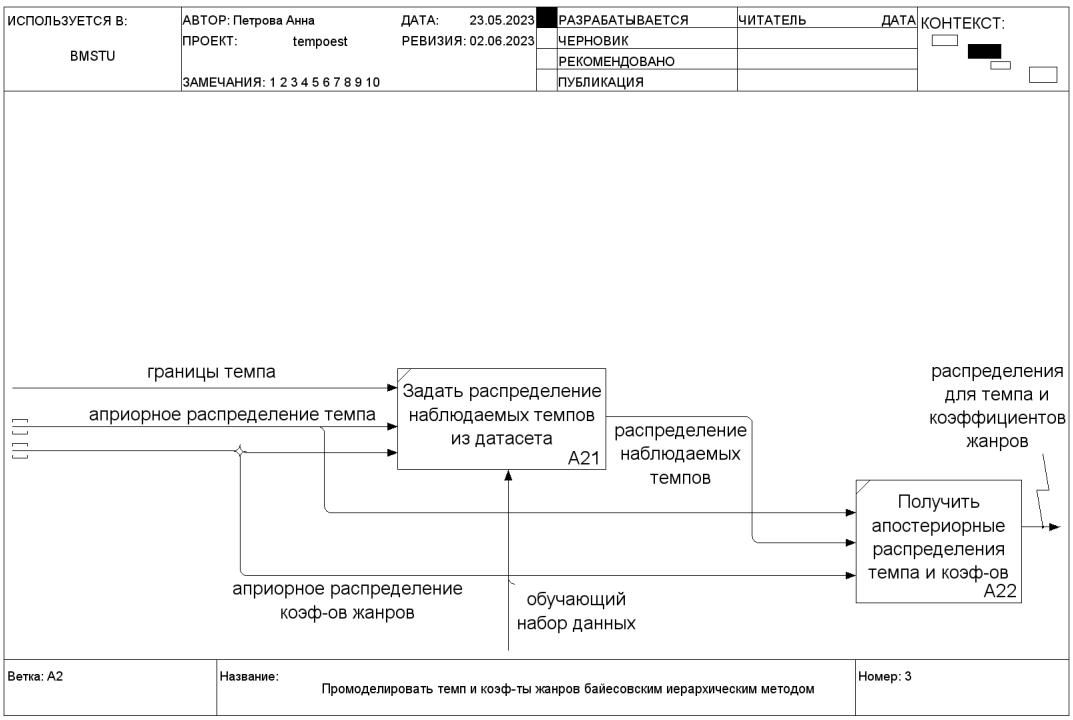

Рисунок 2.10 – Байесовское моделирование

темпа и коэффициентов жанров.

После моделирования для каждого фрагмента аудиофайла рассчитывается примерный диапазон возможных темпов. Для этого сначала определяются так называемые «силы нажатия» на ноты, т. е. амплитуды сигнала во временной области, после чего определяется приблизительное значение темпа на основе корреляции этих амплитуд. Далее относительно найденного темпа делается разброс значений с определенной дельтой. Этот диапазон «накладывается» на полученное байесовским моделированием распределение темпа и получается наиболее вероятный темп (максимум плотности распределения) в данном диапазоне (tempo) (рис. 2.12).

После этого среди апостериорных распределений коэффициентов жанров находится распределение для указанного жанра. В этом распределении ищется наиболее вероятный коэффициент (coef). После чего применяется формула:

$$\text{result} = \text{tempo} + \text{coef} * \sigma_{\text{prior}}. \tag{14}$$

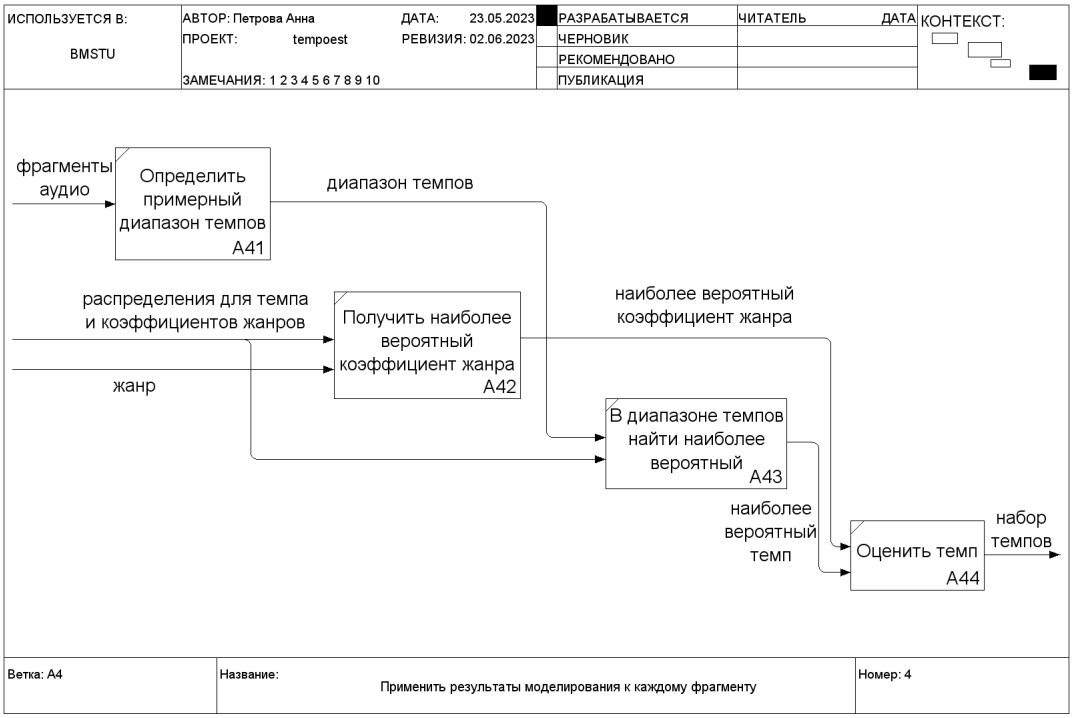

Рисунок 2.11 – Применение результатов к фрагментам аудио

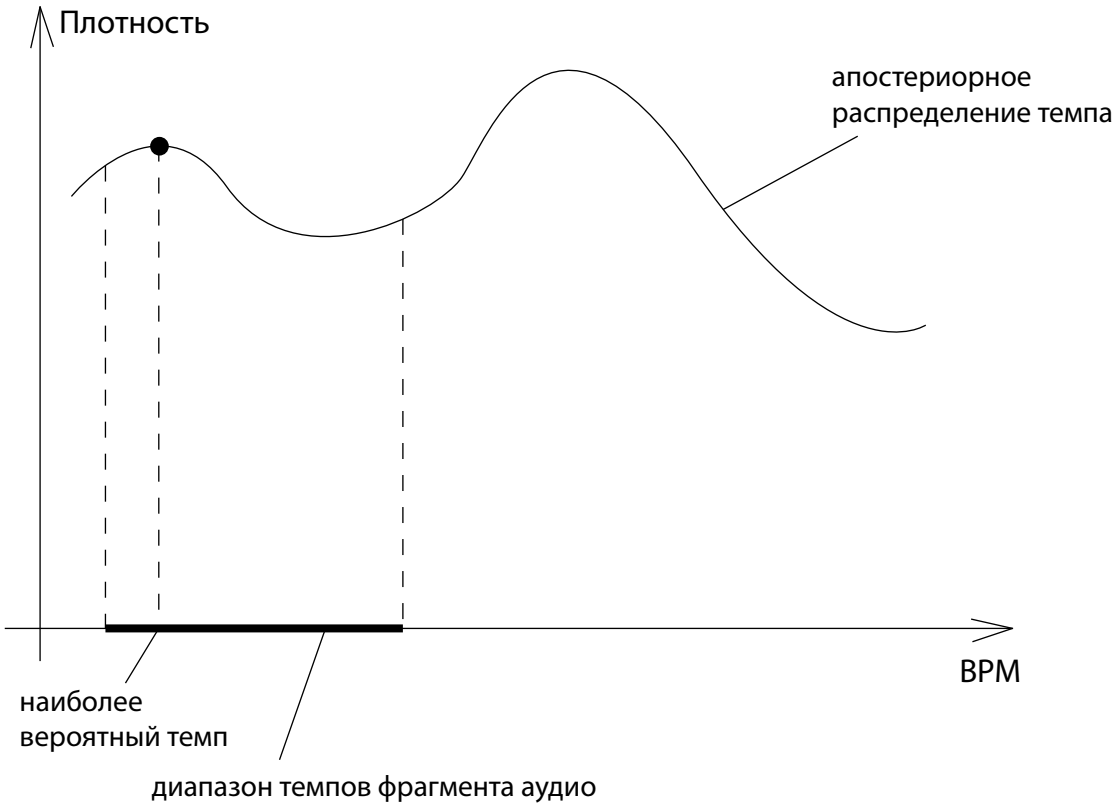

Рисунок 2.12 – Нахождение наиболее вероятного темпа

2.2 Определение переменного тактового размера

На рисунках 2.13 – 2.16 приведены IDEF0-диаграммы для алгоритма определения переменного ритма (тактового размера) музыки.

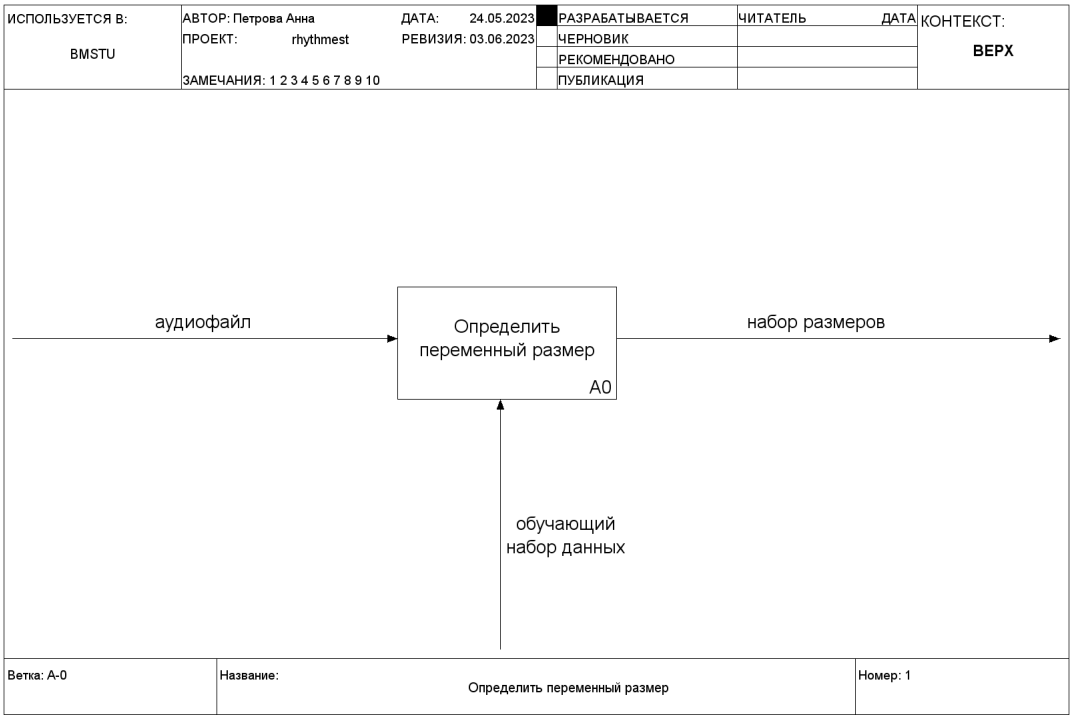

Рисунок 2.13 – IDEF0 нулевого уровня

По аналогии с определением темпа в качестве априорного распределения тактового размера в байесовской модели выбрано равномерное распределение, так как на данном этапе, за неимением каких-либо других характеристик указанного аудиофайла, предполагается, что все размеры для него в заданных пределах равновероятны.

В качестве распределения функции правдоподобия размера выбрано нормальное распределение (т. к. распределение размеров в датасете также близко к нормальному) с математическим ожиданием, равным математическому ожиданию априорного, и дисперсией, также равной априорной.

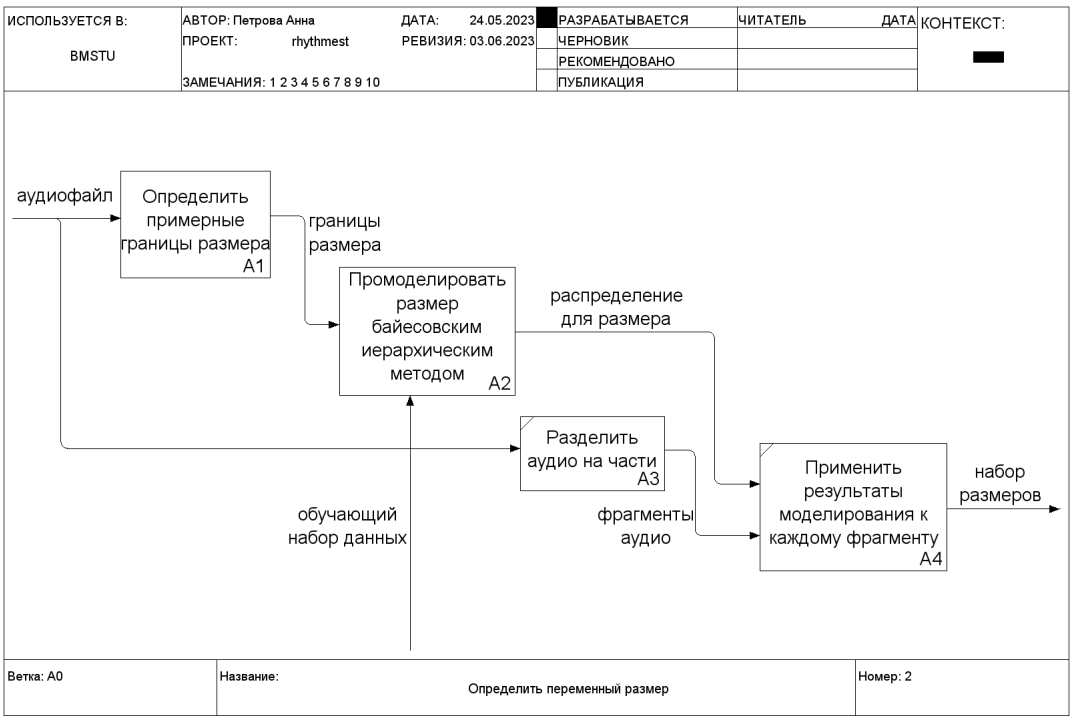

Рисунок 2.14 – Определение переменного ритма

Для задания априорного распределения необходимо определить границы размера. Для этого во всем аудиофайле сначала находятся последовательности амплитуд и «ударов» во временной области. Последовательность «ударов» определяется на основе оценки темпа. В найденной последовательности амплитуд находятся пики (точки максимума). После чего пики амплитуд «накладываются» на последовательность «ударов» и получается последовательность сильных «ударов» (долей) (т. е. предположительные начала тактов). Количество «ударов» между сильными долями и есть тактовый размер. Таким образом определяются примерные границы размеров рассматриваемого аудиофайла.

Остальное происходит аналогично определению темпа. Считается байесовская модель для оценки размера. Далее аудиофайл разделяется на фрагменты по 5 секунд, для каждого фрагмента рассчитывается диапазон размеров по принципу, описанному выше. После чего полученное в результате моделирования апостериорное распределение размеров применяется к данному диапазону, и ищется максимум функции плотности, т. е. наиболее вероятный тактовый размер фрагмента.

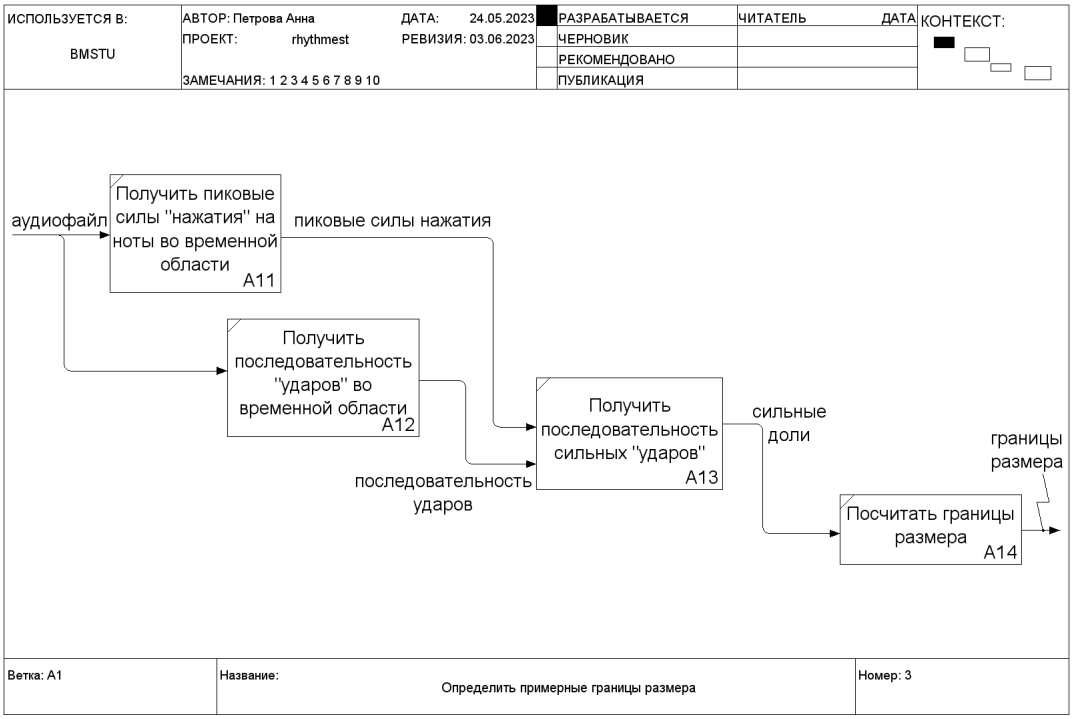

Рисунок 2.15 – Определение границ размера

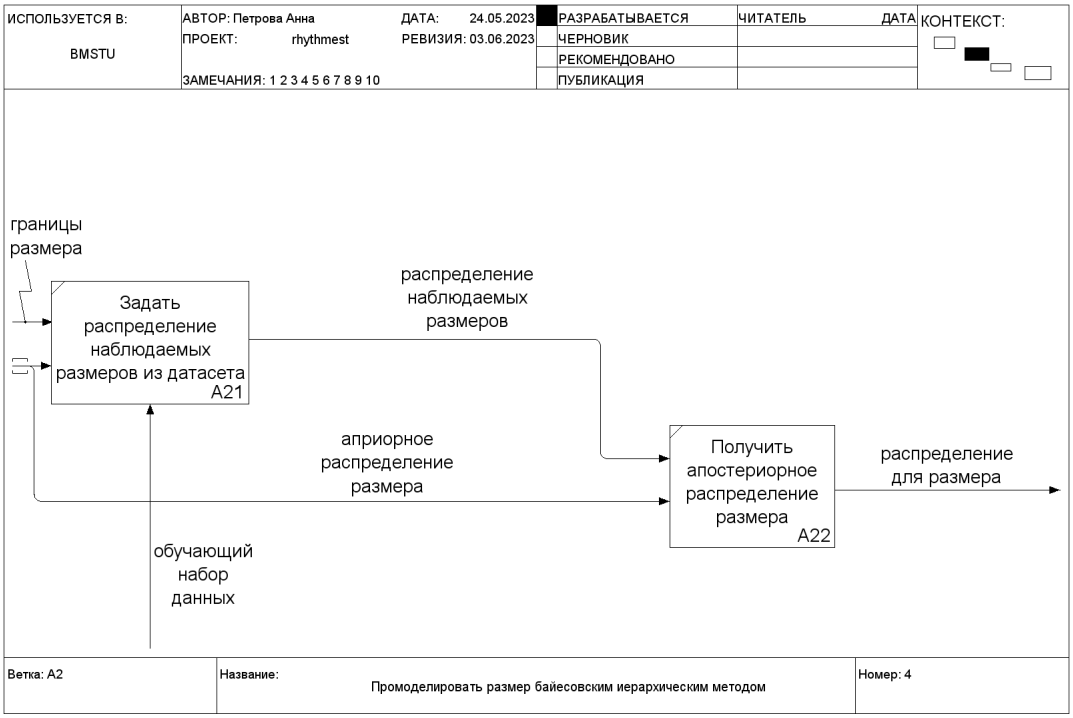

Рисунок 2.16 – Байесовское моделирование

2.3 Структуры данных

Результаты работы программы будут представлены в виде словарей, где в качестве ключей – время от начала аудиофайла в секундах, а в качестве значений – темпы или тактовые размеры соответственно.

Все распределения, значения из датасета, последовательности «ударов» и пики амплитуды будут представлены в виде списков. Списки «ударов» и пиков амплитуды представляют собой последовательность секунд от начала аудиозаписи, в которые происходят «удары» или пики амплитуды соответственно.

2.4 Структура ПО

На рисунке 2.17 показана структура приложения, из которой видно, как происходит взаимодействие компонентов.

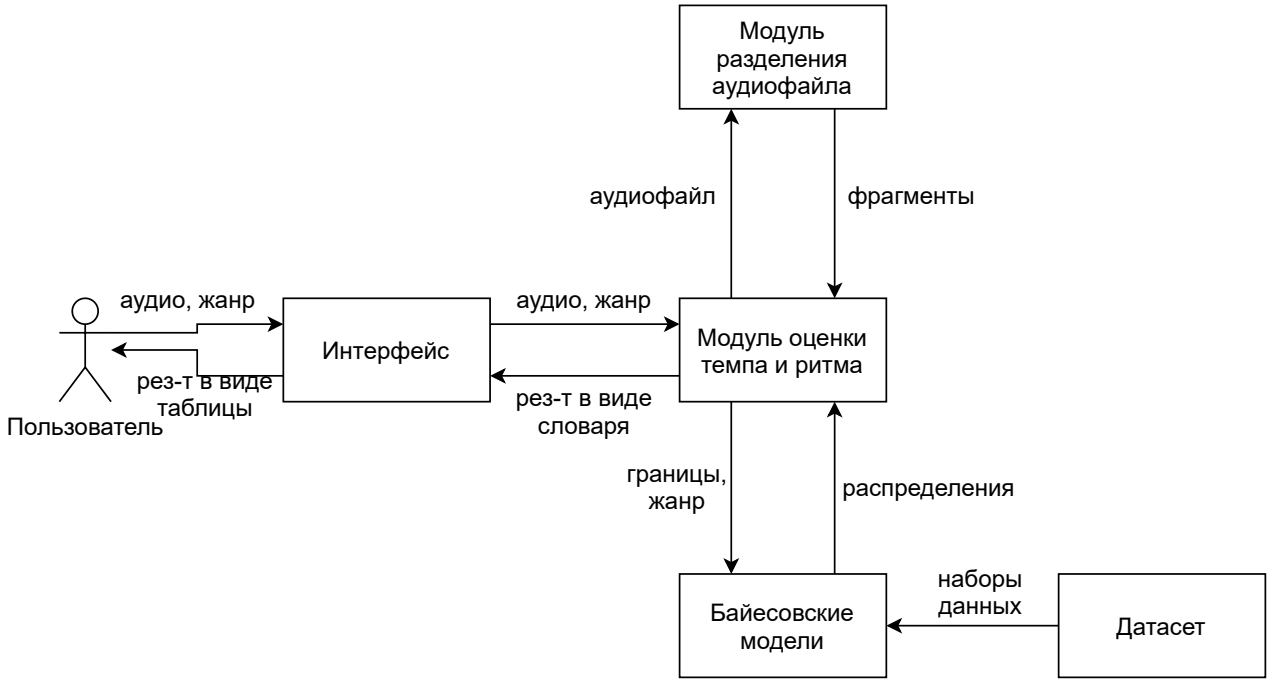

Рисунок 2.17 – Структура приложения

Модуль оценки темпа и ритма применяет результаты байесовского моделирования к фрагментам заданного аудиофайла.

Выводы

На основе теоретических данных, полученных в аналитическом разделе, был разработан метод определения переменного темпа и ритма музыки. Также были приведены IDEF0-диаграммы, схема структуры приложения и описаны основные структуры данных.

3 Технологический раздел

3.1 Выбор средств программной реализации

В качестве языка программирования был выбран Python, так как:

— имеется достаточный опыт программирования на этом языке, что сократит время написания программы;

— для данного языка программирования имеются все необходимые библиотеки для работы с байесовскими моделями и аудиофайлами, что также ускорит процесс разработки.

В качестве среды разработки был выбран PyCharm по следующим причинам:

— он бесплатен для использования студентами;

— он имеет множество встроенных интсрументов, которые облегчают процесс написания и отладки кода;

— имеется достаточный опыт программирования с использованием данной среды разработки, что сократит время изучения возможностей.

Для работы с аудиофайлами использовалась библиотека librosa [19], а для реализации байесовских моделей – библиотека PyMC3 [20]. Кроме того в программе вместо обычных списков часто для удобства использовались numpy массивы [21].

3.2 Детали реализации

3.2.1 Байесовские модели

В листингах 1, 2 представлена реализация байесовских иерархических моделей для оценки темпа и тактового размера соответственно.

Листинг 1: реализация байесовской модели для определения темпа

```
def bpm_model(min_bpm: int, max_bpm: int, bpm_dataset, genre_dataset,
 genres_ints, progress):
  with pm.Model() as model:
    # hyperpriors (lvl 1)
    tempo = pm.Uniform('tempo', lower=min_bpm, upper=max_bpm)
    progress.setValue(20)
    mu = (min_bpm + max_bpm) / 2.0
    sigma = (max_bpm - min_bpm) / 12.0
    genre_coef = pm.Normal('genre_coef', mu=0, sd=1, shape=len(
 genre_dataset.unique()))
    progress.setValue(40)

    # prior (lvl 2)
    bpm_est = mu + genre_coef[genres_ints] * sigma
    progress.setValue(60)

    # likelihood (lvl 3)
    bpm_obs = pm.Normal('bpm_obs', mu=bpm_est, sd=sigma, observed=
 bpm_dataset)
    progress.setValue(80)

    # get the samples
    trace = pm.sample(1000, tune=500, chains=2, cores=1)
    progress.setValue(100)

  return trace
```

Листинг 2: реализация байесовской модели для определения ритма

```
def rhythm_model(measure_min, measure_max, rhythm_dataset, progress):
  with pm.Model() as model:
    # prior
    measure = pm.Uniform('measure', lower=measure_min, upper=measure_max)
    progress.setValue(20)
    mu = (measure_min + measure_max) / 2.0
    progress.setValue(40)
    sigma = (measure_max - measure_min) / 12.0
    progress.setValue(60)
    # likelihood
```

```
1       measure_obs = pm.Normal('measure_obs', mu=mu, sd=sigma, observed=
      rhythm_dataset)
2       progress.setValue(80)
3
4       trace = pm.sample(1000, tune=1000, chains=2)
5       progress.setValue(100)
6
7     return trace
```

На вход моделей поступают границы темпа или размера, массив значений из датасета (для оценки темпа – это темпы и жанры, для ритма – размеры), список жанров, закодированных числами (для темпа) (для размещения коэффициентов по индексам жанров) и линия прогресса из интерфейса для ее обновления.

Выходными данными указанных функций являются распределения соответствующих параметров (в первом случае – это темп и коэффициенты для всех жанров, а во втором – тактовый размер).

Метод sample отвечает за сам процесс моделирования (получения апостериорных распределений).

Расчет диапазона возможных темпов был реализован с помощью метода beat_track() из библиотеки librosa, в качестве дельты взято 40 bpm.

3.2.2 Разделение аудиофайла

В листинге 3 представлена реализация функции разделения аудиофайла на фрагменты.

Листинг 3: разделение аудиофайла на фрагменты

```
1   def split_mp3(audio_path: str, step: int):
2     if not os.path.isdir("tmp"):
3       os.mkdir("tmp")
4     # load audio
5     audio_file = AudioSegment.from_file(audio_path, format="mp3")
6     # fragments length in milliseconds
7     part_length = step
```

```
    # split audio
    parts = [audio_file[i:i + part_length] for i in range(0, len(audio_file),
     part_length)]
    # save fragments
    for i, part in enumerate(parts):
      part.export(f"tmp/part_{i}.mp3", format="mp3")
```

Функция создает временную директорию «tmp/», в которую впоследствии сохраняет фрагменты указанного аудиофайла в формате mp3. После расчета темпов или размеров для всех фрагментов созданная директория очищается и удаляется.

3.2.3 Определение диапазона размеров

В листинге 4 представлена реализация функций для расчета диапазона размеров.

Листинг 4: определение диапазона размеров

```
def calc_measure(downbeats: list) -> int:
    downbeat_inds = [i for i, beat in enumerate(downbeats) if beat == 1]
    if len(downbeat_inds) <= 1:
      return 0
    difs_sum = 0
    for i in range(1, len(downbeat_inds)):
      difs_sum += (downbeat_inds[i] - downbeat_inds[i - 1])
    avg_measure = difs_sum / (len(downbeat_inds) - 1)
    return round(avg_measure)

def get_measure_range(audio_path: str, tail: list):
    y, sr = librosa.load(audio_path)
    onset_env = librosa.onset.onset_strength(y=y, sr=sr)
    # amplitude peaks
    peaks = librosa.util.peak_pick(onset_env, pre_max=3, post_max=3, pre_avg
    =3,
    post_avg=5, delta=0.5, wait=10)
    peaks_time = librosa.frames_to_time(peaks, sr=sr)
    _, beats = librosa.beat.beat_track(onset_envelope=onset_env, sr=sr)
    beats_time = librosa.frames_to_time(beats, sr=sr)
```

```
    # beginnings of bars
    downbeat_times = {}
    for i, beat in enumerate(beats_time):
    if beat in peaks_time:
    downbeat_times[i] = beat
    downbeats = [0 for i in range(len(beats))]
    for beat in downbeat_times.keys():
    downbeats[beat] = 1
    downbeats = tail + downbeats
    prior_measure = calc_measure(downbeats)
    if prior_measure == 0:
      return -1, downbeats
    if prior_measure > 2:
      return 0, np.arange(prior_measure - 2, prior_measure + 3, 1)
    elif prior_measure == 2:
      return 0, np.arange(prior_measure - 1, prior_measure + 3, 1)
    else:
      return 0, np.arange(prior_measure, prior_measure + 3, 1)
```

Функция calc_measure отвечает за расчет тактового размера на основе списка из чередующихся сильных и слабых долей, где сильные доли обозначаются 1, а слабые – 0.

Если в указанном отрывке найдена только одна сильная доля, то этот фрагмент объединяется со следующим. Для этого среди входных данных функции get_measure_range есть параметр tail, который представляет собой список сильных и слабых долей предыдущего фрагмента аудио.

Набор сильных долей формируется на основе совпадений времени в двух списках: пиков амплитуды (peaks_time) и «ударов» (beats_time).

3.3 Пользовательский интерфейс

На рисунке 3.18 представлен интерфейс разработанного приложения.

Приложение поддерживает загрузку только mp3 файлов. Список возможных жанров загружаемой музыки был создан на основе датасета. Темпы и размеры можно определять отдельно, независимо друг от друга.

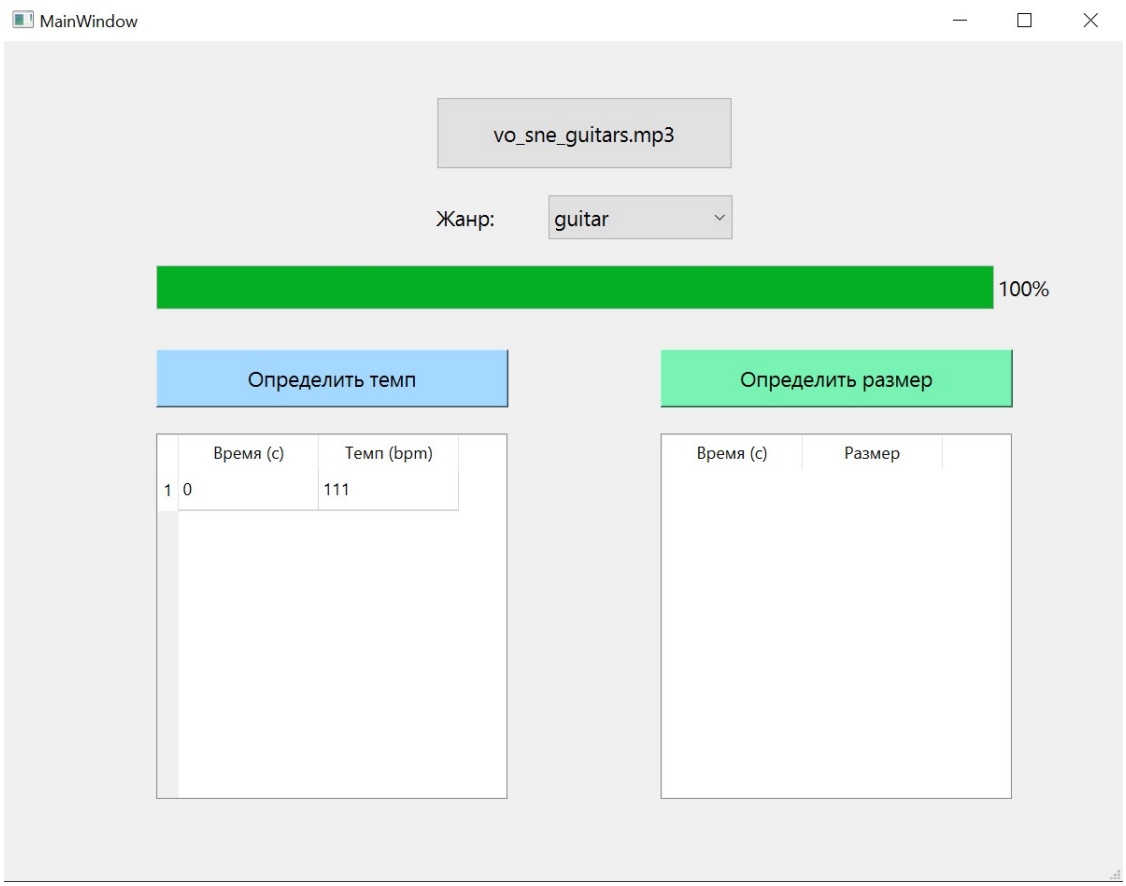

Рисунок 3.18 – Интерфейс приложения

Для того, чтобы определить темп аудиофайла, необходимо:

1) Загрузить аудиофайл, нажав на верхнюю кнопку.
2) Выбрать жанр загруженной музыки из выпадающего списка ниже.
3) Нажать на кнопку «Определить темп».
4) Результаты появятся в левой таблице через 1,5-2 минуты.

Для определения ритма:

1) Загрузить аудиофайл, нажав на верхнюю кнопку.
2) Нажать на кнопку «Определить размер».
3) Результаты появятся в правой таблице в течение минуты.

Жанр в этом случае указывать необязательно.

3.4 Тестирование приложения

В процессе разработки приложения проводились модульные тесты (листинг 5). Тестировались такие функции, как: разделение аудиофайла на фрагменты, очистка директории, удаление директории и расчет тактового размера на основе списка сильных и слабых долей.

Листинг 5: модульные тесты

```
class SplitAudio(unittest.TestCase):
  def test_splitting(self):
    audio = "test_audio/vo_sne.mp3"
    step = 5000
    split_mp3(audio, step)
    self.assertEqual(os.path.exists("tmp/"), True)
    self.assertGreater(len(os.listdir("tmp/")), 0)

class CleanDirs(unittest.TestCase):
  def test_cleaning(self):
    os.mkdir("temp/")
    fp = open('temp/file.txt', 'x')
    fp.close()
    dir = "temp/"
    clean_dir(dir)
    self.assertEqual(len(os.listdir(dir)), 0)
    os.rmdir(dir)
  def test_removing(self):
    os.mkdir("temp/")
    dir = "temp/"
    remove_dir(dir)
    self.assertEqual(os.path.exists("temp/"), False)

class CalcMeasure(unittest.TestCase):
  def test_calculating(self):
    downbeats = [1, 0, 0, 0, 1, 0, 0, 0, 1]
    measure = calc_measure(downbeats)
    self.assertEqual(measure, 4)
```

Как видно из рисунка 3.19, все модульные тесты были успешно пройдены.

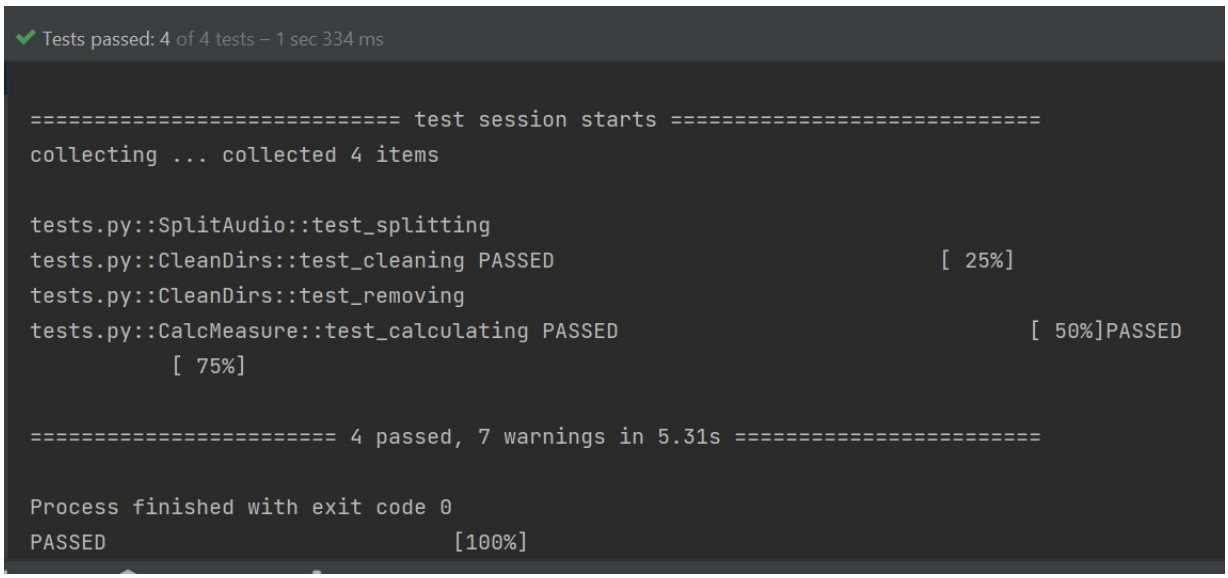

Рисунок 3.19 – Результаты модульного тестирования

Работоспособность программы в целом проверялась вручную. На рисунке 3.20 представлены результаты работы программы на одном из аудиофайлов.

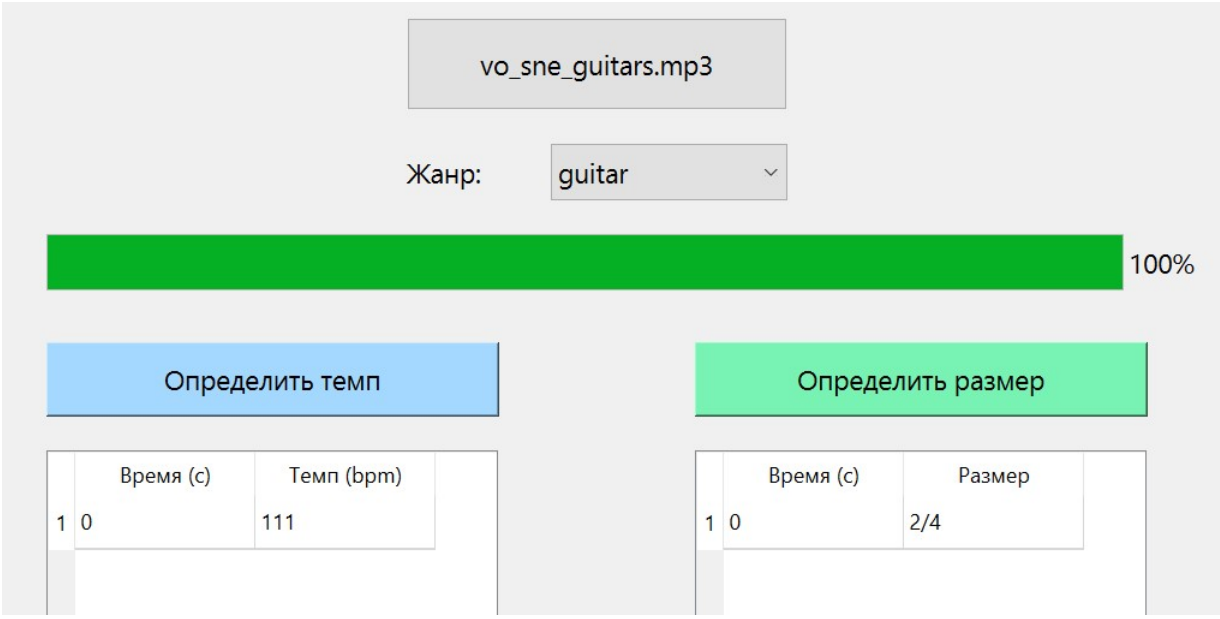

Рисунок 3.20 – Результаты для постоянного темпа и ритма

При этом заранее известно, что темп данной аудиозаписи 125 bpm, а тактовый размер – 4/4 (рис. 3.21, самое левое число – темп в bpm, правее – размер).

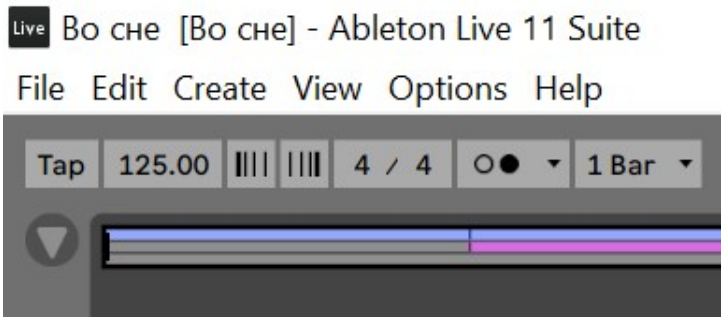

Рисунок 3.21 – Эталонные значения темпа и размера

Таким образом, отклонение полученного темпа от идеального составляет примерно 11%, что является приемлемым результатом, поскольку согласно исследованиям [13] ошибки байесовского метода могут составлять до 18%.

Размер 2/4 также является допустимым результатом в данном случае. Это означает, что сильных долей в аудиозаписи было выявлено в два раза больше, но при этом основной ритм остается тем же, что и при 4/4 (т. к. два такта по 2/4 дают в сумме один такт с размером 4/4).

Для тестирования определения переменного темпа использовался аудиофайл с темпом 115 bpm в первые 25 секунд и 120 bpm в оставшееся время (размер этого музыкального фрагмента постоянен и равен 6/8, что в данном случае может быть приравнено к 3/4).

Результаты работы метода представлены на рисунке 3.22.

Ошибка определения темпа в данном случае составляет примерно 7%, что также укладывается в допустимые пределы.

Для тестирования определения переменного ритма использовался аудиофайл с размером 3/4 в первые 15 секунд и 4/4 bpm в оставшееся время (темп этого музыкального фрагмента постоянен и равен 100 bpm).

Результаты работы метода представлены на рисунке 3.23.

Ошибка определения размера в таком случае составляет примерно 8%.

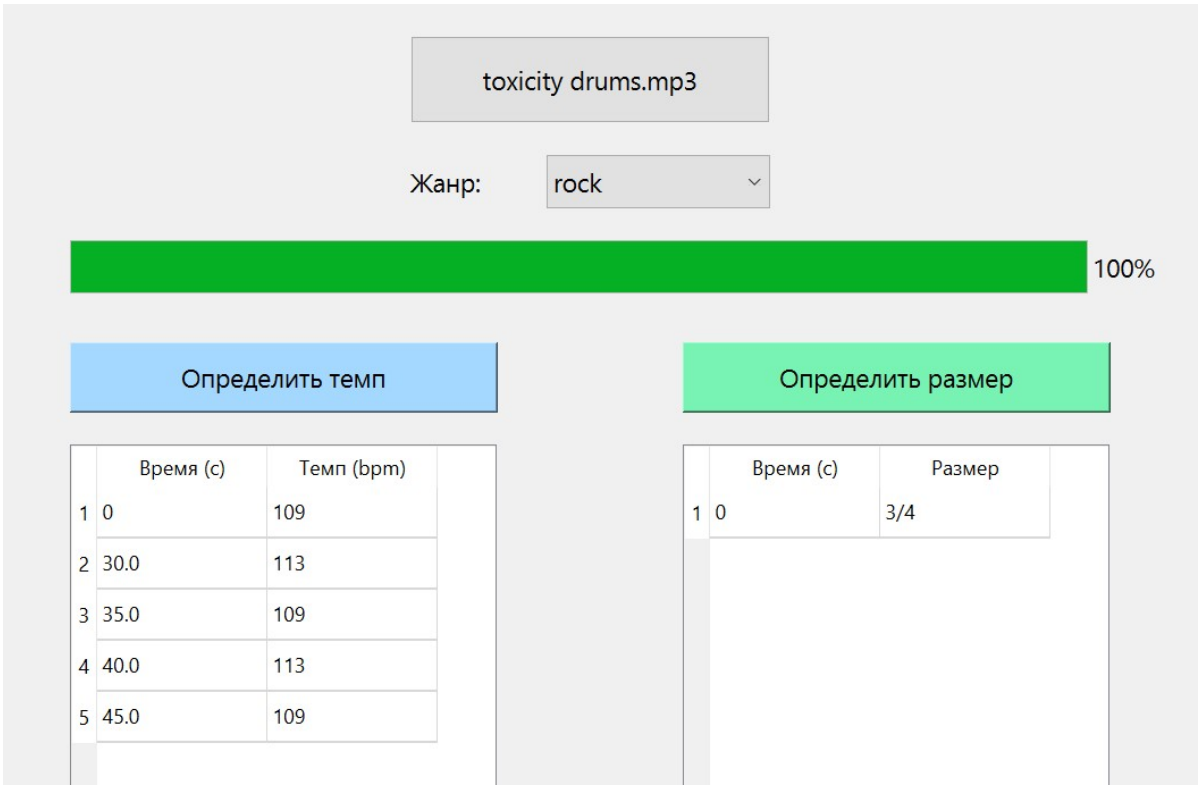

Рисунок 3.22 – Результаты для переменного темпа

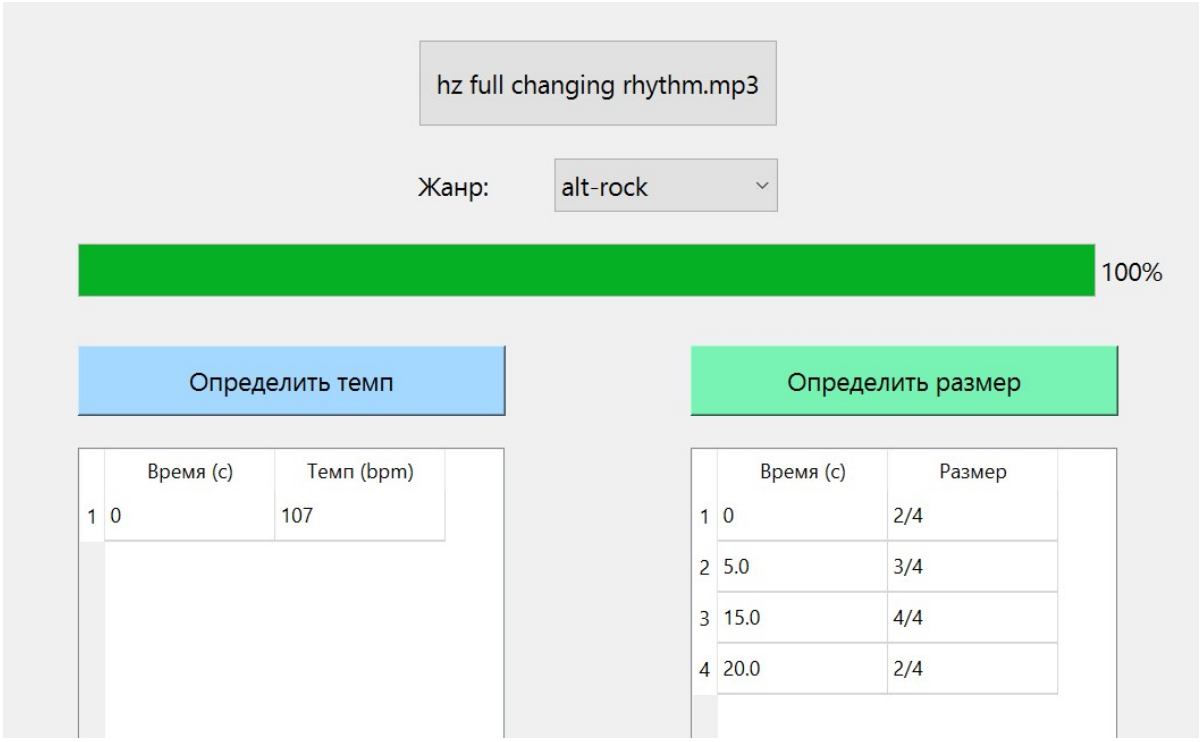

Рисунок 3.23 – Результаты для переменного размера

Выводы

В данном разделе были выбраны язык программирования и среда разработки, а также приведены детали реализации, пользовательский интерфейс и результаты тестирования разработанной программы.

В качестве языка программирования был выбран Python, а в качестве среды разработки – PyCharm.

Была описана реализация байесовских моделей для определения ритма и темпа, разделения аудиофайла на фрагменты и определения диапазона размеров.

Все модульные тесты были успешно пройдены. Результаты функциональных тестов также оказались в пределах допустимости.

4 Исследовательский раздел

4.1 Сравнение результатов работы метода с существующим аналогом

Для сравнения результатов работы ПО по определению темпа музыки использовались результаты работы метода beat_track() из библиотеки librosa. Этот метод определяет средний темп для указанного аудиофайла, поэтому для сравнения аудиозапись также разделялась на 5-секундные фрагменты, для каждой из которых вызывался этот метод.

Исследование проводилось на аудиофайле с темпом 115 bpm в первые 25 секунд и 120 bpm в оставшееся время.

В результате получены следующие значения:

Таблица 4.2 – Результаты работы ПО и аналога

Время (с)	Темп (bpm)	Темп (librosa, bpm)
0	109	152
30	113	161
35	109	117
40	113	161
45	109	123

Ошибка определения темпа для разработанного метода в таком случае составляет примерно 7%, а для библиотеки – 22%.

Для сравнения оценки постоянного темпа использовалась аудиозапись с темпом 100 bpm. В результате разработанный метод выдал 93 bpm, а существующее решение – 99 bpm. Таким образом, ошибка для разработанного ПО снова составила 7%, а для библиотеки – 1%.

На рисунке 4.24 представлена гистограмма, отображающая точность оценки темпа исследуемых программных решений.

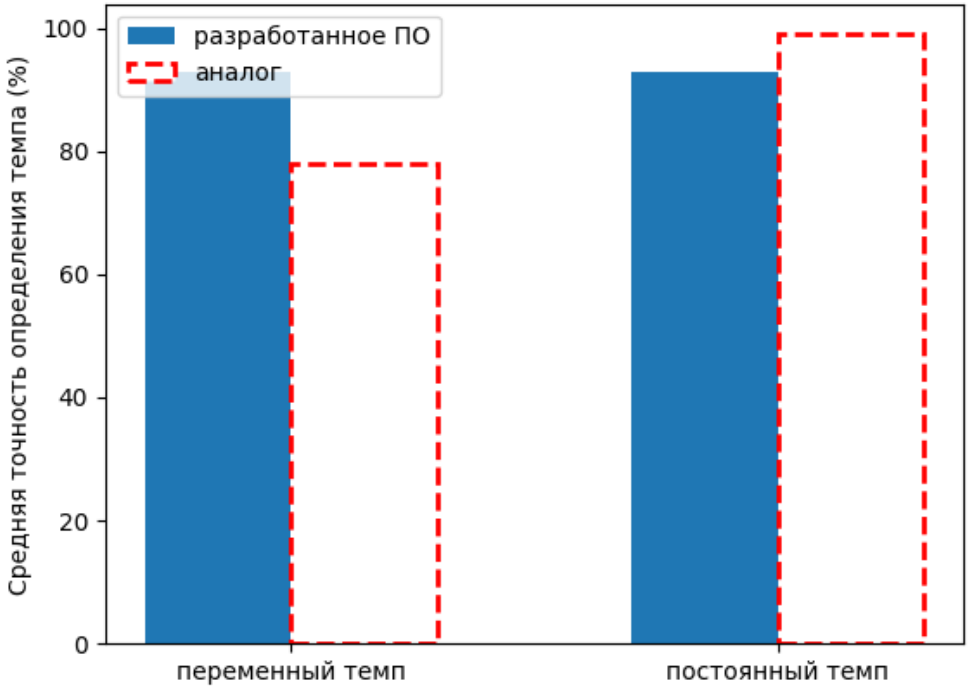

Рисунок 4.24 – Точность результатов работы ПО и аналога

Как видно из гистограммы, разработанный метод превосходит библиотеку в точности оценки переменного темпа музыки, но при этом точность определения постоянного темпа у него чуть ниже, чем у аналога.

4.2 Применимость ПО для аудиозаписей с разным набором инструментов

В качестве тестовых данных для анализа ритма использовались аудиофайлы со следующими инструментами:

1) Только ударные.
2) Только гитара.
3) Гитара вместе с ударными.

Во всех трех аудиозаписях использовался один и тот же музыкальный фрагмент с переменным размером (3/4 первые 15 секунд и 4/4 оставшееся время) (см. рис. 4.25).

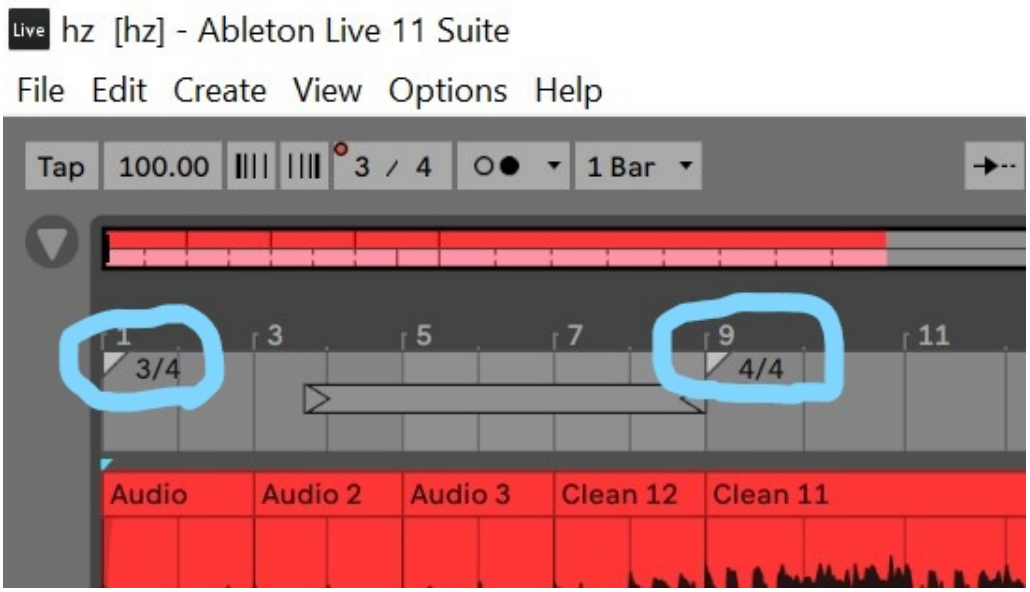

Рисунок 4.25 – Эталонные значения размера

Результаты работы ПО по определению переменного размера для трех указанных аудиофайлов представлены в таблицах 4.3 – 4.5 соответственно.

Таблица 4.3 – Результаты для 1-го аудиофайла

Время (с)	Размер
0	3/4
15	4/4

Таблица 4.4 – Результаты для 2-го аудиофайла

Время (с)	Размер
0	2/4
5	3/4
10	2/4
20	1/4

На рисунке 4.26 приведена средняя точность оценки тактового размера (в процентах) в зависимости от набора инструментов в аудиозаписи.

Таблица 4.5 – Результаты для 3-го аудиофайла

Время (с)	Размер
0	2/4
5	3/4
15	4/4
20	2/4

Как видно из гистограммы, наибольшую точность программа выдает на аудиофайле с ударными, а наименьшую – на аудиозаписи с гитарой. Такие результаты могут быть связаны с небольшими отклонениями гитары от ритмической сетки, а также со смещением сильных долей в гитарной партии. При этом во всех случаях точность определения тактового размера лежит в допустимых пределах ($\geq 82\%$).

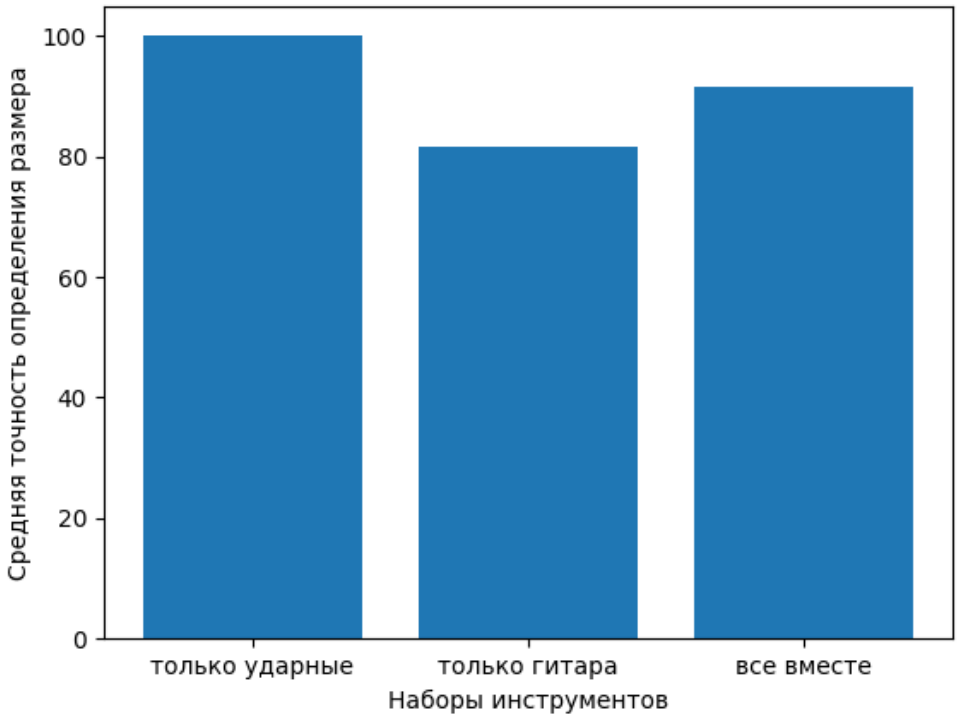

Рисунок 4.26 – точность оценки размера в зависимости от инструментов

Для анализа темпа в свою очередь использовались аудиофайлы со следующими инструментами:

1) Только ударные.
2) Гитары и бас.
3) Ударные, бас и гитара.
4) Полный фрагмент (ударные, бас, гитара и вокал).

Здесь использовался другой музыкальный фрагмент, также одинаковый для всех трех аудиофайлов, с переменным темпом (115 bpm первые 25 секунд и 120 bpm оставшееся время) («Toxicity» - System of a down).

Результаты работы ПО по определению переменного темпа для этих аудиофайлов представлены в таблицах 4.6 – 4.9 соответственно.

Таблица 4.6 – Результаты для 1-го аудиофайла

Время (с)	Темп (bpm)
0	105
30	113
35	105
40	113
45	105

На рисунке 4.27 приведена средняя точность оценки темпа в зависимости от набора инструментов в аудиозаписях для разработанного ПО.

Таблица 4.7 – Результаты для 2-го аудиофайла

Время (с)	Темп (bpm)
0	91
30	186
35	91
40	160
45	91

Таблица 4.8 – Результаты для 3-го аудиофайла

Время (с)	Темп (bpm)
0	131
5	103
10	131
15	103
20	131

Таблица 4.9 – Результаты для 4-го аудиофайла

Время (с)	Темп (bpm)
0	103
5	93
10	103
15	93

Как видно из гистограммы, наилучшие результаты разработанная программа показывает на аудизаписи, содержащей только ударные, и на аудиозаписи без вокала (с ударными, басом и гитарой). Наименьшая точность результатов

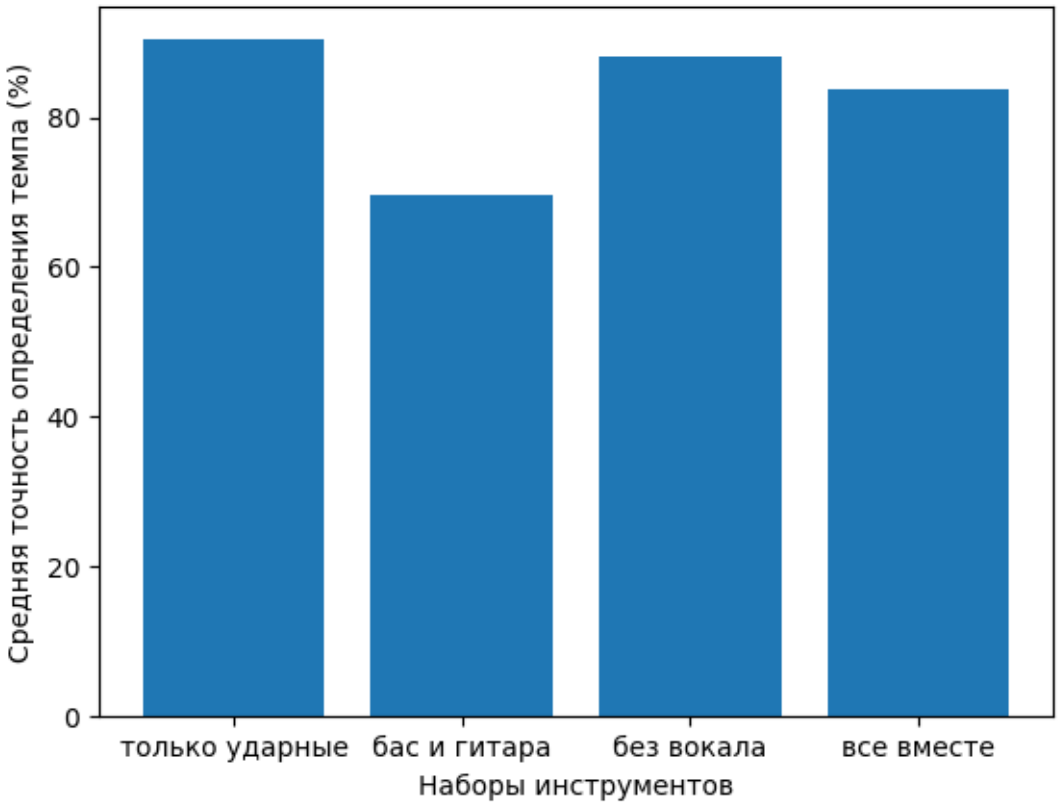

Рисунок 4.27 – точность оценки темпа в зависимости от инструментов

получилась на аудиофайле, содержащем только гитару и бас. Помимо перечисленных ранее причин, это может быть связано также с проблемами в качестве этой записи.

4.3 Применимость ПО для музыки разных жанров

Для исследования работы ПО в данном случае использовались аудиозаписи следующих жанров:

1) Поп (132 bpm, 4/4).
2) Рок (100 bpm, 4/4).
3) Фанк (89 bpm, 4/4).
4) Джаз (129 bpm, 4/4).

Результаты работы метода по определению темпа для перечисленных аудиофайлов представлены в таблицах 4.10 – 4.11.

Таблица 4.10 – Результаты определения темпа для разных жанров

Жанр	**Темп (bpm)**
Поп	118
Рок	113
Фанк	119

Таблица 4.11 – Результаты определения темпа для джаза

Время (с)	**Темп (bpm)**
0	85
5	99
30	85
35	99
60	128
65	99
80	128
85	191
90	87
95	33
100	99
115	191
120	99

На рисунке 4.28 представлена точность определения темпа музыки в зависимости от жанра.

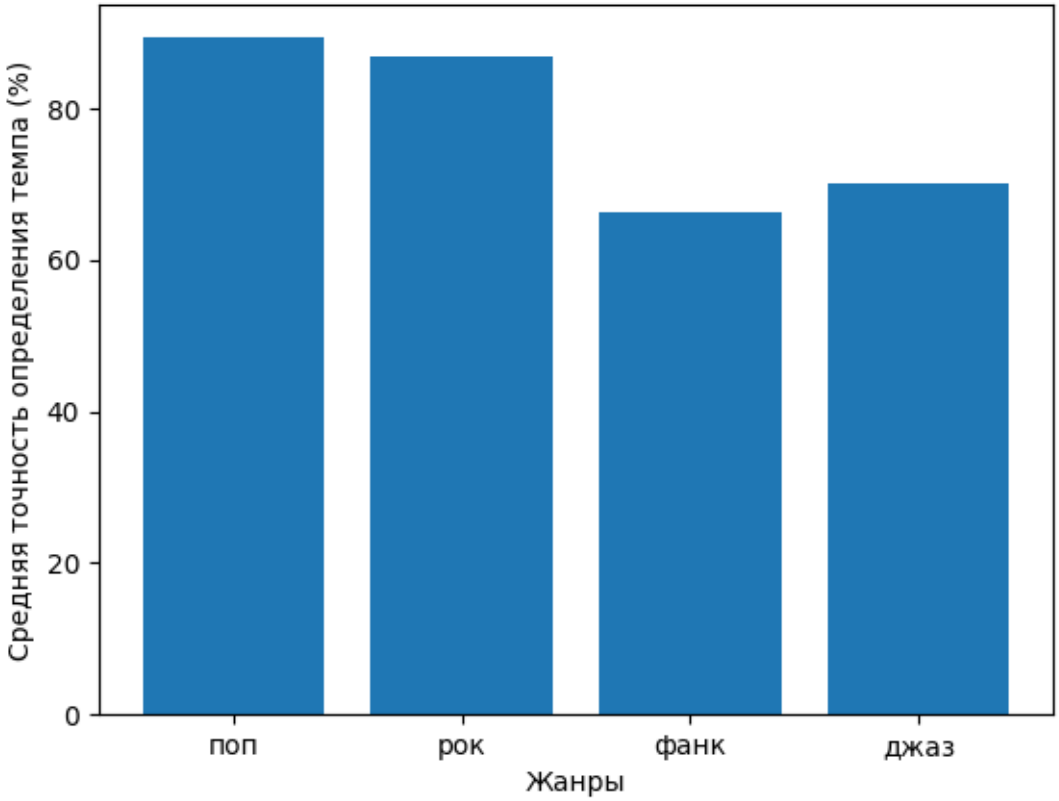

Рисунок 4.28 – точность оценки темпа в зависимости от жанра

Результаты по определению тактового размера для тех же жанров приведены в таблице 4.12.

Таблица 4.12 – Результаты определения размера для разных жанров

Жанр	**Тактовый размер**
Поп	2/4
Рок	2/4
Фанк	5/4
Джаз	3/4

На рисунке 4.29 представлена точность определения тактового размера в зависимости от жанра.

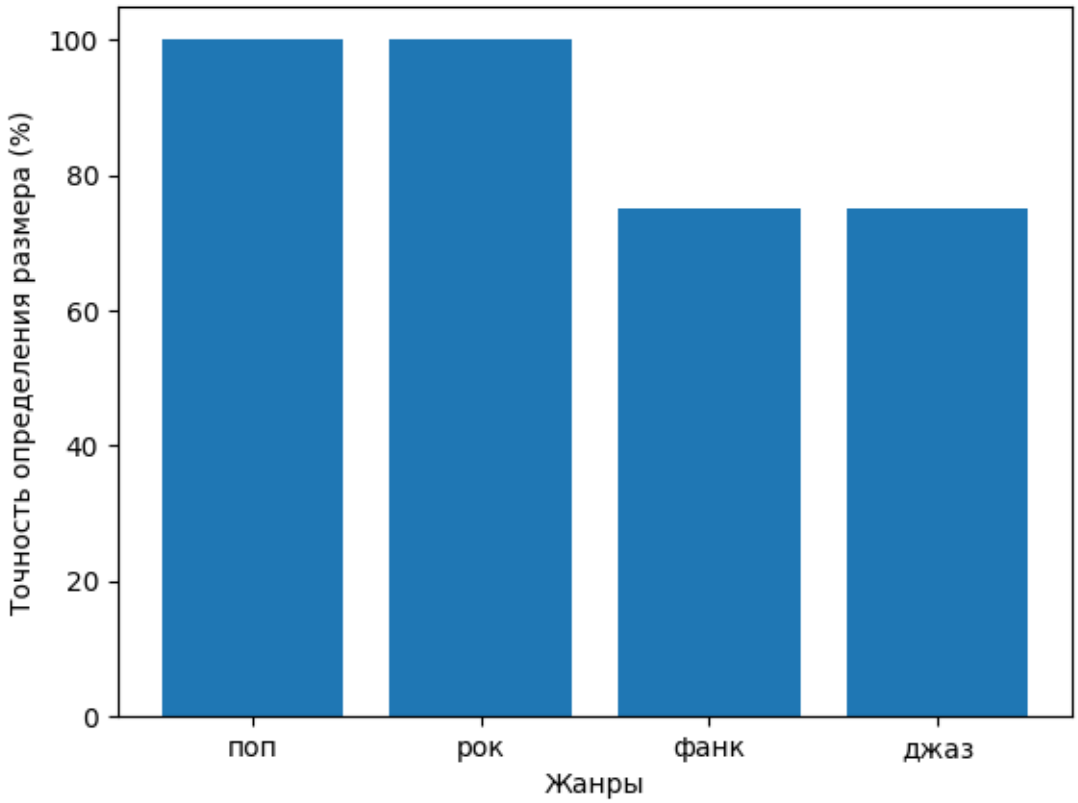

Рисунок 4.29 – точность оценки размера в зависимости от жанра

Как видно из гистограмм, точность оценки темпа и ритма значительно снижается на таких жанрах, как фанк и джаз. Это связано с особенностями этих жанров, такими как смещение ритма и сильных долей, что в свою очередь влияет и на определение темпа.

4.4 Оценка разработанного метода

В результате исследований у разработанного метода были выявлены следующие достоинства:

— более точное определение переменного темпа в сравнении с библиотечным методом;

— высокая точность определения темпа и ритма для аудиозаписей с ударными;

— высокая точность определения темпа и ритма для рок и поп музыки.

При этом у метода были обнаружены такие недостатки, как:

— менее точное определение постоянного темпа по сравнению с библиотечным методом;
— более существенные ошибки при оценке темпа и ритма музыки, содержащей только гитары;
— большие ошибки при определении тактового размера и темпа музыки таких жанров, как фанк и джаз.

Выводы

В данном разделе был проведён анализ применимости разработанного метода для музыки с разным составом инструментов и разных жанров.

В результате было выяснено, что наиболее высокой точности определения темпа и ритма метод достигает на музыке, содержащей ударные инструменты, а также в рок и поп жанрах.

Помимо этого, было проведено сравнение результатов разработанного метода с результатами библиотечной функции для определения темпа и выяснено, что разработанный метод превосходит библиотечный в точности при оценке переменного темпа примерно на 15%.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были рассмотрены методы автоматического определения ритмического рисунка и темпа цифровой музыкальной записи (ДВП, скрытые модели Маркова, байесовское иерархическое моделирование, сверточные нейросети), проведен обзор существующих решений, приведены результаты сравнительного анализа.

Был разработан метод определения переменного ритмического рисунка и переменного темпа цифровой музыкальной записи на основе байесовского иерархического моделирования.

Было разработано программное обеспечение, реализующее описанный метод, выполнено его тестирование, описан пользовательский интерфейс.

После этого было проведено исследование применимости разработанного программного обеспечения на музыке с разным составом инструментов и разных жанров. А также выполнено сравнение результатов работы реализованного метода с результатами, полученными с помощью библиотечной функции.

Таким образом, поставленная цель – реализовать метод автоматического определения темпа и ритма музыки на основе байесовского иерархического моделирования – была достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Benetos E. / Automatic music transcription: challenges and future directions / Benetos E., Dixon S., Giannoulis D., Kirchhoff H., Klapuri A. // Journal of Intelligent Information Systems. – 2013. – С. 407-434.

2. Музыкальный словарь Гроува. // Москва. – 2007. – С. 858.

3. Чехович Д. О. Ритм музыкальный // Большая российская энциклопедия. Москва. – 2015. – Том 28. – С. 541.

4. Sachs C. Rhythm and tempo: a study in music history. – 1953.

5. Cemgil A. T., Desain P., Kappen B. Rhythm quantization for transcription // Computer Music Journal. – 2000. – С. 60-76.

6. Polikar R. The wavelet tutorial // 2-е изд. – 2001. – 67 с.

7. Tzanetakis G., Essl G., Cook P. Audio analysis using the Discrete Wavelet Transform. – 2001.

8. Биккенин Р. Р., Чесноков М. Н. Теория электрической связи. – 2010. – 329 с.

9. Pleshkova S., Panchev K., Bekyarski A. Development of a MIDI synthesizer for test signals to a wireless acoustic sensor network. – 2020.

10. Takeda H., Saito N., Otsuki T. Hidden Markov model for automatic transcription of MIDI signals. – 2002. – С. 428-431.

11. Kübler R. Bayesian Hierarchical Modeling in PyMC3. – 2021.

12. Rouder J. N., Lu J. An introduction to Bayesian hierarchical models with an application in the theory of signal detection // Psychonomic Bulletin & Review. – 2005. – С. 573-604.

13. Nakamura E., Itoyama K., Yoshii K. Rhythm transcription of MIDI performances based on Hierarchical Bayesian Modelling of repetition and modification of musical note patterns. – 2016.

14. Stevens S.S., Volkmann J., Newman E.B. A scale for the measurement of the psychological magnitude pitch // Acoustical Society of America. – 1937. – C. 188.

15. Schreiber H., Muller M. A single-step approach to musical tempo estimation using a convolutional neural network // 2018. – C. 98-105.

16. Clevert D.A., Unterthiner T., Hochreiter S. Fast and accurate deep network learning by exponential linear units (elus). – 2015.

17. Ioffe S., Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. – 2015.

18. Spotify Tracks Dataset [Электронный ресурс]. – Режим доступа: https://www.kaggle.com/datasets/maharshipandya/-spotify-tracks-dataset (дата обращения: 15.05.2023).

19. Librosa: audio and music processing in Python [Электронный ресурс]. – Режим доступа: `https://librosa.org/doc/latest/index.html` (дата обращения: 17.05.2023).

20. PyMC3 Documentation [Электронный ресурс]. – Режим доступа: `https://www.pymc.io/projects/docs/en/v3/` (дата обращения: 17.05.2023).

21. NumPy documentation [Электронный ресурс]. – Режим доступа: `https://numpy.org/doc/stable/` (дата обращения: 17.05.2023).

ПРИЛОЖЕНИЕ А

Листинг 6: модуль оценки темпа и ритма

```
1   import pandas as pd
2   import librosa
3   from scipy import stats
4   import numpy as np
5   import os
6
7   def genres_to_ints(genre_dataset):
8     all_genres = genre_dataset.unique()
9     genres_dict = {}
10    for i, genre in enumerate(all_genres):
11      genres_dict[genre] = i
12    genres_ints = np.array([0] * len(genre_dataset))
13    for i, genre in enumerate(genre_dataset.values):
14      genres_ints[i] = genres_dict[genre]
15    return genres_ints
16
17  def get_tempo_bounds(bpm_dataset, genre_dataset, track_genre: str):
18    rows_num = [i for i in range(len(genre_dataset)) if genre_dataset.values[
      i] == track_genre]
19    tempos = [bpm_dataset.values[i] for i in rows_num]
20    max_bpm = max(tempos)
21    min_bpm = min(tempos)
22    return min_bpm, max_bpm
23
24  def estimate_bpm(audio_path: str, bpm_dataset, genre_dataset, track_genre:
      str, step: int, progress) -> dict:
25    y, sr = librosa.load(audio_path)
26    prior_bpm, _ = librosa.beat.beat_track(y=y, sr=sr)
27    print(prior_bpm)
28    min_bpm, max_bpm = get_tempo_bounds(bpm_dataset, genre_dataset,
      track_genre)
29    genres_ints = genres_to_ints(genre_dataset)
30    trace = bpm_model(min_bpm, max_bpm, bpm_dataset, genre_dataset,
      genres_ints, progress)
31    split_mp3(audio_path, step)
32    bpms = []
33    times = []
```

```
34      t = 0
35      for audio in os.listdir("tmp/"):
36        y, sr = librosa.load("tmp/" + audio)
37        genre_int = np.where(genre_dataset.unique() == track_genre)[0][0]
38        delta = 40
39        prior_bpm, _ = librosa.beat.beat_track(y=y, sr=sr)
40        tempos = np.arange(prior_bpm - delta, prior_bpm + delta, 0.5)
41        genres_samples = trace['genre_coef']
42        genre_coef_samples = [genres_samples[i][genre_int] for i in range(len(
      genres_samples))]
43        density = stats.gaussian_kde(genre_coef_samples)
44        coef_estimate = genre_coef_samples[density(genre_coef_samples).argmax()
      ]
45        sigma = (max_bpm - min_bpm) / 12.0
46        tempo_samples = trace['tempo']
47        density = stats.gaussian_kde(tempo_samples)
48        bpms.append(round(tempos[density(tempos).argmax()] + coef_estimate *
      sigma))
49        times.append(t)
50        t += (step / 1000)
51      inds_to_delete = []
52      for i in range(1, len(bpms)):
53        if bpms[i] == bpms[i - 1]:
54          inds_to_delete.append(i)
55      for i in range(len(inds_to_delete)-1, -1, -1):
56        bpms.pop(inds_to_delete[i])
57        times.pop(inds_to_delete[i])
58      res_tempos = {times[i]: bpms[i] for i in range(len(bpms))}
59      clean_dir("tmp/")
60      remove_dir("tmp/")
61      return res_tempos
62
63    def calc_measure(downbeats: list) -> int:
64      downbeat_inds = [i for i, beat in enumerate(downbeats) if beat == 1]
65      if len(downbeat_inds) <= 1:
66        return 0
67      difs_sum = 0
68      for i in range(1, len(downbeat_inds)):
69        difs_sum += (downbeat_inds[i] - downbeat_inds[i - 1])
70      avg_measure = difs_sum / (len(downbeat_inds) - 1)
```

```
71      return round(avg_measure)
72
73  def get_measure_range(audio_path: str, tail: list):
74      y, sr = librosa.load(audio_path)
75      onset_env = librosa.onset.onset_strength(y=y, sr=sr)
76      peaks = librosa.util.peak_pick(onset_env, pre_max=50, post_max=50,
        pre_avg=50, post_avg=50, delta=0.5, wait=10)
77      peaks_time = librosa.frames_to_time(peaks, sr=sr)
78      _, beats = librosa.beat.beat_track(onset_envelope=onset_env, sr=sr)
79      beats_time = librosa.frames_to_time(beats, sr=sr)
80      downbeat_times = {}
81      for i, beat in enumerate(beats):
82          beat_range = list(range(beat - 3, beat + 4))
83          for b in beat_range:
84              if b in peaks:
85                  downbeat_times[i] = beat
86      downbeats = [0 for i in range(len(beats))]
87      for beat in downbeat_times.keys():
88          downbeats[beat] = 1
89      downbeats = tail + downbeats
90      prior_measure = calc_measure(downbeats)
91      if prior_measure == 0:
92          return -1, downbeats, prior_measure
93      if prior_measure > 2:
94          return 0, np.arange(prior_measure - 2, prior_measure + 3, 1),
        prior_measure
95      elif prior_measure == 2:
96          return 0, np.arange(prior_measure - 1, prior_measure + 3, 1),
        prior_measure
97      else:
98          return 0, np.arange(prior_measure, prior_measure + 3, 1), prior_measure
99
100 def estimate_rhythm(audio_path: str, rhythm_dataset, step: int, progress)
      -> dict:
101     _, prior_range, _ = get_measure_range(audio_path, [])
102     trace = rhythm_model(prior_range[0], prior_range[-1], rhythm_dataset,
        progress)
103     split_mp3(audio_path, step)
104     measures = []
105     times = []
```

```
106     t = 0
107     tail = []
108     for audio in os.listdir("tmp/"):
109       er, measure_range, sig = get_measure_range("tmp/" + audio, tail)
110       if er == -1:  #                          ,
111         tail = measure_range
112         continue
113       else:
114         tail = []
115       measure_samples = trace['measure']
116       density = stats.gaussian_kde(measure_samples)
117       measure_estimate = measure_range[density(measure_range).argmax()]
118       measures.append(round(measure_estimate))
119       times.append(t)
120       t += (step / 1000)
121     inds_to_delete = []
122     for i in range(1, len(measures)):
123       if measures[i] == measures[i - 1]:
124         inds_to_delete.append(i)
125     for i in range(len(inds_to_delete) - 1, -1, -1):
126       measures.pop(inds_to_delete[i])
127       times.pop(inds_to_delete[i])
128     res_measures = {times[i]: measures[i] for i in range(len(measures))}
129     clean_dir("tmp/")
130     remove_dir("tmp/")
131     return res_measures
```

ПРИЛОЖЕНИЕ Б

Листинг 7: байесовские модели

```
import pymc3 as pm

def rhythm_model(measure_min, measure_max, rhythm_dataset, progress):
  with pm.Model() as model:
    measure = pm.Uniform('measure', lower=measure_min, upper=measure_max)
    progress.setValue(20)
    mu = (measure_min + measure_max) / 2.0
    progress.setValue(40)
    sigma = (measure_max - measure_min) / 12.0
    progress.setValue(60)
    measure_obs = pm.Normal('measure_obs', mu=mu, sd=sigma, observed=
  rhythm_dataset)
    progress.setValue(80)
    trace = pm.sample(1000, tune=1000, chains=2)
    progress.setValue(100)
  return trace

def bpm_model(min_bpm: int, max_bpm: int, bpm_dataset, genre_dataset,
  genres_ints, progress):
  with pm.Model() as model:
    tempo = pm.Uniform('tempo', lower=min_bpm, upper=max_bpm)
    progress.setValue(20)
    mu = (min_bpm + max_bpm) / 2.0
    sigma = (max_bpm - min_bpm) / 12.0
    genre_coef = pm.Normal('genre_coef', mu=0, sd=1, shape=len(
  genre_dataset.unique()))
    progress.setValue(40)
    bpm_est = mu + genre_coef[genres_ints] * sigma
    progress.setValue(60)
    bpm_obs = pm.Normal('bpm_obs', mu=bpm_est, sd=sigma, observed=
  bpm_dataset)
    progress.setValue(80)
    trace = pm.sample(1000, tune=500, chains=2, cores=1)
    progress.setValue(100)
  return trace
```

ПРИЛОЖЕНИЕ В

Листинг 8: разделение аудиофайла

```
from pydub import AudioSegment
import os

def split_mp3(audio_path: str, step: int):
  if not os.path.isdir("tmp"):
    os.mkdir("tmp")
  audio_file = AudioSegment.from_file(audio_path, format="mp3")
  part_length = step
  parts = [audio_file[i:i + part_length] for i in range(0, len(audio_file),
   part_length)]
  for i, part in enumerate(parts):
    part.export(f"tmp/part_{i}.mp3", format="mp3")
```

Листинг 9: очистка временных директорий

```
import os

def clean_dir(dir_path: str):
  for root, dirs, files in os.walk(dir_path):
    for name in files:
      os.remove(os.path.join(root, name))

def remove_dir(dir_path: str):
  if os.path.isdir(dir_path):
    os.rmdir(dir_path)
```