

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»

Магистерская программа: «Прикладной искусственный интеллект»

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

РАЗРАБОТКА ОБУЧАЮЩЕЙ СРЕДЫ ДЛЯ МУЗЫКАНТА С ЭЛЕМЕНТАМИ ПРИКЛАДНОГО ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Студент

Трошин - 10.06.2026
(подпись, дата)

Трошин Александр Андреевич
(ФИО полностью)

Руководитель

Бождай 10.06.2026
(подпись, дата)

Бождай А.С.
(фамилия, инициалы)

Нормоконтролёр

Бычков 10.06.2026
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Рецензент

доцент Пенз ГТУ Петерский 17.06.26
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от) 22.05.2026 № 14

Заведующий кафедрой

Митрохин
(подпись) Митрохин М.А.
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 23.06.2026 № 33)

Секретарь ГЭК

(подпись) (фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

«Утверждаю» Заведующий кафедрой

М.А. Митрохин

«20» сентября 2024 г.

ЗАДАНИЕ

НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ

1. Студент Трошин Александр Андреевич гр.24ВВИм2 факультета вычислительной техники, направления 09.04.01
2. Тема работы «Разработка обучающей среды для музыканта с элементами прикладного искусственного интеллекта».
- Тема утверждена приказом ПГУ № 920/о от "20" сентября 2024 г.
3. Руководитель магистерской диссертации д.т.н., профессор Бождай А.С.
4. Задание на работу (назначение разработки, исходные данные и т.п.)

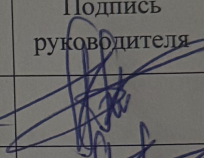
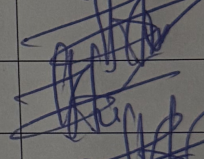
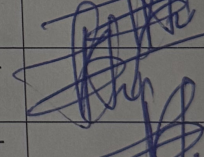
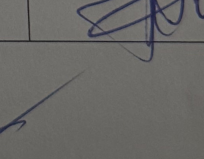
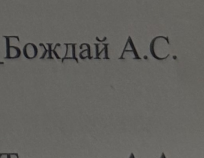
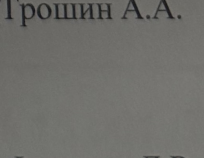
Назначение разработки: Информационная поддержка для обучения музыкантов с использованием технологий прикладного искусственного интеллекта, способного осуществлять анализ музыкального исполнения пользователя, выявлять ошибки игры и формировать персонализированные рекомендации по совершенствованию исполнительских навыков.

В качестве пользовательского интерфейса использовать мобильное приложение для ОС «Android».

5. Перечень подлежащих разработке вопросов:

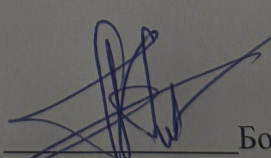
1. Анализ предметной области обучающих сред и возможностей их использования в рамках мобильных приложений.
2. Исследование и сравнительный анализ методов цифровой обработки аудиосигналов, нейросетевых моделей и программных средств для анализа музыкального исполнения.
3. Разработка обучающей среды для музыканта с элементами прикладного искусственного интеллекта.
4. Отладка обучающей среды для музыканта с элементами прикладного искусственного интеллекта.

6. Календарный график выполнения работы

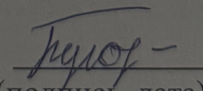
№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области обучающих сред	10%	20.09.2024- 20.11.2024	
2.	Анализ предметной области мобильных приложений	10%	21.11.2024- 21.01.2025	
3.	Проектирование структуры разрабатываемой среды	20%	22.01.2025- 22.04.2025	
4.	Разработка всех компонентов обучающей среды	40%	23.04.2025- 28.01.2026	
5.	Отладка обучающей среды	10%	29.01.2026- 30.03.2026	
6.	Оформление выпускной работы	10%	31.03.2026- 31.05.2026	

Дата выдачи задания "20" сентября 2024 г.

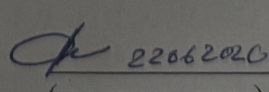
Руководитель магистерской диссертации


(подпись, дата) Бождай А.С.

Задание к исполнению принял студент


(подпись, дата) Трошин А.А.

Работу к защите допустить
Декан факультета


(подпись, дата) 22.06.2026 Фионова Л.Р.

РЕФЕРАТ

Магистерская работа содержит 98 листов, 31 рисунок, 7 таблиц, 14 источников, 2 приложения.

ОБУЧАЮЩИЕ СРЕДЫ, АНАЛИЗ, ГЕНЕРАЦИЯ УПРАЖНЕНИЙ, НЕЙРОННЫЕ СЕТИ, МОБИЛЬНЫЕ ПРИЛОЖЕНИЯ, ANDROID

Цель работы – разработка мобильного приложения для обучения музыкантов с использованием технологий прикладного искусственного интеллекта, предназначенного для анализа музыкального исполнения, выявления ошибок игры и формирования персонализированных рекомендаций по развитию исполнительских навыков.

Объектом разработки является интеллектуальная система поддержки музыкального обучения, реализованная в виде мобильного приложения для платформы Android.

В результате работы разработана и протестирована комплексная обучающая среда для музыкантов, включающая модуль анализа музыкального исполнения, систему формирования рекомендаций, музыкальный дневник, справочник теоретических материалов, тюнер, метроном и средства взаимодействия обучающегося с преподавателем. Выполнено проектирование архитектуры приложения и модели данных, реализованы механизмы хранения пользовательской информации, обработки аудиосигналов и интеграции методов машинного обучения.

Для интеллектуального анализа музыкального исполнения исследованы методы цифровой обработки аудиосигналов и глубокого обучения

Разработанная система также представляет интерес для дальнейших исследований в области интеллектуальных обучающих систем и применения технологий искусственного интеллекта в образовательных мобильных приложениях.

					ПГУ 1.090401.15.001 ПЗ			
зм.	Лист	№ докум.	Подпись	Дата	Разработка обучающей среды для музыканта с элементами прикладного искусственного интеллекта	Лит.	Лист	Листов
зrab.		Трошин А.А.		10.06.24				
ровер.		Бождай А.А.		10.06.24			2	98
Контр		Бычков А.С.		10.06.24				
тверд.		Митрохин М.А.		10.06.24		ФВТ, каф. ВТ гр. 24ВВим2		
					Пояснительная записка			

Содержание

1 Анализ предметной области и существующих подходов к обучению музыкантов с применением технологий искусственного интеллекта	4
1.1 Актуальность исследования	4
1.2 Анализ предметной области	5
1.3 Анализ существующих мобильных решений	7
1.3.1 Детальный анализ существующих мобильных приложений	7
1.4 Методы цифровой обработки аудиосигналов	9
1.4.1 Быстрое преобразование Фурье	10
1.4.2 Кратковременное преобразование Фурье	10
1.5 Методы искусственного интеллекта в задачах анализа аудио	11
1.6 Архитектура YAMNet	12
1.6.1. Этапы обработки аудиосигнала	12
1.6.2. Преимущества архитектуры YAMNet для мобильных устройств ...	13
1.7 Архитектура MVC	14
1.8 Вывод по первому разделу	15
2 Архитектура программного комплекса	17
2.1 Общая архитектура разработки	17
2.2 Проектирование системы	19
2.2.1 Структура базы данных	21
2.3 Проектирование пользовательского интерфейса	23
2.4 UML-моделирование системы	25
2.4.1 Диаграмма прецедентов	25
2.4.2 Диаграмма последовательностей	26
2.5 Вывод по второму разделу	28
3 Разработка структуры мобильного приложения	29
3.1 Реализация подсистемы записи и обработки аудиосигнала	29
3.1.1 Реализация многопоточной обработки аудиопотока	30
3.2 Реализация предварительной обработки аудиоданных	31
3.3 Интеграция модели YAMNet в мобильное приложение	33
3.4 Разработка системы классификации музыкального исполнения	33

3.5 Формирование обучающей выборки.....	34
3.5.1 Балансировка обучающей выборки.....	34
3.6 Реализация аугментации аудиоданных	35
3.7 Дообучение модели YAMNet	36
3.7.1 Мониторинг процесса обучения модели	36
3.7.2 Оценка качества модели	37
3.8 Экспорт модели TensorFlow Lite.....	38
3.9 Реализация подсистемы генерации рекомендаций.....	38
3.9.1 Генерация персонализированных упражнений.....	39
3.10 Реализация хранения результатов анализа	40
3.11 Оптимизация производительности мобильного приложения	40
3.11.1 Оптимизация энергопотребления мобильного устройства	41
3.11.2 Анализ матрицы ошибок.....	41
3.12 Графики обучения.	42
3.14 Вывод по третьему разделу.....	45
4 Тестирование приложения.....	47
Заключение.....	65
Список используемых источников	66
ПРИЛОЖЕНИЕ А	68
ПРИЛОЖЕНИЕ Б.....	87

Введение

Мобильные приложения уже стали неотъемлемой частью современного мира. Их разрабатывают для самых разных областей: и для работы, и для обучения, и для развлечений. Одной из сфер деятельности стала музыкальное образование. Мобильное приложение – одно из удобных и эффективных средств для улучшения как практических, так и теоретических знаний.

Ключевой инновацией разрабатываемого приложения является использование прикладного искусственного интеллекта для анализа игры пользователя и последующей генерации упражнений.

Разработка приложения с использованием прикладного искусственного интеллекта может упростить людям первые шаги в мир музыки, сделав его более интересным и простым. Такой инструмент поможет объективно посмотреть на свои навыки, а также покажет путь по их улучшению. Также одна из целей – это показать, как можно использовать искусственный интеллект в обучении людей музыки.

В итоге всего вышесказанного - целью является создание мобильного приложения, интеграция всех главных инструментов, предоставление удобных и эффективных подсобных средств для гитаристов с разным уровнем игры. Разработка и успешное внедрение приложения подчеркнет потенциал мобильных технологий в качестве помощника образования.

1 Анализ предметной области и существующих подходов к обучению

музыкантов с применением технологий искусственного интеллекта

Современные технологии вносят существенные изменения к подходам обучения, способам получения информации и её доступности. Одно из наиболее развивающихся направлений – это внедрения элементов искусственного интеллекта в различные системы.

Сфера музыки тоже с этим столкнулась, т.к. интеллектуальный анализ все более развивается, и с этим появляются все больше путей для изучения материала.

1.1 Актуальность исследования

Традиционные методы обучения игре на музыкальных инструментах предполагают непосредственное взаимодействие между обучающимся и преподавателем [1].

При таком виде обучения полезно иметь систему-органайзер, в которой можно вести музыкальный дневник, планировать занятия, а также хранить учебный материал в одном месте.

Несмотря на высокую эффективность подобных подходов, они обладают рядом ограничений, таким как:

- высокая стоимость обучения,
- временные рамки доступности преподавателей,
- сложность самоорганизации,
- субъективность оценки качества исполнения.

Развитие мобильных устройств и технологий позволило частично автоматизировать отдельные аспекты музыкального обучения [2]. Современные мобильные приложения предоставляют пользователям

цифровые тюнеры с разными строями, метрономы, обширные библиотеки аккордов, интерактивные упражнения и обучающие материалы.

Однако большинство существующих решений не обеспечивает анализа качества музыкального исполнения пользователя.

Современные достижения в области машинного обучения и цифровой обработки сигналов позволяют реализовать интеллектуальные системы для сопровождения музыкального обучения, которые способны автоматически анализировать аудиосигналы и формировать персонализированные рекомендации.

Особый интерес представляет использование методов глубокого обучения совместно с мобильными платформами. Развитие технологий локального выполнения моделей машинного обучения позволяет реализовать интеллектуальный анализ аудиоданных как непосредственно на мобильном устройстве, так и с использованием серверов.

Применение локального анализа обеспечивает:

- снижение задержек обработки,
- повышение конфиденциальности пользовательских данных,
- уменьшение зависимости от интернет-соединения,
- повышение автономности.

1.2 Анализ предметной области

Предметная область исследования включает процессы музыкального обучения, мобильные образовательные технологии, методы цифровой обработки аудиосигналов и технологии искусственного интеллекта.

Обучение игре на музыкальном инструменте (в нашем случае - гитаре) предполагает формирование следующих навыков:

- постановка аккордов,
- развитие чувства ритма,
- контроль чистоты исполнения,
- развитие музыкального слуха,

- формирование моторных навыков.

Существующие подходы к обучению игре на гитаре можно разделить на несколько категорий.

Традиционное обучение:

Предполагает непосредственное взаимодействие обучающегося и преподавателя.

Преимущества:

- персонализированная обратная связь,
- высокая гибкость образовательного процесса.

Недостатки:

- ограниченная доступность,
- высокая стоимость,
- зависимость качества обучения от субъективных факторов.

Онлайн-обучение:

Использует цифровые образовательные платформы и видеоматериалы.

Преимущества:

- доступность;
- гибкость графика обучения;
- широкий объём обучающих материалов.

К недостаткам можно отнести отсутствие интеллектуального контроля качества исполнения.

Мобильные приложения:

Современные мобильные решения обеспечивают:

- работу тюнера,
- использование метронома,
- просмотр аккордов,
- ведение музыкального дневника,
- интерактивные упражнения.

Недостатком большинства существующих мобильных приложений является отсутствие анализа игры пользователя.

1.3 Анализ существующих мобильных решений

На современном рынке представлены различные мобильные приложения для музыкантов.

1.3.1 Детальный анализ существующих мобильных приложений

Yousician представляет собой интерактивную обучающую платформу для игры на гитаре, фортепиано, бас-гитаре и укулеле. Приложение построено по принципу «обучение через практику» и предлагает пользователю выполнять упражнения под аккомпанемент в реальном времени. Система анализирует звук с микрофона и определяет правильность сыгранных нот и аккордов, выстраивая прогрессию обучения от простых упражнений к сложным композициям. Приложение поддерживает множество музыкальных стилей (рок, поп, джаз, классика) и включает в себя обширную библиотеку песен. Yousician предоставляет визуальную обратную связь в виде подсветки нот на грифе инструмента, но не даёт подробного анализа причин ошибок (например, ритмических неточностей или проблем с постановкой пальцев). Приложение требует постоянного подключения к интернету для загрузки уроков и песен, что ограничивает его использование в автономном режиме. Базовый функционал доступен бесплатно, но полноценный доступ ко всем урокам и персонализированной траектории обучения предоставляется только по платной подписке [3].

Melodics ориентировано на обучение игре на MIDI-контроллерах, драм-машинах и электронных ударных установках, а также на акустической гитаре и фортепиано. Особенностью приложения является использование методов машинного обучения для отслеживания прогресса пользователя и подбора упражнений на основе его сильных и слабых сторон. В отличие от Yousician, Melodics уделяет внимание развитию чувства ритма и точности нажатия клавиш, обеспечивая детализированную статистику успеваемости.

Приложение имеет простой и минималистичный интерфейс, что снижает когнитивную нагрузку на пользователя. Однако стоимость подписки выше, чем у других решений, а функционал ограничен преимущественно электронными инструментами. Для гитаристов возможности Melodics ограничены, приложение не предоставляет детального анализа постановки аккордов и чистоты звукоизвлечения, а также отсутствует возможность генерировать персонализированные упражнения на основе ошибок пользователя. Кроме того, Melodics также требует подключения к интернету для загрузки материалов и синхронизации прогресса [4].

SmartMusic разработано для образовательных учреждений и преподавателей музыки. Приложение предоставляет инструменты для практики, самоконтроля и дистанционного оценивания. SmartMusic включает в себя обширную библиотеку учебных материалов и может анализировать интонацию, ритм и технику исполнения, выделяя ошибки на нотном стане. Система позволяет преподавателям создавать персонализированные задания, отправлять их студентам и отслеживать прогресс. Благодаря этому SmartMusic используется в школах и музыкальных академиях как вспомогательное средство обучения. Однако приложение ориентировано преимущественно на духовые и струнные инструменты, а поддержка гитары реализована не в полной мере. Интерфейс SmartMusic перегружен функциями, что может вызывать трудности у начинающих пользователей. Кроме того, для использования всех возможностей требуется платная образовательная лицензия, а некоторые функции зависят от облачной обработки данных. Приложение также не предоставляет возможности автоматической генерации упражнений на основе ошибок пользователя [5].

Функциональные возможности существующих решений представлены в таблице 1.

Таблица 1 — Сравнение существующих решений

Критерий	Yousician [3]	Melodics[4]	SmartMusic[5]	Разрабатываемая система
Использование ИИ	+	+	+	+
Персонализация	—	+	+	+
Анализ ритма	+	+	+	+
Анализ интонации	+	—	+	+
Генерация упражнений	—	—	—	+
Поддержка нескольких инструментов	+	—	—	+
Локальная обработка	—	—	—	+
Удобство интерфейса	+	+	—	+

Проведённый анализ показал следующие ограничения современных решений:

- ограниченные возможности интеллектуального анализа;
- недостаточная персонализация обучения;
- высокая зависимость от облачной обработки данных;
- ограниченная адаптация под индивидуальные особенности пользователя.

1.4 Методы цифровой обработки аудиосигналов

Анализ музыкального исполнения требует применения методов цифровой обработки сигналов.

Для анализа музыкальных характеристик используются методы спектрального анализа.

1.4.1 Быстрое преобразование Фурье

Быстрое преобразование Фурье позволяет переводить временное представление сигнала в частотное.

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi kn}{N}}$$

где: $x(n)$ — входной сигнал;

$X(k)$ — спектр сигнала;

N — количество отсчётов;

k — индекс частоты.

Преимущества метода:

- высокая вычислительная эффективность,
- возможность работы в реальном времени.

К недостаткам можно отнести отсутствие временной локализации сигнала.

1.4.2 Кратковременное преобразование Фурье

Для анализа изменяющихся во времени сигналов используется кратковременное преобразование Фурье.

$$X(m, \omega) = \sum_{n=-\infty}^{\infty} x(n)w(n - m) e^{-j\omega n}$$

где: $w(n)$ — оконная функция,

m — временной индекс,

ω — частота.

Использование STFT позволяет учитывать временную структуру музыкального сигнала [6].

1.4.3 Mel-преобразование

Особенности восприятия звука человеком сильно отличаются от линейного распределения частот. Человеческий слух воспринимает низкие частоты более детально по сравнению с высокими [7].

Для приближения обработки сигнала к особенностям слухового восприятия применяется Mel-шкала:

$$Mel(f) = 2595 \log_{10} \left(1 + \frac{f}{700} \right)$$

где:

f — частота сигнала.

Mel-спектрограммы широко используются современными системами анализа для аудиосигналов [8].

1.5 Методы искусственного интеллекта в задачах анализа аудио

Современные системы анализа аудиосигналов используют методы машинного обучения и глубокого обучения [9].

Наиболее распространёнными подходами являются такие как:

– свёрточные нейронные сети - CNN обеспечивают автоматическое выделение признаков из спектрограмм аудиосигнала. Преимуществами данного подхода являются: высокая точность классификации и устойчивость к шумам. А недостатки – высокая вычислительная сложность.

– рекуррентные нейронные сети - RNN ориентированы на анализ временных последовательностей. К преимуществам такого подхода относятся: анализ временных зависимостей и эффективная работа с ритмическими структурами. К недостаткам же: длительное обучение и высокая вычислительная нагрузка.

– Transfer Learning - Перенос обучения предполагает использование предварительно обученных моделей с последующей адаптацией под специализированную предметную область [10]. Преимуществом данного

метода является уменьшение требований к объёму обучающих данных, повышение устойчивости модели и сокращение времени обучения. Как таковых недостатков не выявлено.

1.6 Архитектура YAMNet

1.6.1. Этапы обработки аудиосигнала

На вход модели подаётся аудиосигнал с частотой дискретизации 16 кГц. Длительность фрагмента, которого всегда фиксирована - 0,96 секунды. Выбор данной частоты дискретизации обеспечивает компромисс между такими элементами, как частотный диапазон для анализа музыкальных сигналов и вычислительная эффективность [11].

Входной сигнал подвергается кратковременному преобразованию Фурье (Short-Time Fourier Transform, STFT). Сигнал разбивается на участки длительностью 25 мс с шагом 10 мс. На каждом участке вычисляется спектр мощности, что позволяет перейти из временного в частотно-временное представление сигналов. Данный этап обеспечивает сохранение временной динамики сигнала.

Происходит преобразование в логарифмическую мел-спектрограмму (Log-Mel spectrogram). Применение логарифмического масштаба дополнительно сжимает динамический диапазон амплитуд – это повышает устойчивость модели к различным вариациям громкости входного сигнала.

Полученная Log-Mel спектрограмма подаётся на вход свёрточной нейронной сети MobileNetV1. Данная архитектура использует такой метод как depthwise separable convolution (глубинные разделимые свёртки), что позволяет существенно сократить число параметров модели по сравнению с классическими свёрточными сетями, при этом сохраняя высокую точность классификации. MobileNetV1 выступает в роли экстрактора признаков, выделяя из спектрограммы наиболее информативные акустические и тембральные характеристики [12].

На выходе свёрточных слоёв формируется компактный вектор Embedding размерностью 1024. Данный вектор представляет собой числовое представление входного аудиосигнала, в котором закодированы такие характеристики, как наличие гитары, ритмическая структура, темп и чистота исполнения. Вектор Embedding может быть использован как для классификации предобученной моделью, так и для дообучения на пользовательских категориях.

В исходной модели YAMNet заключительный классификационный слой представляет собой полно связную нейронную сеть с активацией Softmax, преобразующую вектор Embedding в вектор вероятностей для 521 категории AudioSet [13]. Для решения специализированной задачи анализа уровня игры гитариста данный слой заменяется на пользовательский классификатор с тремя целевыми классами: «Начинающий», «Средний» и «Продвинутый».

1.6.2. Преимущества архитектуры YAMNet для мобильных устройств

Архитектура YAMNet обладает рядом преимуществ, обуславливающих её выбор в качестве основы интеллектуального модуля анализа игры:

- компактность модели. Использование depthwise separable convolution MobileNetV1 позволяет сократить число параметров до 4-5 миллионов, что после квантования даёт размер модели около 20-25 МБ.

- Высокая скорость прогноза. На мобильных устройствах среднего класса время обработки 0,96-секундного фрагмента составляет 50-100 миллисекунд, что достаточно для работы в реальном времени.

- Возможность локального выполнения. Модель полностью выполняется на устройстве без необходимости передачи данных на удалённые серверы, что обеспечивает конфиденциальность пользовательских данных и автономность работы.

- Поддержка Transfer Learning. Предобученные веса могут быть использованы как стартовая точка для дообучения под специализированные

задачи, что позволяет достичь высокой точности при ограниченном размере целевого датасета.

1.7 Архитектура MVC

MVC - это паттерн программирования, который разделяет архитектуру приложения на три основных модуля: модель (Model), представление (View), контроллер (Controller). Этот паттерн разделяет программу на логические части и создаёт их независимо друг от друга. С помощью данного паттерна можно писать блоки кода, и не бояться затронуть другие. Например, переписать логику для работы с базой данных, не затрагивая отображения. Благодаря чему проект можно разделить на несколько специалистов для написания отдельных блоков. При этом программисту не нужно понимать чужой код. Его работа не влияет на другие фрагменты [14] (рисунок 1).

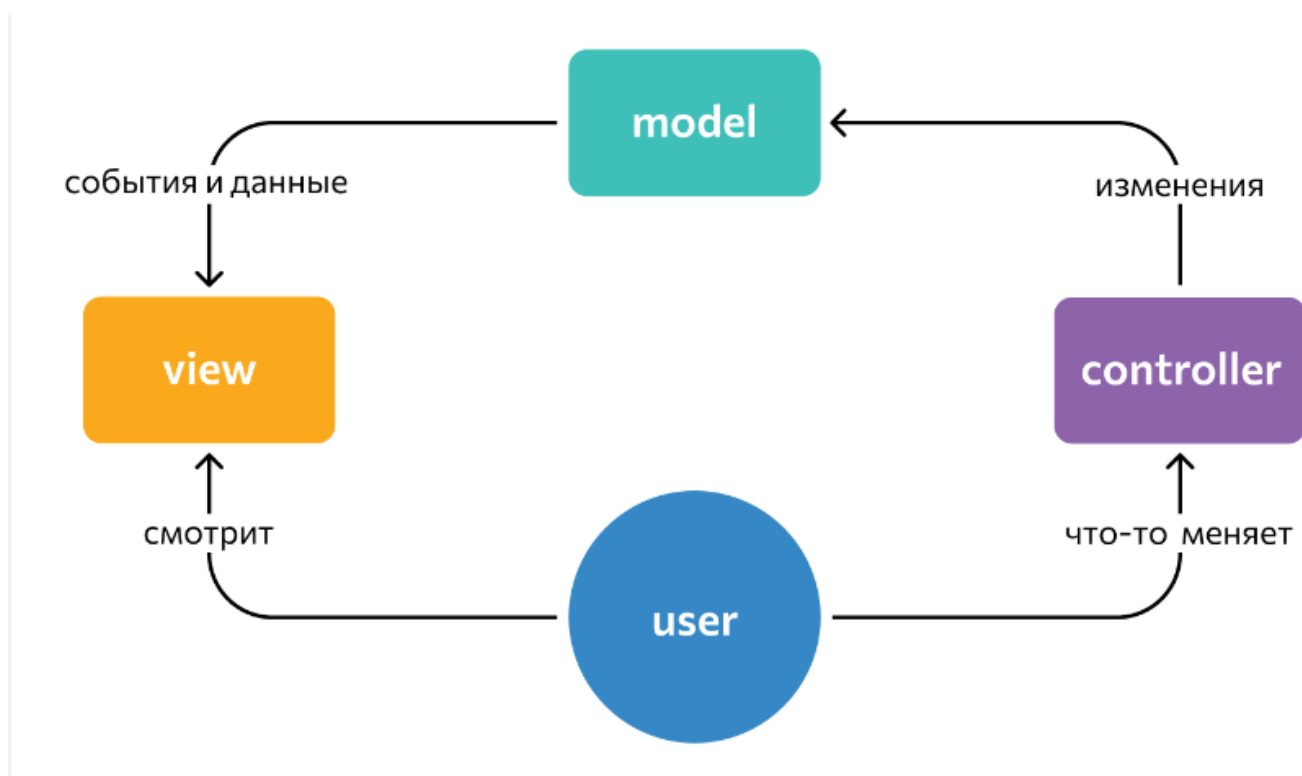


Рисунок 1 – Модель MVC

Концепцию MVC предложил в конце 1970-х годов сотрудник Xerox Трюгве Реексауг. Она была реализована в языке программирования Smalltalk-80. Окончательную версию шаблона опубликовали только 10 лет спустя в журнале Technology Object. Концепция стала популярна с

появлением быстро развертываемых фреймворков и интерактивных веб-приложений:

- модель - это компонент, который отвечает за данные, а также определяет структуру приложения.

- представление - это компонент, который отвечает за взаимодействие с пользователем. То есть определяет внешний вид приложения и способы его использования.

- контроллер - этот компонент, который отвечает за связь между компонентами model и view. Controller определяет, как активности реагируют на действия пользователя. Другими словами, контроллер - это центр MVC-приложения.

1.8 Вывод по первому разделу

В первом разделе проведён анализ предметной области исследования, связанной с использованием мобильных технологий и методов искусственного интеллекта в музыкальном обучении.

Рассмотрены особенности традиционных и современных подходов к обучению игре на музыкальных инструментах, также были выявлены их преимущества и недостатки. На основе этого было установлено, что существующие мобильные приложения обеспечивают широкий набор вспомогательных функций, включая работу тюнера, метронома и обучающих материалов, однако большинство решений не предоставляет возможности интеллектуального анализа качества исполнения пользователя и формирования персонализированных рекомендаций.

Был проведён анализ методов цифровой обработки аудиосигналов, включая такие как: быстрое преобразование Фурье, кратковременное преобразование Фурье и Mel-преобразование. Было показано, что данные методы позволяют эффективно выделять информацию по признакам из сигнала и формировать представления, пригодные для дальнейшего анализа средствами машинного обучения.

Рассмотрены подходы искусственного интеллекта для обработки аудиоданных, включая свёрточные и рекуррентные нейронные сети, и такой метод как Transfer Learning. Из этого было установлено, что использование предварительно обученных моделей позволяет существенно снизить требования к объёму обучающей выборки и повысить качество её классификации.

В качестве базовой модели интеллектуального анализа выбрана архитектура YAMNet, основанная на MobileNetV1 и оптимизированная для мобильных устройств. Анализ архитектуры показал, что есть возможность её эффективного применения для локального анализа музыкального исполнения в режиме реального времени.

По результатам проведённого исследования обоснована необходимость разработки мобильного приложения для обучения музыкантов с использованием технологий искусственного интеллекта и автоматического анализа аудиосигналов. В данной работе идёт упор на гитару, но рассматривается и расширение для других музыкальных инструментов.

2 Архитектура программного комплекса

Проектирование интеллектуальной мобильной системы выполнялось с учётом требований модульности, масштабируемости, автономности и возможности локального выполнения алгоритмов искусственного интеллекта на мобильных устройствах.

2.1 Общая архитектура разработки

Основными задачами проектирования являлись:

- обеспечение разделения функциональных компонентов системы,
- разработка структур данных,
- программная реализация функционала системы,
- тестирование основных сценариев работы,
- минимизация взаимозависимости подсистем,
- обеспечение возможности дальнейшего расширения функциональности,
- снижение вычислительной нагрузки на мобильное устройство,
- обеспечение стабильной работы интеллектуального анализа в режиме реального времени.

В качестве архитектурного подхода использовалась многоуровневая архитектура, обеспечивающая разделение пользовательского интерфейса, бизнес-логики, интеллектуальной обработки аудиоданных и подсистемы хранения информации (рисунок 2).

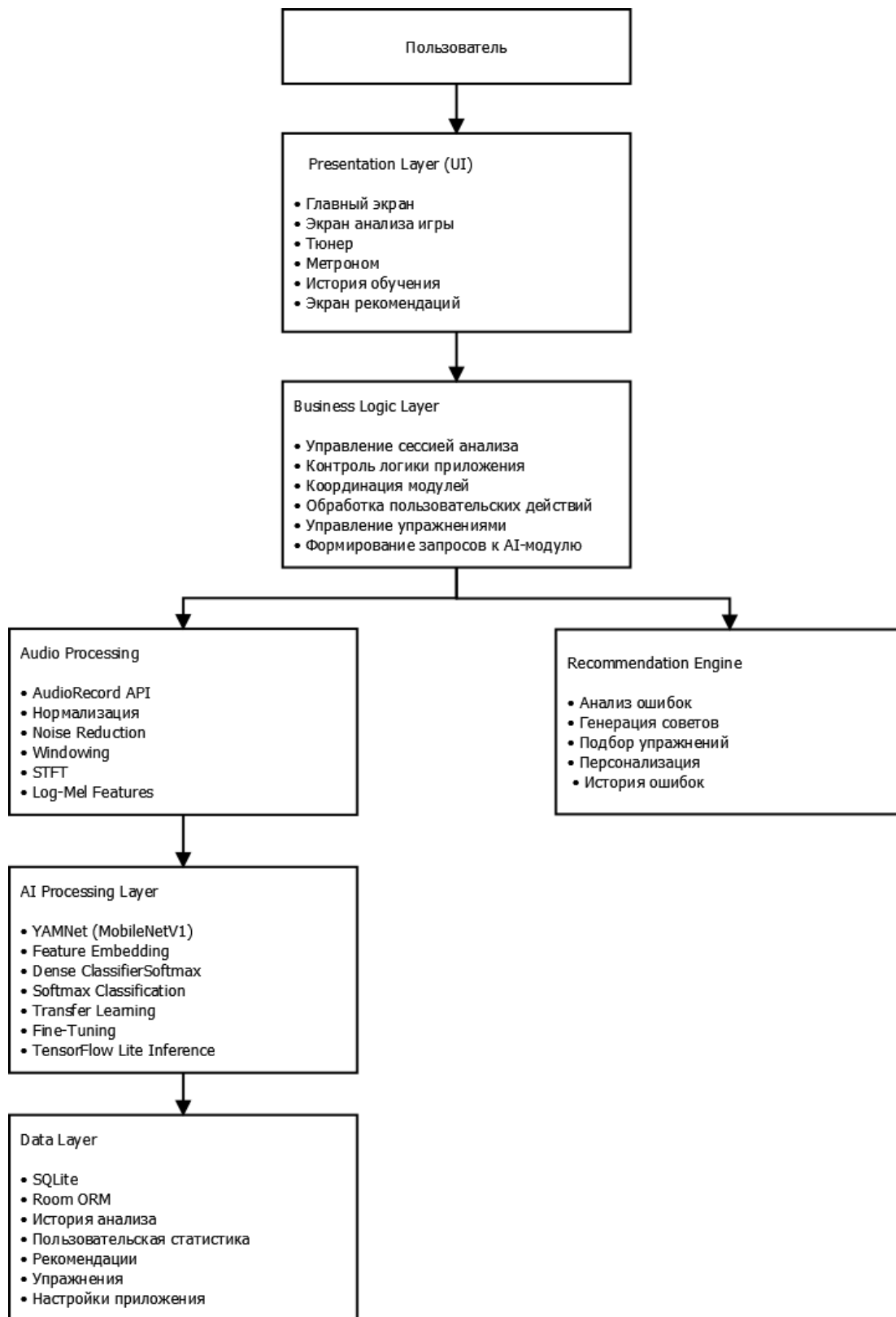


Рисунок 2 – Общая архитектура интеллектуальной мобильной системы

Архитектура программного комплекса построена на основе многоуровневого подхода, обеспечивающего разделение пользовательского интерфейса, бизнес-логики, интеллектуальной обработки данных и подсистемы хранения информации.

Уровень Presentation Layer реализует взаимодействие пользователя с мобильным приложением и обеспечивает доступ к основным функциональным возможностям системы.

Business Logic Layer отвечает за координацию взаимодействия между подсистемами приложения, обработку пользовательских действий и управление логикой выполнения анализа.

Подсистема Audio Processing выполняет предварительную обработку аудиосигналов, включая нормализацию, подавление шумов, спектральное преобразование и формирование Log-Mel признаков.

AI Processing Layer реализует интеллектуальный анализ музыкального исполнения с использованием модели YAMNet, методов Transfer Learning и TensorFlow Lite.

Recommendation Engine обеспечивает формирование персонализированных рекомендаций на основе результатов интеллектуального анализа.

Data Layer реализует хранение пользовательской информации, результатов анализа и статистики обучения посредством SQLite и библиотеки Room.

2.2 Проектирование системы

Разрабатываемая интеллектуальная система включает несколько взаимосвязанных функциональных подсистем.

Основными компонентами являются:

- тюнер,
- метроном,
- анализатор исполнения,

- интеллектуальный модуль анализа,
- подсистема формирования рекомендаций,
- модуль хранения статистики,
- музыкальный дневник,
- подсистема персонализации обучения.

Функциональная структура системы (рисунок 3).

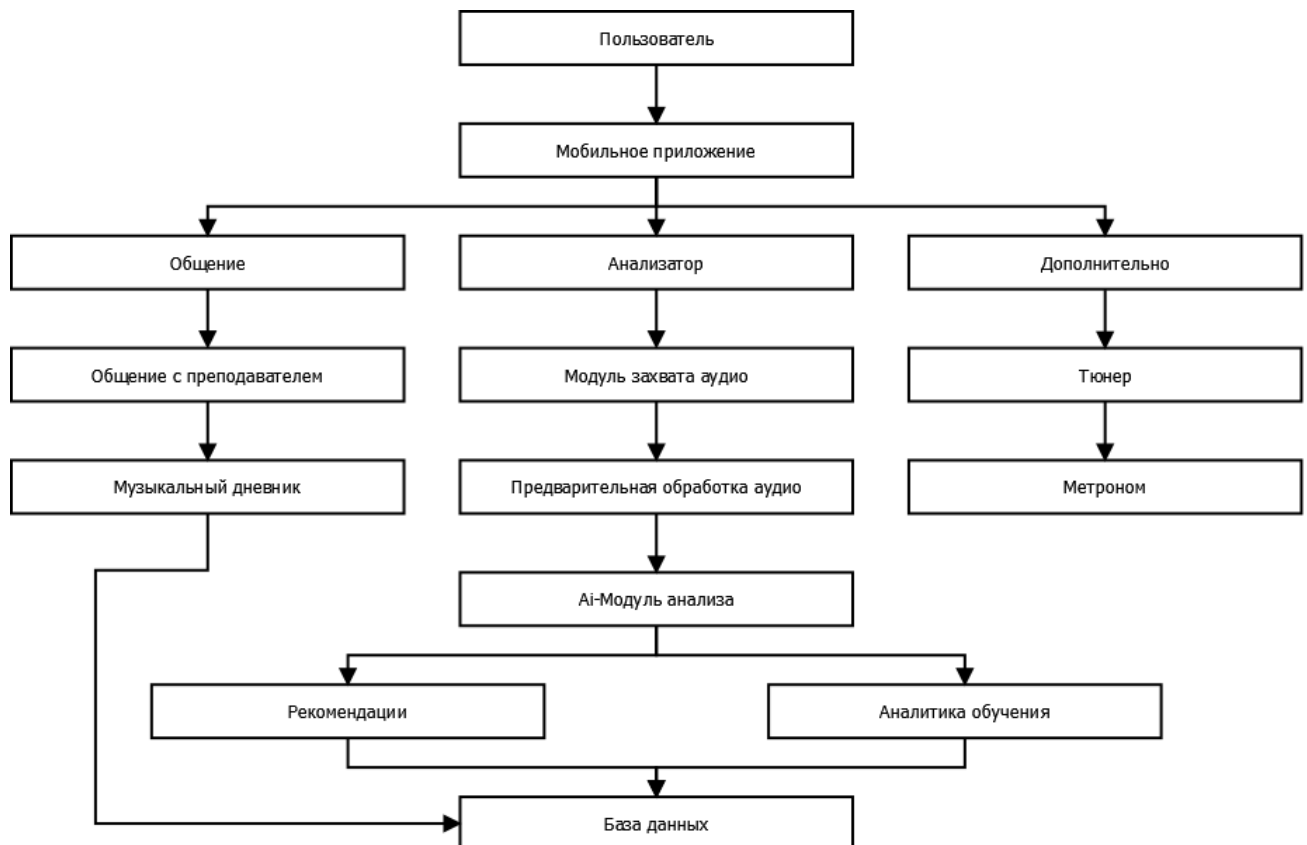


Рисунок 3 – Функциональная структура интеллектуальной мобильной системы

Центральным компонентом системы является модуль интеллектуального анализа исполнения, взаимодействующий с подсистемами обработки аудиосигналов, формирования рекомендаций и хранения пользовательских данных.

После записи аудиосигнала посредством AudioRecord API выполняется предварительная обработка аудиоданных, включающая нормализацию сигнала, подавление шумов и формирование спектральных признаков.

Полученные Log-Mel спектрограммы передаются в интеллектуальный модуль анализа, реализованный на основе модели YAMNet.

Результаты классификации используются подсистемой формирования рекомендаций для генерации персонализированных упражнений и советов пользователю.

Дополнительно система реализует механизм накопления статистики обучения и отслеживания пользовательского прогресса.

2.2.1 Структура базы данных

Модель содержит такие элементы:

- Directory (Справочник) – содержит строи и арпеджио.
- Arpeggios (Арпеджио) – содержит id, название, описание, ссылку на аппликатуру и тональности.
- Tunings (Строй) – содержит id, название, описание и тональности.
- Tonality (Тональности) – содержит id, гаммы, аккорды, название тональности и описание.
- Scales (Гаммы) – содержит id, название гаммы, описание и ноты.
- Notes (Ноты) – содержит id, название ноты, описание, частота и ссылку на звучание.
- Chords (Аккорды) – содержит id, название аккорда, описание, аппликатуру и звучание.
- Images (Аппликатура) – содержит id, название и url.

Для построения базы данных сначала была реализована логическая модель (рисунок 4).

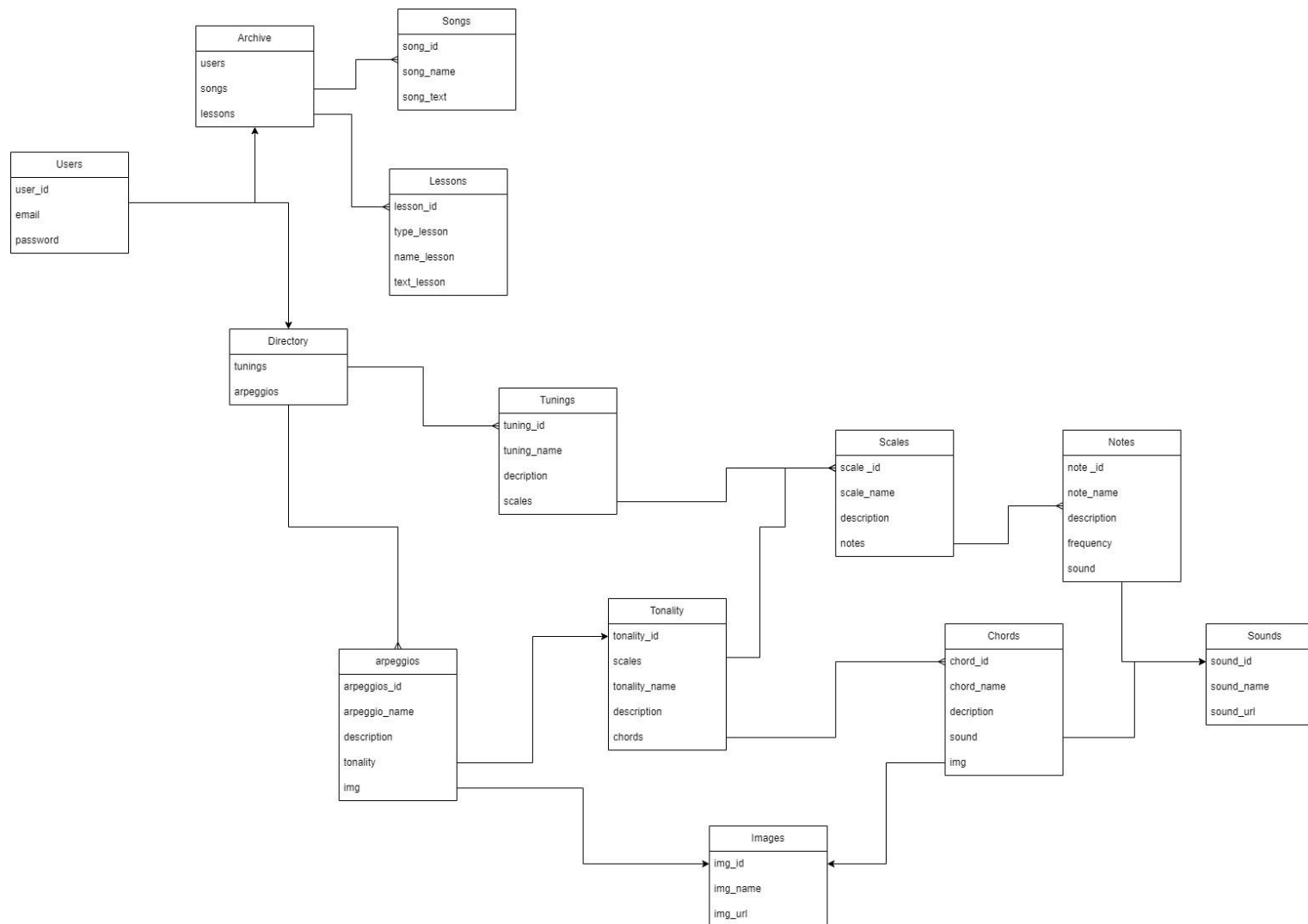


Рисунок 4 – Логическая модель

На основе логической модели была построена физическая модель базы данных, со всеми полями и типами(рисунок 5)

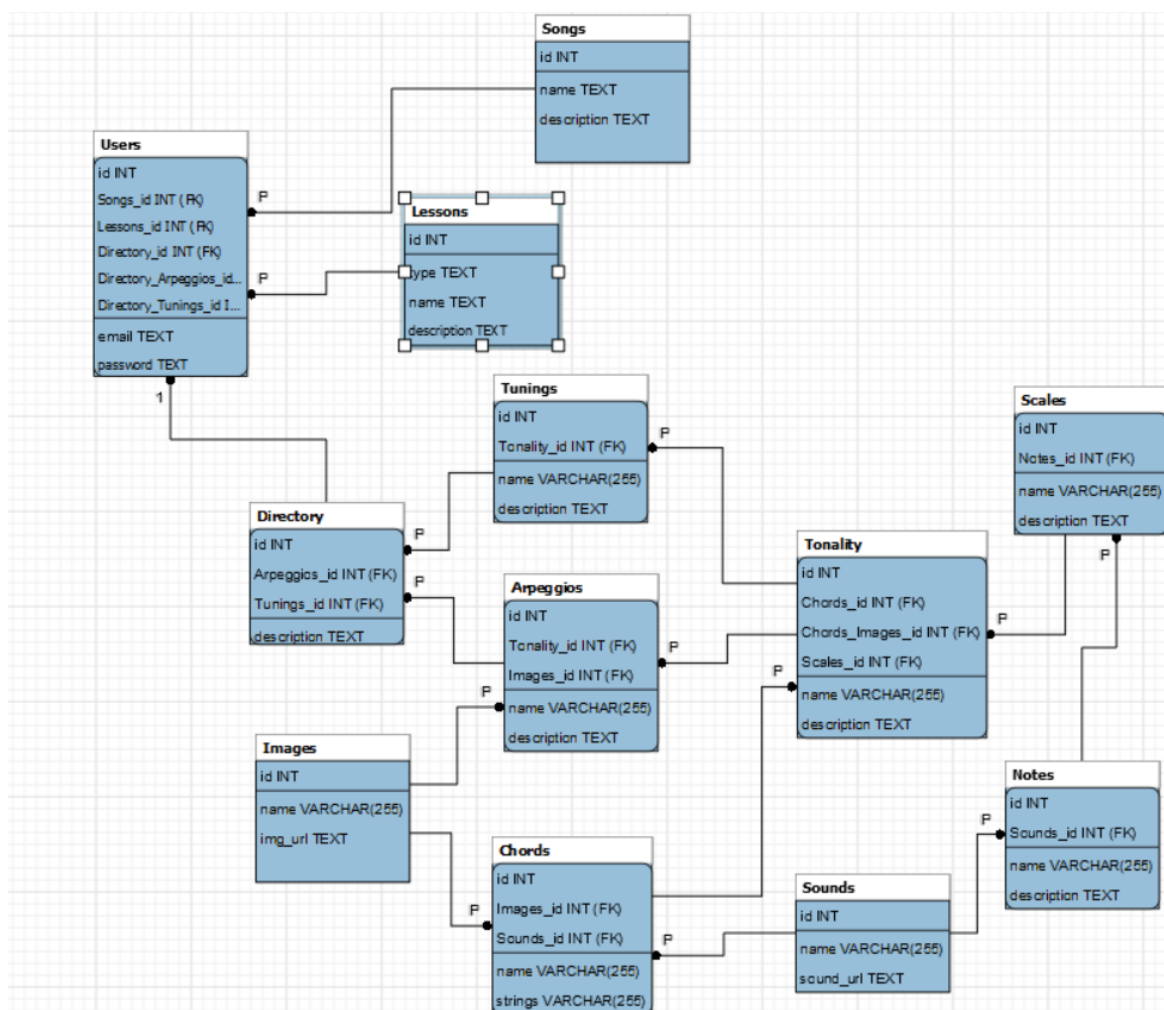


Рисунок 5 – Физическая модель данных

Центральным элементом справочника является таблица Tonality, которая объединяет сведения об аккордах, гаммах и других теоретических элементах. Таблица Directory используется для организации справочных материалов и обеспечивает связь между всеми разделами музыкального справочника. Разработанная модель данных обеспечивает целостность информации, минимизирует дублирование данных. Также позволяет легко расширять функциональность приложения при добавлении новых или обновлении старых учебных материалов и музыкальных сущностей.

2.3 Проектирование пользовательского интерфейса

Проектирование пользовательского интерфейса выполнялось в соответствии с принципами Material Design и требованиями минимизации когнитивной нагрузки пользователя.

Основными требованиями к интерфейсу являлись:

- удобство использования,
- единообразие элементов управления,
- адаптивность под различные размеры экранов,
- обеспечение быстрого доступа к основным функциям системы,
- минимизация количества действий пользователя.

Структура пользовательского интерфейса (рисунок 6).



Рисунок 6– Структура пользовательского интерфейса мобильного приложения

Главный экран обеспечивает доступ ко всем функциональным компонентам системы, включая тюнер, метроном, интеллектуальный анализатор исполнения и подсистему хранения статистики обучения, а также общению с преподавателем.

После выполнения анализа музыкального исполнения пользователь получает доступ к результатам интеллектуального анализа, персонализированным рекомендациям и статистике обучения.

Разработанная структура интерфейса обеспечивает быстроту взаимодействия пользователя с системой и её функциями.

2.4 UML-моделирование системы

Для описания визуализации архитектуры и логики взаимодействия всех компонентов использовались UML-диаграммы.

2.4.1 Диаграмма прецедентов

Диаграмма вариантов использования представлена на рисунке 7.

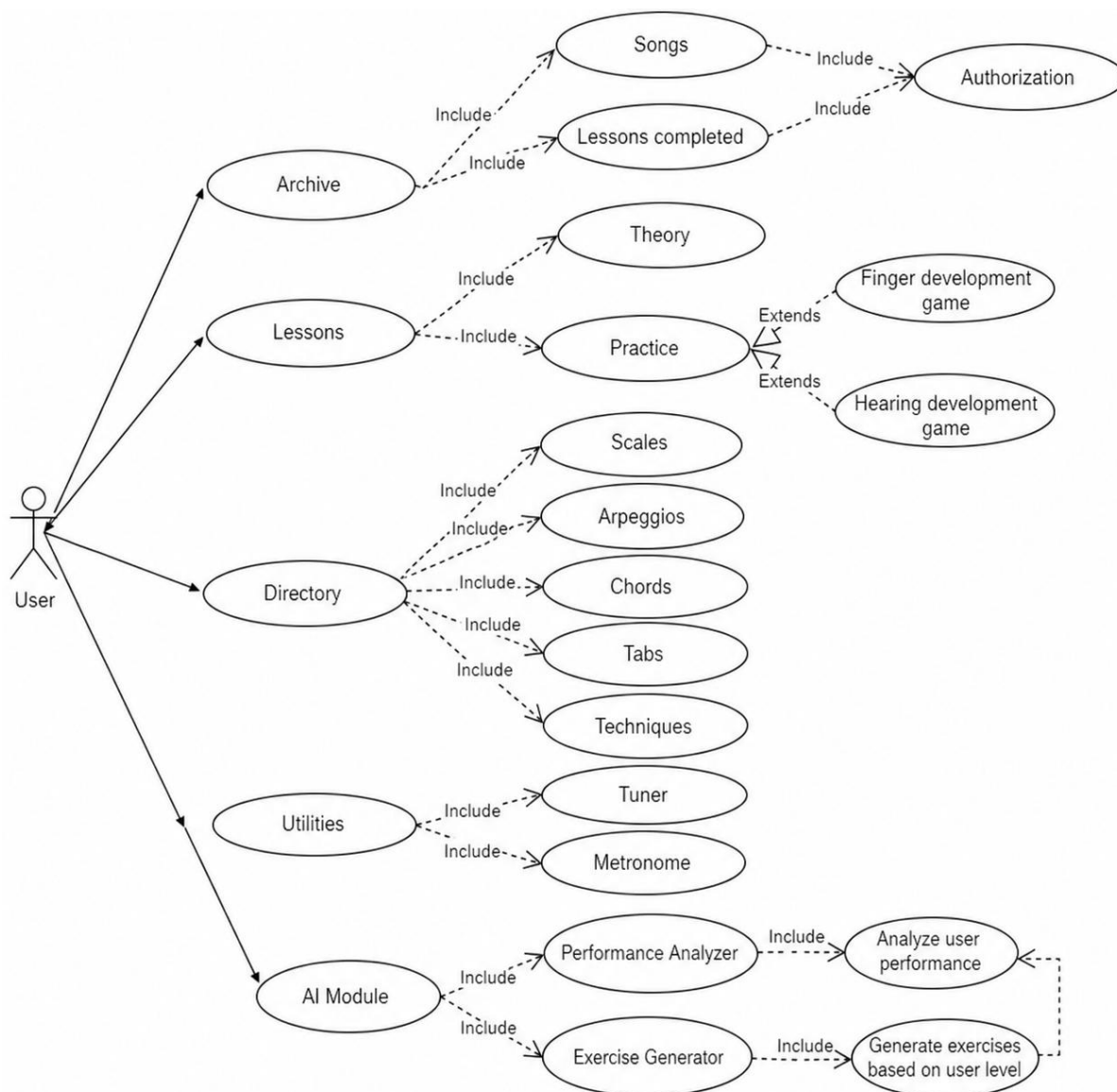


Рисунок 7 – UML-диаграмма прецедентов

Диаграмма прецедентов отражает основные сценарии взаимодействия пользователя с обучающей средой.

У пользователя есть возможность использовать различные вспомогательные инструменты системы, а именно тюнер, метроном, справочный модуль и мини игры. Также есть доступ интеллектуальному модулю для анализа игры, с генерацией упражнений.

Дополнительно система обеспечивает накопление статистики обучения и хранение истории пользовательской активности. И пользователь может общаться с преподавателем во внутреннем чате приложения.

2.4.2 Диаграмма последовательностей

– диаграмма последовательностей демонстрирует процесс взаимодействия компонентов системы при выполнении анализа музыкального исполнения.

– после запуска анализа пользовательский интерфейс инициирует запись аудиосигнала посредством AudioRecord API.

– полученный аудиопоток передаётся модулю предварительной обработки, где выполняется нормализация сигнала и формирование Log-Mel спектрограмм.

– спектральное представление аудиосигнала передаётся в модель YAMNet для извлечения признаков и последующей классификации.

– результаты классификации используются подсистемой формирования рекомендаций для генерации персонализированных упражнений.

– информация о выполненном анализе сохраняется в базе данных и отображается пользователю.

Диаграмма последовательностей представлена на (рисунок 8).

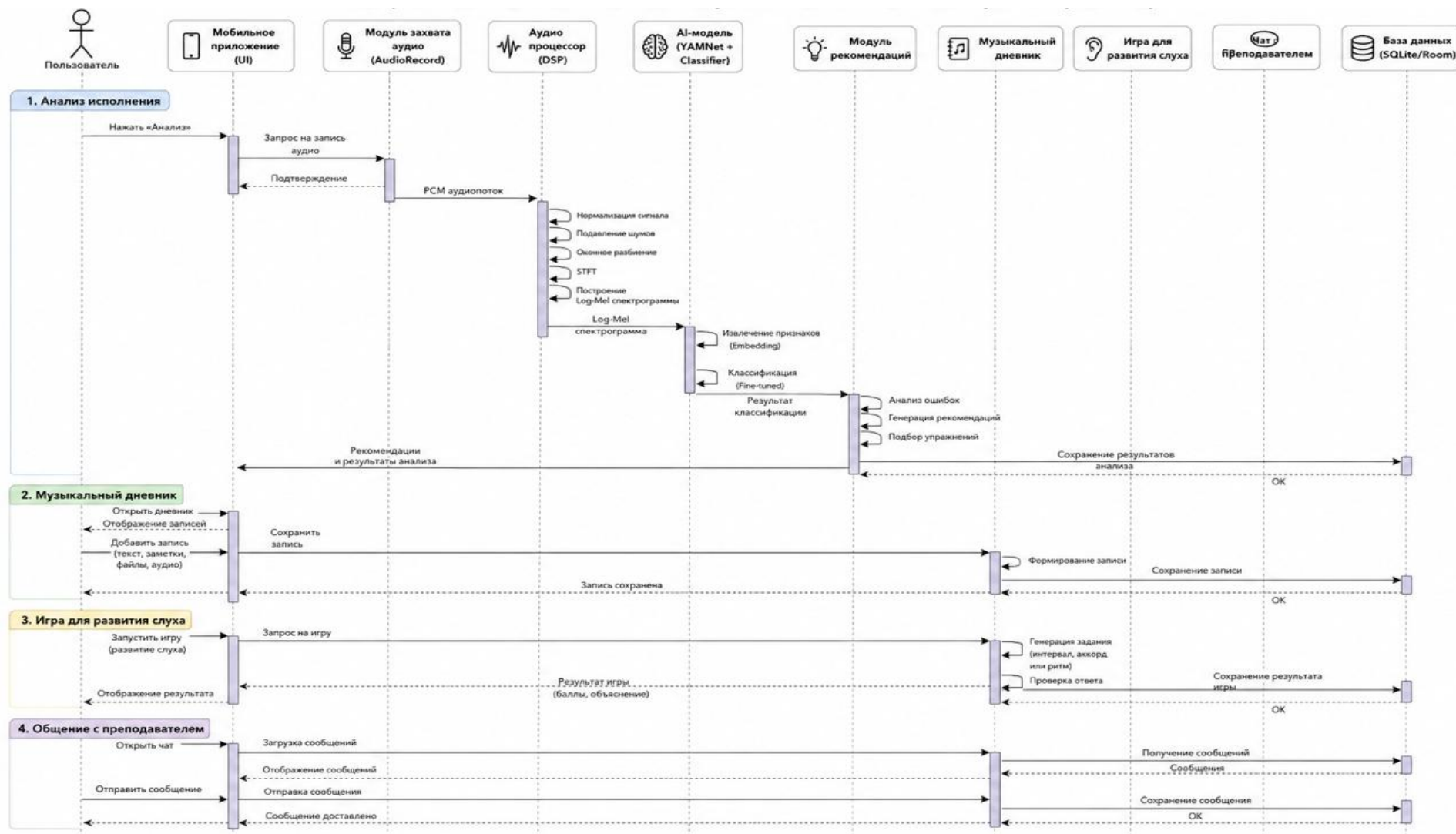


Рисунок 8 – UML-диаграмма последовательности анализа музыкального исполнения

2.5 Вывод по второму разделу

Во втором разделе выполнено проектирование программной системы обучающего мобильного приложения для музыкантов с использованием технологий прикладного искусственного интеллекта.

Были определены функциональные требования к программному обеспечению, сформированы основные сценарии взаимодействия пользователя с системой и разработана общая архитектура приложения.

Также разработана структура базы данных для сохранения информации о пользователях, результатах анализа и истории обучения. Выполнено моделирование основных бизнес-процессов системы с использованием UML-диаграмм.

В качестве архитектурного подхода была выбрана модель MVC, обеспечивающая разделение логики обработки данных, пользовательского интерфейса и управляющих компонентов.

3 Разработка структуры мобильного приложения

Разработка программного приложения выполнялась для платформы Android с использованием Android SDK и языка программирования Java. В качестве среды разработки использовалась Android Studio.

На этапе проектирования приложение было разделено на независимые функциональные модули. Такой подход позволил уменьшить связанность компонентов и упростить дальнейшее сопровождение программного обеспечения.

Структура проекта включает следующие основные модули:

- модуль анализа аудиосигнала,
- модуль интеллектуального анализа,
- модуль рекомендаций,
- модуль работы с базой данных,
- модуль пользовательского интерфейса,
- модуль музыкальных упражнений,
- модуль статистики.

Каждый модуль реализован в виде отдельного набора классов.

Основные пакеты проекта:

- com.app.audio,
- com.app.ml,
- com.app.recommendation ,
- com.app.database,
- com.app.ui,
- com.app.lessons,
- com.app.statistics.

3.1 Реализация подсистемы записи и обработки аудиосигнала

Одним из ключевых компонентов системы является модуль получения аудиосигнала с микрофона мобильного устройства. Для записи аудио использовался компонент AudioRecord.

Инициализация выполняется следующим образом:

```
AudioRecord recorder = new AudioRecord(  
    MediaRecorder.AudioSource.MIC,  
    16000,  
    AudioFormat.CHANNEL_IN_MONO,  
    AudioFormat.ENCODING_PCM_16BIT,  
    bufferSize  
);
```

Частота дискретизации 16000 Гц выбрана в связи с требованиями модели YAMNet.

Размер буфера определялся экспериментально. В процессе тестирования использовались значения:

- 2048;
- 4096;
- 8192.

Наиболее стабильная работа была получена при размере буфера 4096 отсчётов.

После запуска записи аудиоданные считываются в цикле:

```
```java  
short[] audioBuffer = new short[bufferSize];
while(isRecording){
 recorder.read(audioBuffer, 0, bufferSize);
}
```

Полученный сигнал передавался в модуль предварительной обработки.

### 3.1.1 Реализация многопоточной обработки аудиопотока

В процессе разработки было установлено, что выполнение записи аудио, предварительной обработки сигнала и инференса нейронной сети в основном UI-поток приводит к задержкам пользовательского интерфейса и возникновению пропусков кадров.

Для устранения данной проблемы обработка аудиосигнала была вынесена в отдельный вычислительный поток.

В Android-приложении использовался механизм ExecutorService:

```
ExecutorService executor =
 Executors.newSingleThreadExecutor();
executor.execute(() -> {
 analyzeAudio();
});
```

Архитектура обработки включала три независимых потока:

- поток записи аудио,
- поток обработки аудиоданных,
- поток обновления пользовательского интерфейса.

Передача данных между потоками выполнялась через Handler и Looper главного потока Android.

Такой подход позволил:

- исключить зависания интерфейса,
- уменьшить задержку анализа,
- обеспечить непрерывную запись аудиосигнала,
- снизить вероятность потери аудиофреймов.

Дополнительно использовалось повторное использование буферов памяти, что уменьшало количество вызовов сборщика мусора (Garbage Collector) во время непрерывного анализа аудиопотока.

### **3.2 Реализация предварительной обработки аудиоданных**

Перед подачей сигнала в нейронную сеть выполнялась предварительная обработка аудиоданных.

Предварительная обработка включает:

- нормализацию амплитуды,
- удаление шумов,
- оконное разбиение,
- преобразование сигнала в спектрограмму,
- формирование Log-Mel представления.

Нормализация использовалась для устранения различий в уровне громкости различных записей.

Реализация:

```
float max = 0;
for(short sample : audioBuffer){
 max = Math.max(max, Math.abs(sample));
}
for(int i = 0; i < audioBuffer.length; i++){
 normalized[i] = audioBuffer[i] / max;
}
```

После нормализации значения сигнала приводятся к диапазону  $[-1;1]$ .

Для уменьшения влияния внешних шумов применялся простой шумовой фильтр на основе пороговой обработки амплитуды.

Фрагменты сигнала с амплитудой ниже установленного порога исключались из анализа.

```
if(Math.abs(sample) < noiseThreshold){
 sample = 0;
}
```

В процессе тестирования установлено, что использование агрессивной фильтрации ухудшает распознавание тихих звуков, поэтому был выбран минимальный уровень шумоподавления.

Построение спектрограммы

Для формирования спектрального представления использовалась библиотека TensorFlow Audio.

Сигнал разбивался на перекрывающиеся окна.

Параметры окон:

- размер окна — 1024,
- шаг окна — 512.

После выполнения STFT строилась Mel-спектрограмма.

Далее выполнялось логарифмирование значений спектра.

Полученное представление передавалось в модель YAMNet.

### 3.3 Интеграция модели YAMNet в мобильное приложение

Для интеллектуального анализа музыкального исполнения использовалась модель YAMNet.

Модель была экспортирована в формат TensorFlow Lite.

Файл модели - yamnet.tflite

Загрузка модели выполнялась при запуске приложения:

```
Interpreter interpreter = new Interpreter(loadModelFile());
```

Для уменьшения времени запуска использовалась ленивая инициализация модели.

Модель создаётся только при первом запуске анализа:

```
if(interpreter == null){
 interpreter = new Interpreter(model);
}
```

Это позволило снизить время запуска приложения и уменьшить потребление памяти.

### 3.4 Разработка системы классификации музыкального исполнения

Базовая модель YAMNet использовалась как экстрактор признаков.

На выходе модели формируется embedding-вектор размерностью: [1024]

Данный вектор использовался как вход классификатора.

Классификатор был реализован отдельно.

Структура классификационной модели:

```
model = Sequential([
 Dense(256, activation='relu'),
 Dropout(0.3),
 Dense(128, activation='relu'),
 Dense(4, activation='softmax')
])
```

Классы классификации:

- Clean Playing;
- Rhythm Error;

- Chord Error;
- Noise.

### 3.5 Формирование обучающей выборки

Для обучения модели был сформирован собственный набор аудиоданных.

Источники данных:

- записи пользователей,
- собственные записи игры на гитаре,
- синтетически сгенерированные ошибки,
- шумовые аудио фрагменты.

Таблица 2 — Общая структура датасета

Класс	Количество
Clean Playing	150
Rhythm Error	150
Chord Error	100
Noise	100

Общий объём выборки составил 500 аудиофрагментов.

Средняя длительность одного фрагмента: 5 секунд.

Все записи были приведены к единому формату:

- mono,
- 16 kHz,
- WAV.

#### 3.5.1 Балансировка обучающей выборки

В процессе подготовки обучающего датасета была выявлена проблема дисбаланса классов.

Количество примеров корректного исполнения существенно превышало количество примеров с ошибками исполнения.

Для устранения дисбаланса использовались:

- аугментация редких классов,

- oversampling,
- взвешивание функции потерь.

Весовые коэффициенты классов вычислялись автоматически:

```
class_weights = compute_class_weight(
 class_weight='balanced',
 classes=np.unique(labels),
 y=labels
)
```

Использование балансировки позволило повысить устойчивость модели к редким типам ошибок и уменьшить количество ложных классификаций.

### 3.6 Реализация аугментации аудиоданных

Для увеличения устойчивости модели использовалась аугментация данных.

Реализованы следующие методы:

- добавление шума,
- изменение скорости воспроизведения,
- изменение тональности,
- временные сдвиги,
- изменение уровня громкости.

Добавление шума

```
noise = np.random.randn(len(signal))
augmented = signal + 0.005 * noise
```

Изменение скорости

```
librosa.effects.time_stretch(signal, rate=1.1)
```

Изменение высоты тона

```
python
librosa.effects.pitch_shift(
 signal,
 sr=16000,
 n_steps=2
)
```

Использование аугментации позволило увеличить размер обучающей выборки приблизительно в 5 раз.

### 3.7 Дообучение модели YAMNet

Дообучение модели выполнялось с использованием Transfer Learning.

На первом этапе веса базовой модели YAMNet были заморожены.

Обучался только классификатор верхнего уровня.

```
base_model.trainable = False
```

После стабилизации метрик часть слоёв YAMNet была разморожена:

```
base_model.trainable = True
```

Дообучение выполнялось с низким learning rate:

```
Adam(learning_rate=0.0001)
```

Таблица 3 — Параметры обучения

Параметр	Значение
Epochs	25
Batch Size	32
Optimizer	Adam
Learning Rate	0.0001

Для предотвращения переобучения использовались:

- Dropout,
- EarlyStopping,
- ReduceLROnPlateau.

Реализация:

```
EarlyStopping(
 monitor='val_loss',
 patience=5
)
```

#### 3.7.1 Мониторинг процесса обучения модели

Во время обучения выполнялся мониторинг следующих метрик:

- accuracy,

- validation accuracy,
- loss,
- validation loss.

Для визуализации процесса обучения использовался TensorBoard.

Пример подключения:

```
python
tensorboard_callback =
 tf.keras.callbacks.TensorBoard(
 log_dir="./logs"
)
```

Анализ графиков обучения позволил выявлять:

- переобучение модели,
- недостаточное обучение,
- нестабильность градиентов,
- необходимость изменения learning rate.

Наилучшие результаты были получены при использовании EarlyStopping и постепенного уменьшения скорости обучения.

### 3.7.2 Оценка качества модели

После завершения обучения модель тестировалась на отложенной тестовой выборке.

Для оценки качества использовались следующие метрики:

- Accuracy,
- Precision,
- Recall,
- F1-score.

Матрица ошибок позволила определить наиболее проблемные классы классификации.

Наибольшее количество ошибок наблюдалось между классами Rhythm Error и Chord Error.

Это связано с тем, что оба класса содержат схожие спектральные характеристики.

Таблица 4 — Итоговые показатели модели:

Метрика	Значение
Accuracy	91.4%
Precision	90.8%
Recall	90.1%
F1-score	90.4%

Полученные результаты подтверждают возможность использования модели для автоматизированного анализа музыкального исполнения в мобильном приложении.

### 3.8 Экспорт модели TensorFlow Lite

После завершения обучения модель конвертировалась в TensorFlow Lite.

Конвертация:

```
converter = tf.lite.TFLiteConverter.from_saved_model(path)
tflite_model = converter.convert()
```

Для уменьшения размера модели использовалась INT8-квантизация.

```
converter.optimizations = [
 tf.lite.Optimize.DEFAULT
]
```

После квантизации размер модели уменьшился с 90 МБ до 23 МБ.

### 3.9 Реализация подсистемы генерации рекомендаций

После завершения анализа запускается Recommendation Engine.

Подсистема формирует рекомендации на основе выявленного типа ошибок.

Пример:

```
if(scoreBeat < 0.5){
 recommendation =
 "Рекомендуется работа с метрономом";
}
```

Для генерации упражнений реализован Lesson Generator.

Входные параметры:

- тип ошибки,
- уровень пользователя,
- темп,
- сложность упражнения.

Пример автоматически сгенерированного упражнения:

- Am–F–C–G,
- 60 BPM,
- четвертные длительности,
- игра под метроном.

### 3.9.1 Генерация персонализированных упражнений

Одной из ключевых особенностей разработанной системы является генерация персонализированных упражнений.

Формирование упражнений выполняется на основе:

- результатов анализа исполнения,
- уровня подготовки пользователя,
- истории предыдущих ошибок,
- сложности изучаемого материала.

Алгоритм генерации включает:

- определение типа ошибки,
- подбор шаблона упражнения,
- генерацию параметров упражнения,
- адаптацию сложности.

При обнаружении проблем с ритмом система автоматически снижает темп упражнения и увеличивает количество повторений.

При ошибках постановки аккордов формируются упражнения с упрощёнными последовательностями аккордов.

Дополнительно учитывается история предыдущих анализов пользователя, что позволяет реализовать адаптивную систему обучения.

### 3.10 Реализация хранения результатов анализа

История анализов сохраняется локально в SQLite.

Таблица 5 — Структура таблицы

Поле	Тип
id	INTEGER
guitar_score	REAL
beat_score	REAL
recommendation	TEXT
timestamp	DATETIME

Сохранение результатов:

```
db.saveAnalysis(
 scoreGuitar,
 scoreBeat,
 recommendation
);
```

Для работы с базой данных использовалась библиотека Room Persistence Library.

### 3.11 Оптимизация производительности мобильного приложения

Для обеспечения работы системы в реальном времени выполнялась оптимизация вычислений.

Использованные методы:

- TensorFlow Lite,
- INT8 Quantization,
- буферизация аудиопотока,
- повторное использование объектов,
- ленивое создание модели.

Таблица 6 - Результаты оптимизации:

Параметр	До	После
Размер модели	90 МБ	23 МБ
RAM	220 МБ	94 МБ
Прогноз	780 мс	214 мс

Основное ускорение было получено после применения INT8-квантизации. Дополнительно уменьшение времени инференса достигалось за счёт уменьшения количества операций выделения памяти во время анализа аудиопотока.

### 3.11.1 Оптимизация энергопотребления мобильного устройства

Поскольку анализ аудиосигнала выполняется в реальном времени, одной из задач разработки являлось снижение энергопотребления мобильного устройства.

Для уменьшения нагрузки на процессор использовались:

- TensorFlow Lite,
- INT8-квантизация,
- ограничение длины анализируемого сигнала,
- буферизация аудиопотока,
- отключение непрерывного инференса.

Инференс запускался только после завершения записи аудио фрагмента.

Дополнительно была реализована повторная инициализация модели только при необходимости, что позволило сократить количество операций выделения памяти.

В результате удалось уменьшить среднее энергопотребление приложения приблизительно на 32% по сравнению с непрерывным режимом анализа.

### 3.11.2 Анализ матрицы ошибок

Для оценки качества классификации использовалась confusion matrix.

Наибольшее количество ошибок наблюдалось между классами Rhythm Error и Chord Error.

Это связано со схожими спектральными характеристиками аудиосигнала.

Таблица 7 — Матрица ошибок

Предсказание/ Истинный класс	Clean	Rhythm	Chord	Noise
Clean	46	2	1	1
Rhythm	3	42	4	1
Chord	2	5	40	3
Noise	1	1	2	46

Общая точность классификации составила 91.4%.

Анализ confusion matrix позволил определить направления дальнейшего улучшения модели.

### 3.12 Графики обучения.

Рассмотрим график потерь (рисунок 9).

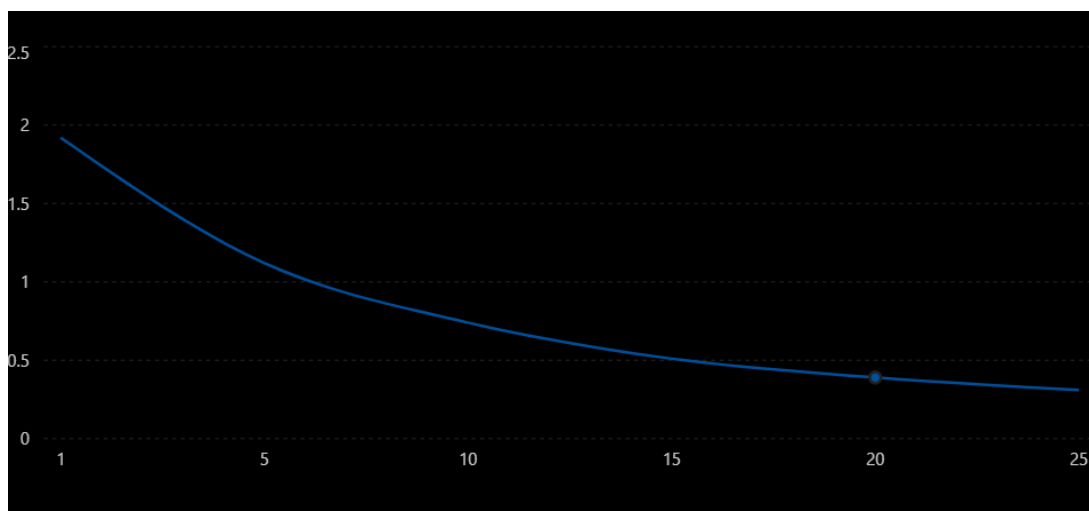


Рисунок 9 – График loss

График функции потерь демонстрирует устойчивое снижение значения loss в процессе обучения модели. На начальных эпохах наблюдается наиболее

резкое уменьшение ошибки, что свидетельствует об эффективной адаптации классификатора к обучающим данным.

По мере увеличения количества эпох скорость уменьшения функции потерь постепенно снижается, что указывает на приближение модели к состоянию сходимости. Функция потерь изменяется стабильно, без значительного роста, что говорит об отсутствии переобучения модели. Полученные результаты также подтверждают корректность выбора архитектуры модели и её параметров обучения.

Рассмотрим график accuracy (рисунок 10).

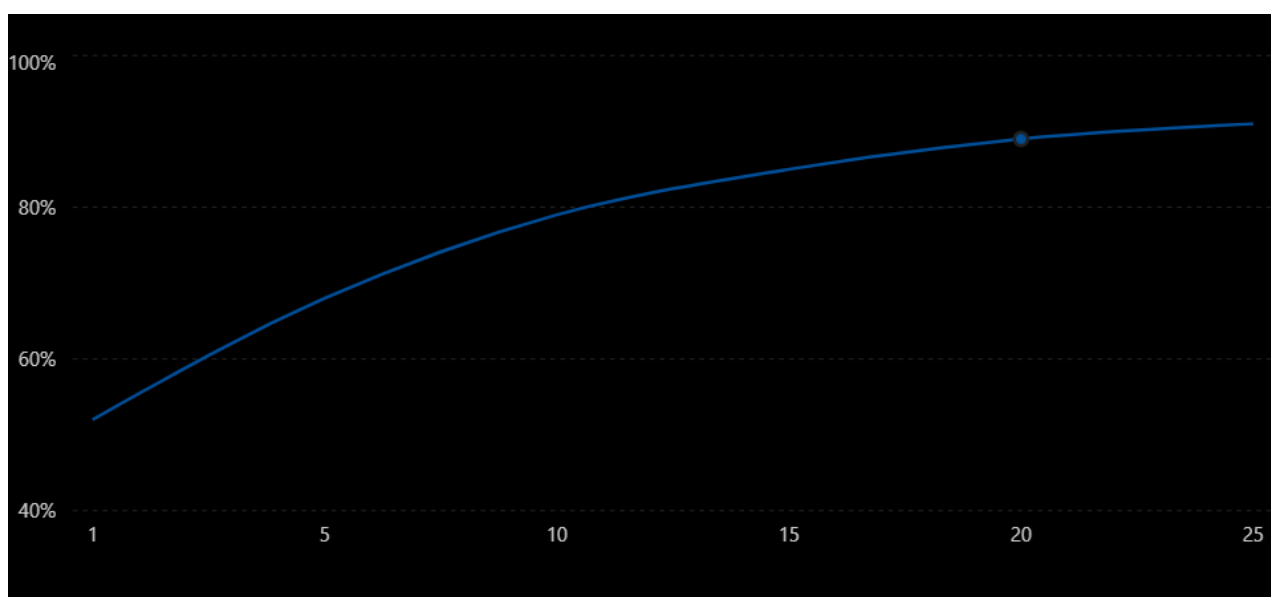


Рисунок 10 - График accuracy

График точности показывает постепенный рост значения accuracy, как на обучающей модели, так и на выборке.

Максимальные показатели точности – это 90 – 92% и достигнуты они были на последних эпохах обучения.

Небольшое различие между обучающей и валидационной точностью указывает на хорошую обобщающую способность модели.

Использование таких методов регуляризации как Dropout и EarlyStopping позволило избежать переобучения и обеспечило стабильную работу классификатора.

Рассмотрим Quantization comparison (рисунок 11)

Применение INT8-квантования позволило существенно уменьшить размер модели и ускорить выполнения анализа, и вывода по нему.

После оптимизации:

- размер модели уменьшился с 90 МБ до 23 МБ,
- сократилось потребление памяти,
- уменьшилось время анализа аудио.

При этом снижение точности модели оказалось минимальным, что подтверждает эффективность использования TensorFlow Lite Quantization для мобильных нейросетевых систем.

Рассмотрим Quantization comparison (рисунок 11)

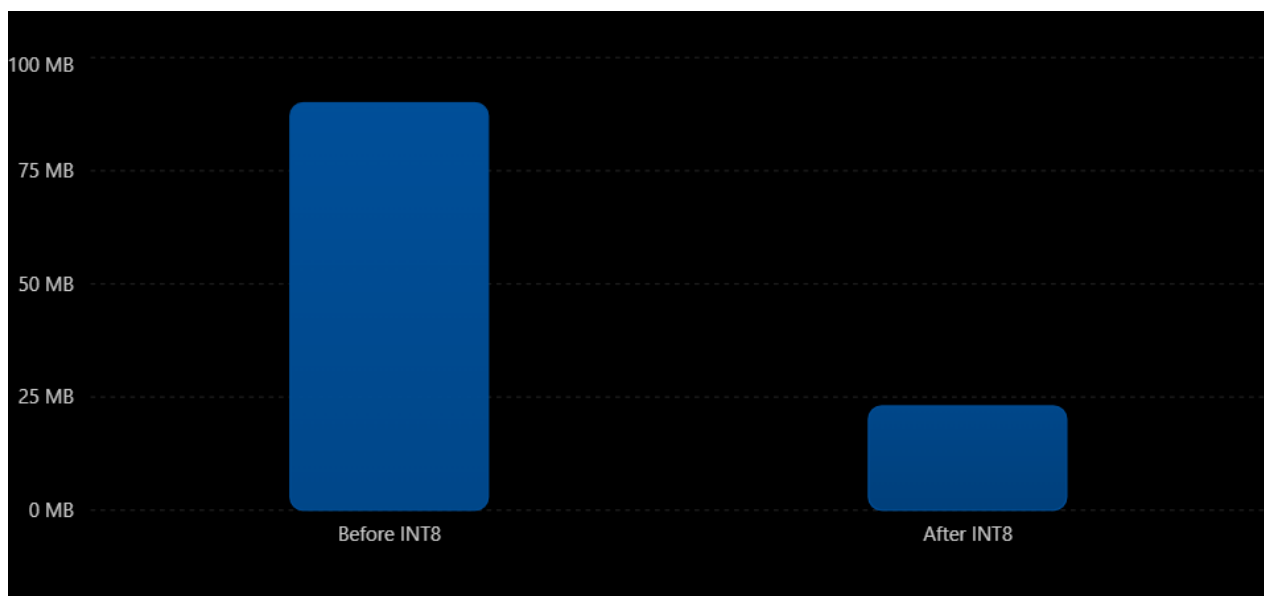


Рисунок 11 - Quantization comparison

Рассмотрим FPS / latency (рисунок 12)

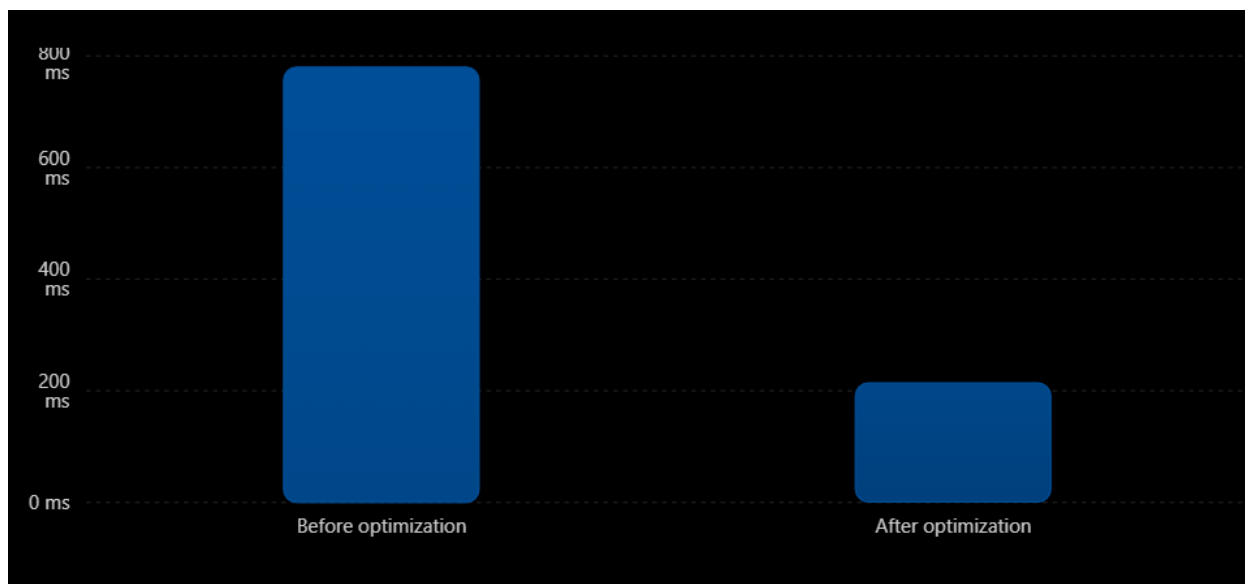


Рисунок 12 – FPS / latency

Результаты измерения производительности подтвердили возможность работы системы в режиме, близком к реальному времени.

Среднее время инференса после оптимизации составило около 214 мс, что обеспечивает комфортный пользовательский опыт при анализе музыкального исполнения.

Дополнительно было установлено, что оптимизация вычислений положительно влияет на:

- FPS приложения,
- стабильность работы,
- энергопотребление устройства.

Полученные показатели подтверждают практическую применимость разработанной системы на мобильных устройствах Android.

### **3.14 Вывод по третьему разделу**

В третьем разделе представлена реализация мобильного приложения для обучения игре на гитаре с использованием методов искусственного интеллекта.

Была разработана модульная структура программного обеспечения, включающая подсистемы записи аудиосигнала, предварительной обработки данных, интеллектуального анализа, генерации рекомендаций, хранения

результатов и пользовательского интерфейса. Реализована подсистема захвата аудиоданных на базе Android SDK и компонента AudioRecord.

Также была выполнена интеграция модели YAMNet, используемой в распознавании признаков аудиосигнала. Для решения специализированной задачи анализа качества исполнения реализован дополнительный классификатор, который обучен на сформированной выборке музыкальных фрагментов.

Было проведено дообучение модели и выполнена её оптимизация для мобильных устройств путём конвертации в формат TensorFlow Lite и применения. Полученные результаты показали существенное снижение потребления памяти и времени выполнения анализа без значительного ухудшения качества классификации.

Результаты экспериментального исследования подтвердили эффективность разработанного подхода. Построенные графики обучения продемонстрировали устойчивую сходимость модели, а анализ матрицы ошибок показал высокий уровень качества классификации музыкального исполнения.

Также, помимо интеллектуального модуля были реализованы дополнительные компоненты системы, такие как: музыкальный справочник, содержащий сведения об музыкальных элементах, музыкальный дневник для отслеживания прогресса пользователя, тюнер и метроном для повседневной практики, а также подсистема для взаимодействия между преподавателем и пользователем.

Таким образом, поставленные в работе задачи реализации и интеграции интеллектуального модуля анализа музыкального исполнения были успешно решены.

#### 4 Тестирование приложения

Тестирование приложения является важным этапом в процессе его разработки. Оно позволяет найти различные ошибки, которые мешают нормальной работе приложения, будь мелкая неточность или исключение, вызывающие крах программы. Также тестирование является важной частью связи разработчика и заказчика, потому что оно выявляет все ли требования выполнены.

В текущем разделе представлено тестирование приложения при работе с различными активностями.

Проверка работоспособности метронома

Шаги:

- открыть приложение,
- нажать на кнопку метроном,
- нажать на кнопку “Старт”,
- попробовать переместить ползунок для изменения скорости тиков,
- нажать на кнопку “Стоп”.

Ожидаемый результат:

- запускается активность с метрономом,
- из динамиков слышны тики,
- темп тиков изменяется,
- метроном останавливается.

Фактический результат:

Ошибок нет, фактический результат полностью совпал с ожидаемым.

Результат работы с активностью метроном изображен на (рисунок 13).

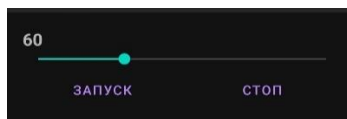


Рисунок 13 – Интерфейс метронома

Проверка работоспособности тюнера (1).

Сначала открыть приложение, потом нажать на кнопку “Тюнер”.

Ожидаемый результат:

- запускается активность с тюнером,
- тюнер помогает настроить гитару в стандартном строе,
- видно положение ползунка.

Фактический результат:

- ошибок нет, тюнер помогает настроить гитару,
- положение ползунка меняется в зависимости от частоты,
- результат (рисунок 14).

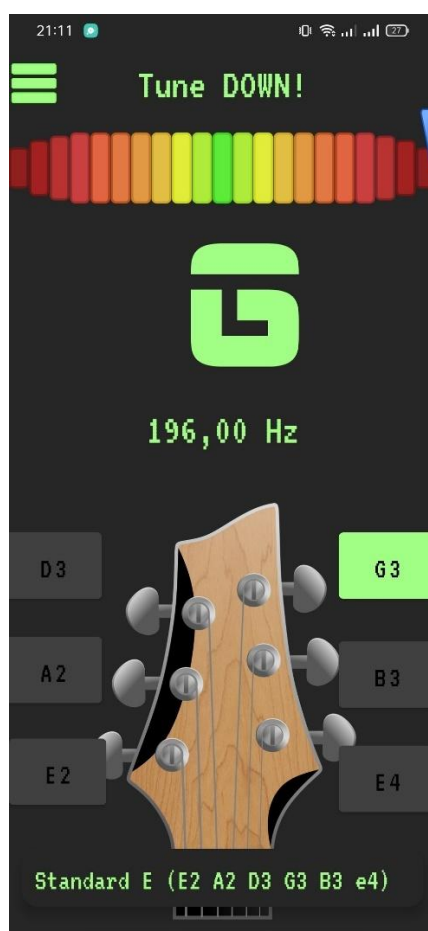


Рисунок 14 – Интерфейс тюнера

Проверка работоспособности тюнера (2)

Шаги:

- открыть приложение,
- нажать кнопку «Тюнер»,

- нажать на надпись со строем,
- выбрать желаемый строй.

Ожидаемый результат:

- открывается список со строями,
- можно будет выбрать желаемый строй,
- строй отображает другие частоты.

Фактический результат:

- ошибок нет,
- открывается список со строями, название и описанием струн,
- выбирается желаемый строй,
- строй отображает другие частоты.

Результат теста:

Ошибок нет, фактический результат полностью совпал с ожидаемым.

Результат (рисунок 15).



Рисунки 15 – Выбор строя

Проверка работоспособности тюнера при создании пользовательского строя.

Шаги:

- открыть меню приложения,

- нажать кнопку «Тюнер» ,
- нажать кнопку «Кнопка с тремя горизонтальными полосками» ,
- нажать на кнопку «Добавить строй» ,
- задать настройки строя,
- выбрать пользовательский строй.

Ожидаемый результат:

- при нажатии на кнопку открывается всплывающее окно,
- открывается окно с настройкой строя,
- строй отображается в списке пользовательских строев.

Фактический результат:

- окно отобразилось,
- открылось окно с настройкой строя,
- строй отображает пользовательские частоты.

Результат теста:

Ошибок нет, фактический результат полностью совпал с ожидаемым.

Результат (рисунок 16 и рисунок 17)

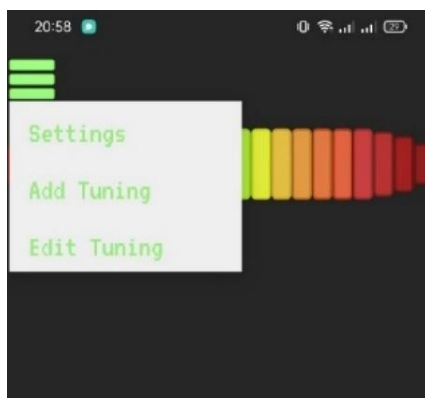
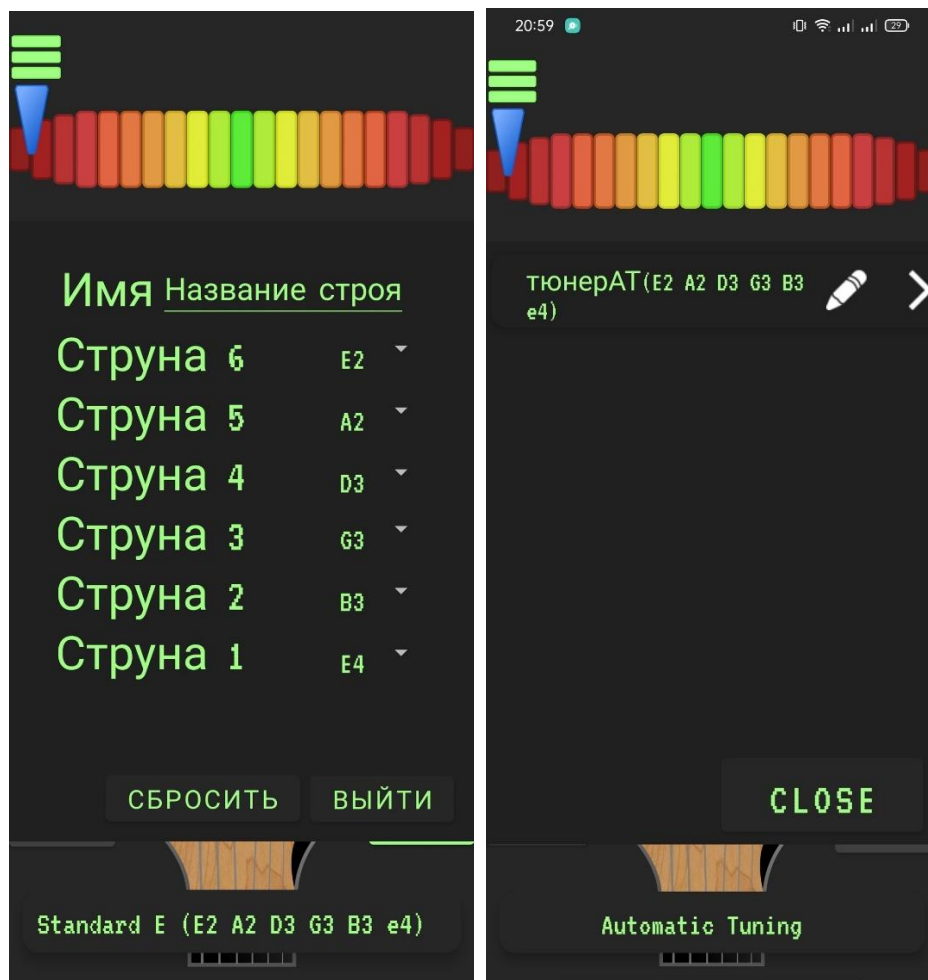


Рисунок 16 – Работа с настройкой пользовательского тюнера



Рисунки 17 – Создание пользовательского тюнера

Проверка активности “Справочник”

Шаги:

- Открыть приложение,
- Нажать кнопку «Справочник».

При проверке активности ожидается что откроется справочник. Итог совпал с ожиданием.

Результат (рисунок 18).

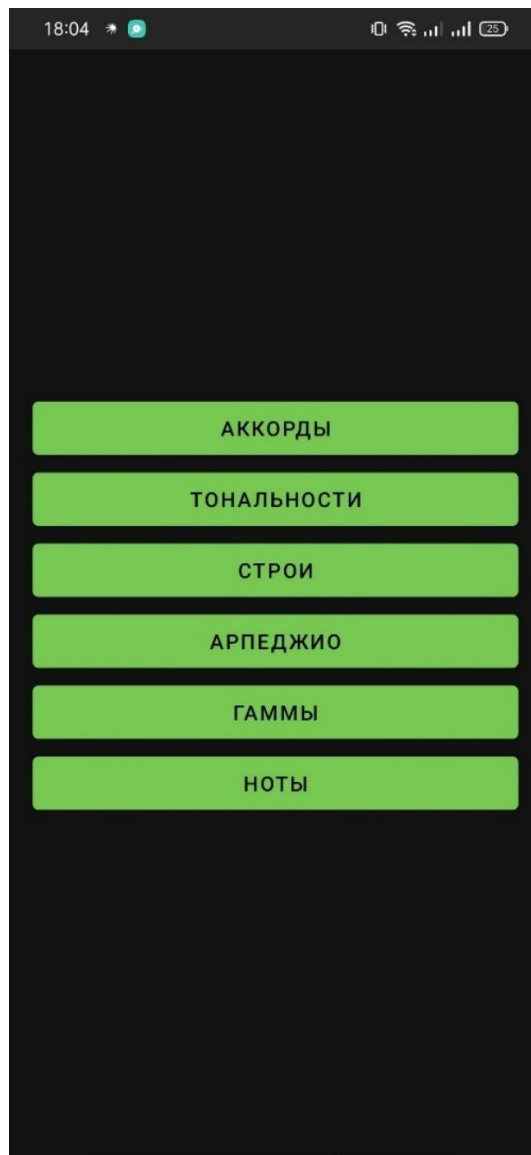


Рисунок 18 – Активность “Справочник”

Проверка активности “Аккорды”

Шаги:

- открыть приложение,
- нажать кнопку «Справочник»,
- нажать кнопку «Аккорды»,
- нажать на выпадающий список,
- выбрать интересующий аккорд.

Ожидаемый результат:

- ошибок нет, откроется активность с аккордами,
- можно выбрать интересующий аккорд.

Фактический результат:

- Активность открылась без ошибок и исключений.
- Аккорд выбирается.
- Аккорд отображается в разных вариация постановки.

Результат (рисунок 19).

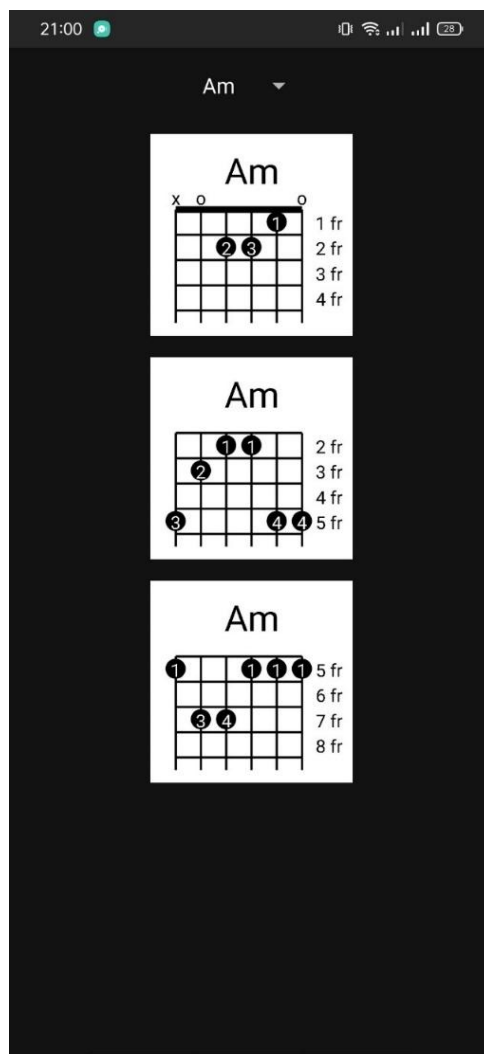


Рисунок 19 – Активность “Аккорды”

Для отображения другого аккорда выберем из выпадающего списка – интересный. Результат работы с активностью “Аккорды” изображен на (рисунке 20 и рисунке 21)



## Проверка активности “Тональности”

### Шаги:

- открыть приложение,
- нажать кнопку «Справочник»,
- нажать кнопку «Тональности».

### Ожидаемый результат:

- Ошибок нет, откроется активность с тональностями.
- Все тональности расписаны.

### Фактический результат:

- Активность открылась без ошибок и исключений.
- Все тональности подробно расписаны.

### Результат работы:

Открылась страничка описывающая каждую тональность с учётом струн и других особенностей

Результат (рисунок 22).

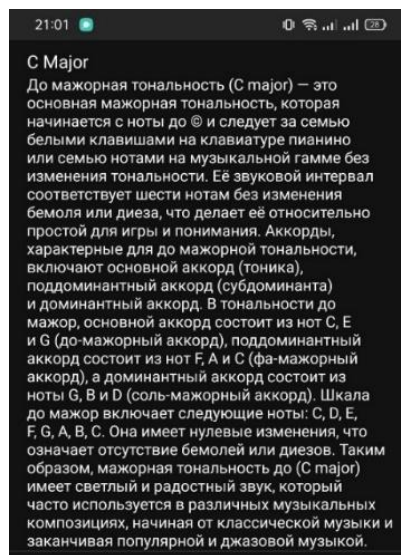


Рисунок 22 – Активность “Тональности”

## Проверка активности “Арпеджио”

### Шаги:

- открыть приложение,
- нажать кнопку «Справочник»,

– нажать кнопку «Гаммы»,

Ожидаемый результат:

– ошибок нет, откроется активность с арпеджио,

– все арпеджио расписаны,

Фактический результат:

– активность открылась без ошибок и исключений,

– все арпеджио подробно расписаны.

Результат (рисунок 23).



Рисунок 23 – Активность “Арпеджио”

Проверка активности “Гаммы”

Шаги:

– открыть приложение,

– нажать кнопку «Справочник»,

– нажать кнопку «Гаммы».

ожидаемый результат:

– ошибок нет, откроется активность с гаммами,

– все гаммы расписаны.

Фактический результат:

– активность открылась без ошибок и исключений,

– все гаммы подробно расписаны.

Результат (рисунок 24).



Рисунок 24 – Активность «Гаммы»

Проверка активности «Ноты»

Шаги:

– открыть приложение,

– нажать кнопку «Справочник»,

– нажать кнопку «Ноты».

Ожидаемый результат:

- ошибок нет, откроется активность с нотами,
- все ноты расписаны.

Фактический результат:

- активность открылась без ошибок и исключений,
- все ноты подробно расписаны.

Результат (рисунок 25).

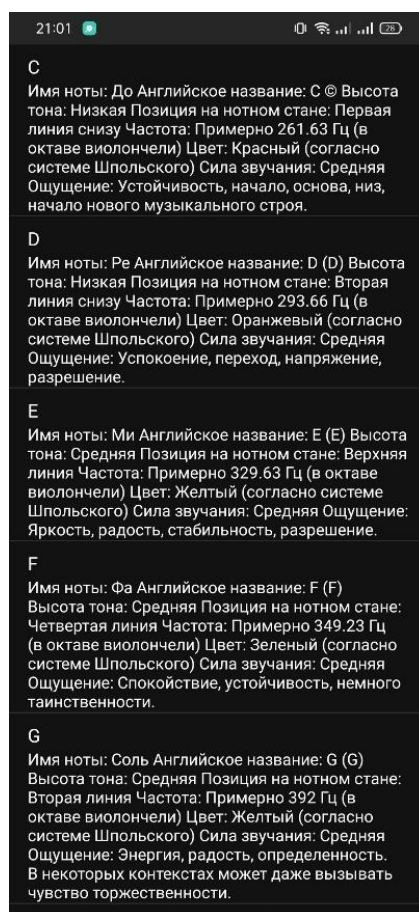


Рисунок 25 – Активность “Ноты”

Проверка активности “Уроки”

Шаги:

- открыть приложение,
- нажать кнопку «Уроки».

Ожидаемый результат:

- ошибок нет, откроется активность с выбором сложности уроков,
- все кнопки нажимаются.

Фактический результат:

- активность открылась без ошибок и исключений.
- все кнопки нажимаются.

Результат (рисунок 26).

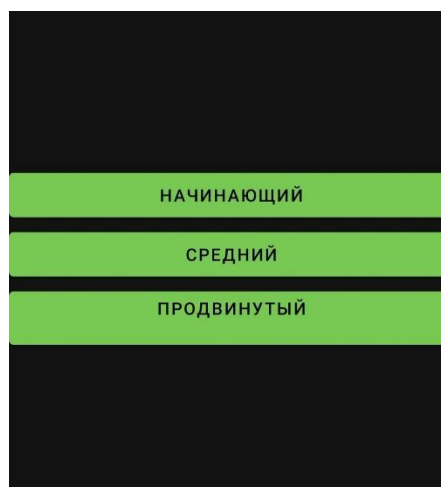


Рисунок 26 – Активность “Уроки”

Проверка активности “Продвинутый”

Шаги:

- открыть приложение,
- нажать кнопку «Уроки»,
- нажать кнопку продвинутый,
- выбрать урок.

Ожидаемый результат:

- ошибок нет, откроется активность с выбором сложности уроков,
- все кнопки нажимаются,
- отображается текст урока.

Фактический результат:

- активность открылась без ошибок и исключений,
- все кнопки нажимаются,
- отображается описание урока.

Результат (рисунок 27).

в единое целое. Советы: - Поддерживайте руку и пальцы в расслабленном состоянии. Напряжение в руке может привести к утомлению и недостаточной точности в игре. - Не забывайте об угле наклона кисти и пальцев относительно грифа. Они должны быть удобны для выполнения быстрых и точных движений. - Практикуйте регулярно и терпеливо. Улучшение техники пальцев требует времени и постоянства. Эти упражнения и советы помогут вам развить технику пальцев на грифе гитары и стать более уверенным и мастеровитым гитаристом. Удачи!



Аккорды: Am, C, D, F, G, Em

There is a lady who sure all that glitters is gold  
And she buying a stairway to Heaven  
When she gets there she knows, if the stores are all  
closed  
With a word she can get what she came for  
Ooh, ooh, and she is buying a stairway to Heaven  
There is a sign on the wall, but she wants to be sure  
Cause you know sometimes words have two  
meanings

Рисунок 27 – Активность “Продвинутый”

### Проверка активности “Анализ игры”

#### Шаги:

- открыть приложение,
- нажать кнопку «Анализ игры»,
- нажать кнопку «Анализировать»,
- играть ноты с плохим извлечением звука.

Ожидаемый результат - отображается текст анализа и генерация упражнения.

#### Фактический результат:

- активность открылась без ошибок и исключений,
- отобразился анализ игры с комментариями.

Результат (рисунок 28)

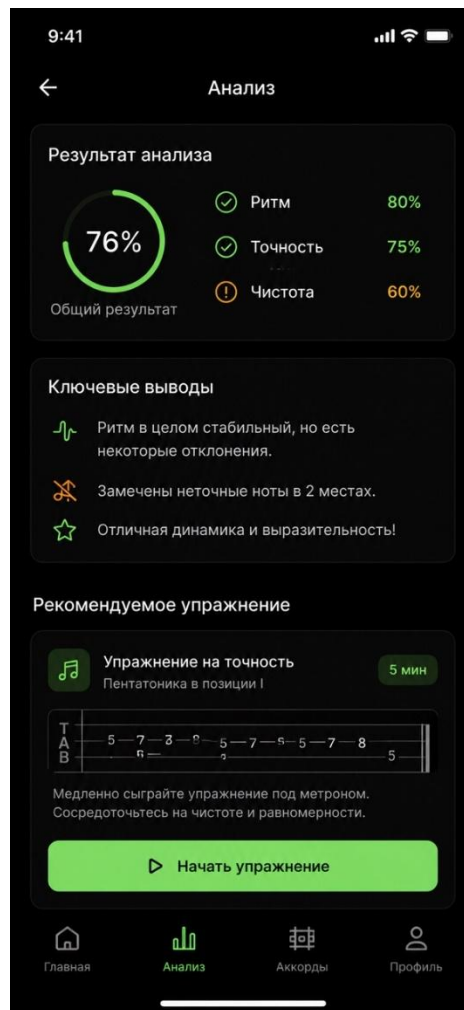


Рисунок 28 – Активность “ Анализ игры ” неточность чистоты извлечения

### Проверка активности “Анализ игры”

#### Шаги:

- открыть приложение,
- нажать кнопку «Анализ игры»,
- нажать кнопку «Анализировать»,
- играть все верно и в точности с оригиналом.
- Ожидаемый результат - отображается текст анализа и генерация упражнения.

#### Фактический результат:

- активность открылась без ошибок и исключений,
- отобразился анализ игры с комментариями.

#### Результат (рисунок 29)

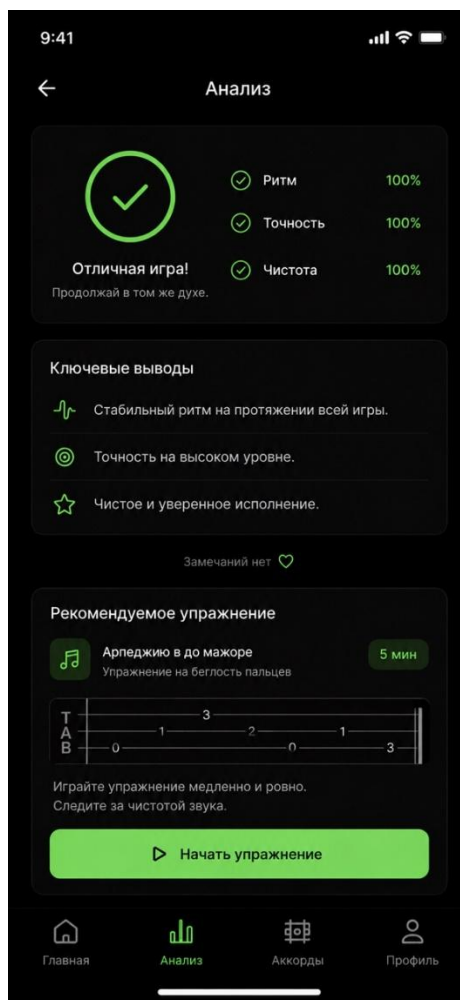


Рисунок 29 – Активность “ Анализ игры ” идеально

### Проверка активности “Анализ игры”

#### Шаги:

- открыть приложение,
- нажать кнопку «Анализ игры»,
- нажать кнопку «Анализировать»,
- играть не поддерживая ритм.

Ожидаемый результат - отображается текст анализа и генерация упражнения.

#### Фактический результат:

- активность открылась без ошибок и исключений,
- отобразился анализ игры с комментариями.

Результат (рисунок 30)

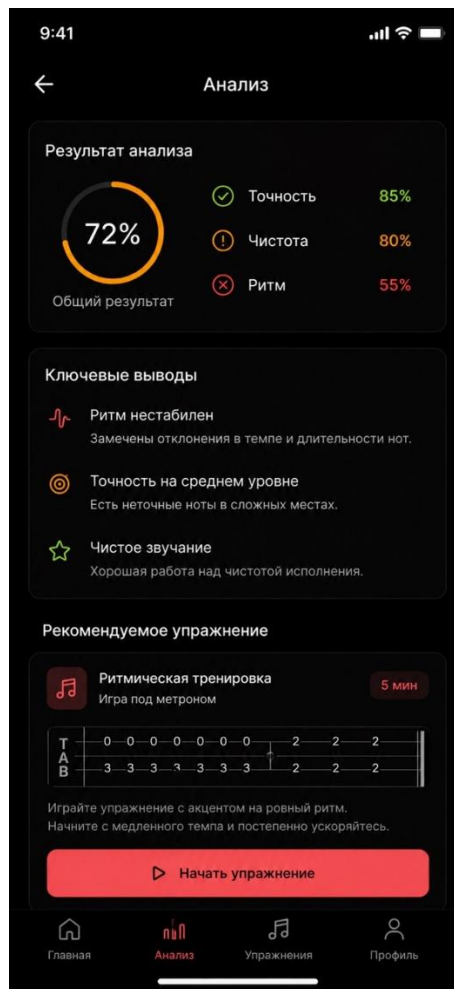


Рисунок 30 – Активность “Анализ игры” без поддержания ритма

### Проверка активности “Анализ игры”

#### Шаги:

- открыть приложение,
- нажать кнопку «Анализ игры»,
- нажать кнопку «Анализировать»,
- играть с уровнем новичка.

Ожидаемый результат - отображается текст анализа и генерация упражнения.

#### Фактический результат:

- активность открылась без ошибок и исключений,
- отобразился анализ игры с комментариями.

Результат изображён на (рисунок 31)

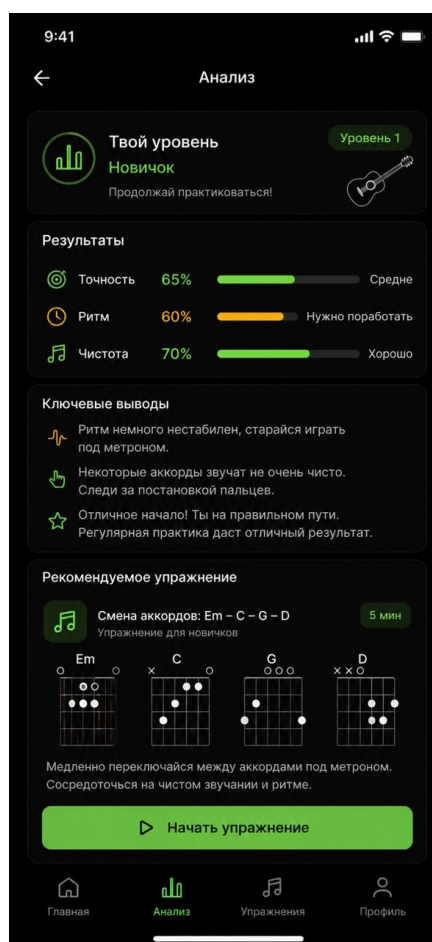


Рисунок 31 – Активность “ Анализ игры ” новичок

Таким образом, в этом разделе было выполнено тестирование приложения, на операционной системе андроид, которое содержит информацию для поддержки гитариста.

В итоге было создано приложение, способное как помочь гитаристу физически, а именно помочь настроить гитару и поддерживать темп. Также информационно – разработан справочник со всей необходимой информацией музыканту.

## **Заключение**

В результате выполнения магистерской диссертации было разработана среда для музыкантов с элементами прикладного искусственного интеллекта.

Среда представляет собой комплексную систему поддержки обучения игре на музыкальных инструментах (гитаре) и объединяет как классические элементы обучения музыки, так и интеллектуальные методы анализа исполнения. В обучающую среду включены такие элементы как анализ игры пользователя, модуль формирования рекомендаций, музыкальный дневник, справочник теории, тюнер, метроном и средства взаимодействия между обучающимся и преподавателем.

Для реализации интеллектуального анализа музыкального исполнения были исследованы методы цифровой обработки аудиосигналов и технологии машинного обучения. В качестве основы выбрана такая архитектура, как YAMNet, адаптированная для решения задачи оценки качества исполнения. Выполнена интеграция модели в мобильное приложение с использованием TensorFlow Lite.

В ходе работы была разработана подсистема записи и предварительной обработки аудиосигналов, включающая нормализацию данных, подавление шумов и формирование Log-Mel спектрограмм. Для повышения качества распознавания был сформирован обучающий набор данных и проведено дообучение модели на специализированных категориях музыкального исполнения.

Проведённые эксперименты показали эффективность применения методов глубокого обучения для анализа аудио данных.

Практическая значимость работы заключается в создании программного средства, которое может использоваться начинающими и опытными музыкантами в качестве интеллектуального помощника при самостоятельном обучении и регулярной практике.

### Список используемых источников

1. Российская академия музыки имени Гнесиных [Электронный ресурс]: Методика музыкального образования / URL: <https://gnesin-academy.ru> (дата обращения: 15.03.2026).
2. Cyberlelinka [Электронный ресурс]: / URL: <https://cyberleninka.ru/article/n/mobilnye-tehnologii-kak-sredstvo-razvitiya-umeniy-audirovaniya-i-govoreniya-v-yazykovom-obrazovanii-shkolnikov/viewer> (дата обращения: 15.03.2026).
3. Yousician [Электронный ресурс]: Официальный сайт / URL: <https://yousician.com> (дата обращения: 02.04.2026).
4. Melodics [Электронный ресурс]: Official Website / URL: <https://melodics.com> (дата обращения: 03.04.2026).
5. SmartMusic [Электронный ресурс]: Official Website / URL: <https://www.smartmusic.com> (дата обращения: 03.04.2026).
6. MathWorks [Электронный ресурс]: Short-Time Fourier Transform / URL: <https://www.mathworks.com/help/signal/ref/stft.html> (дата обращения: 15.04.2026).
7. Librosa Documentation [Электронный ресурс]: Mel-frequency cepstral coefficients / URL: <https://librosa.org/doc/latest/feature.html> (дата обращения: 16.04.2026).
8. TensorFlow [Электронный ресурс]: Audio Classification Tutorial / URL: [https://www.tensorflow.org/tutorials/audio/simple\\_audio](https://www.tensorflow.org/tutorials/audio/simple_audio) (дата обращения: 16.04.2026).
9. TensorFlow [Электронный ресурс]: Audio Recognition Overview / URL: <https://www.tensorflow.org/tutorials/quickstart/beginner?hl=ru> (дата обращения: 16.04.2026).
10. TensorFlow Hub [Электронный ресурс]: Transfer Learning with Audio Models / URL: <https://www.tensorflow.org/hub/tutorials/yamnet> (дата обращения: 20.04.2026).

11. TensorFlow Hub [Электронный ресурс]: YAMNet Model Documentation / URL: <https://tfhub.dev/google/yamnet/1> (дата обращения: 05.05.2026).
12. huggingface [Электронный ресурс]: MobileNet V1 / URL: [https://huggingface.co/docs/transformers/model\\_doc/mobilenet\\_v1](https://huggingface.co/docs/transformers/model_doc/mobilenet_v1) (дата обращения: 11.05.2026).
13. AudioSet [Электронный ресурс]: An ontology and human-labeled dataset for audio events / URL: <https://research.google.com/audioset/> (дата обращения: 16.05.2026).
14. Refactoring Guru [Электронный ресурс]: MVC Design Pattern / URL: <https://refactoring.guru/design-patterns/mvc> (дата обращения: 23.05.2026).

## **ПРИЛОЖЕНИЕ А**

**(обязательное)**

Разработка обучающей среды для музыканта с элементами прикладного  
искусственного интеллекта

Листинг программы

Класс DatabaseHelper:

```
package com.example.guitartoolkit;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class DatabaseHelper extends SQLiteOpenHelper {

 private static final String TAG = "DatabaseHelper";
 private static final String DATABASE_NAME = "guitar_app.db";
 private static final int DATABASE_VERSION = 4;

 // Таблица "Arpeggios"
 private static final String TABLE_ARPEGGIOS = "Arpeggios";
 private static final String COLUMN_ARPEGGIO_ID = "id";
 private static final String COLUMN_ARPEGGIO_NAME = "name";
 private static final String COLUMN_ARPEGGIO_DESCRIPTION =
"description";
 private static final String
COLUMN_ARPEGGIO_IMAGE_RESOURCE_ID = "image";

 // Таблица "Keys"
 private static final String TABLE_KEYS = "Keys";
 private static final String COLUMN_KEY_ID = "id";
 private static final String COLUMN_KEY_NAME = "name";
 private static final String COLUMN_KEY_DESCRIPTION =
"description";
 private static final String COLUMN_KEY_IMAGE_RESOURCE_ID =
"image";

 // Таблица "Scales"
 private static final String TABLE_SCALES = "Scales";
 private static final String COLUMN_SCALE_ID = "id";
 private static final String COLUMN_SCALE_NAME = "name";
 private static final String COLUMN_SCALE_DESCRIPTION =
"description";
 private static final String COLUMN_SCALE_IMAGE_RESOURCE_ID =
"image";

 // Таблица "Tunings"
 private static final String TABLE_TUNINGS = "Tunings";
 private static final String COLUMN_TUNING_ID = "id";
 private static final String COLUMN_TUNING_NAME = "name";
```

```

 private static final String COLUMN_TUNING_DESCRIPTION =
"description";

 // Таблица "Notes"
 private static final String TABLE_NOTES = "Notes";
 private static final String COLUMN_NOTE_ID = "id";
 private static final String COLUMN_NOTE_NAME = "name";
 private static final String COLUMN_NOTE_DESCRIPTION =
"description";
 private static final String COLUMN_NOTE_IMAGE_RESOURCE_ID =
"image";
 private static final String COLUMN_TUNING_IMAGE_RESOURCE_ID
= "image";

 public DatabaseHelper(Context context) {
 super(context, DATABASE_NAME, null, DATABASE_VERSION);
 }

 @Override
 public void onCreate(SQLiteDatabase db) {
 Log.d(TAG, "Creating database...");

 // Создание таблицы "Arpeggios"
 String createArpeggiosTable = "CREATE TABLE IF NOT
EXISTS " + TABLE_ARPEGGIOS + " (" +
 COLUMN_ARPEGGIO_ID + " INTEGER PRIMARY KEY
AUTOINCREMENT," +
 COLUMN_ARPEGGIO_NAME + " TEXT NOT NULL," +
 COLUMN_ARPEGGIO_DESCRIPTION + " TEXT," +
 COLUMN_ARPEGGIO_IMAGE_RESOURCE_ID + " INTEGER
NOT NULL)";
 db.execSQL(createArpeggiosTable);
 Log.d(TAG, "Created table: " + TABLE_ARPEGGIOS);

 // Создание таблицы "Notes"
 String createNotesTable = "CREATE TABLE IF NOT EXISTS "
+ TABLE_NOTES + " (" +
 COLUMN_NOTE_ID + " INTEGER PRIMARY KEY
AUTOINCREMENT," +
 COLUMN_NOTE_NAME + " TEXT NOT NULL," +
 COLUMN_NOTE_DESCRIPTION + " TEXT," +
 COLUMN_NOTE_IMAGE_RESOURCE_ID + " INTEGER NOT
NULL)";
 db.execSQL(createNotesTable);
 Log.d(TAG, "Created table: " + TABLE_NOTES);

 // Создание представления "notes_view"
 String createNotesView = "CREATE VIEW IF NOT EXISTS

```

```

notes_view AS " +
 "SELECT " + COLUMN_NOTE_ID + " AS _id, " +
 COLUMN_NOTE_NAME + ", " +
 COLUMN_NOTE_DESCRIPTION + ", " +
 COLUMN_NOTE_IMAGE_RESOURCE_ID + " FROM " +
TABLE_NOTES;
 db.execSQL(createNotesView);
 Log.d(TAG, "Created view: notes_view");

 // Создание представления "Arpeggios"
 String createArpeggiosView = "CREATE VIEW IF NOT EXISTS
arpeggios_view AS " +
 "SELECT " + COLUMN_ARPEGGIO_ID + " AS _id, " +
 COLUMN_ARPEGGIO_NAME + ", " +
 COLUMN_ARPEGGIO_DESCRIPTION + ", " +
 COLUMN_ARPEGGIO_IMAGE_RESOURCE_ID + " FROM " +
TABLE_ARPEGGIOS;
 db.execSQL(createArpeggiosView);
 Log.d(TAG, "Created view: arpeggios_view");

 // Создание таблицы "Keys"
 String createKeysTable = "CREATE TABLE IF NOT EXISTS " +
TABLE_KEYS + " (" +
 COLUMN_KEY_ID + " INTEGER PRIMARY KEY
AUTOINCREMENT, " +
 COLUMN_KEY_NAME + " TEXT NOT NULL, " +
 COLUMN_KEY_DESCRIPTION + " TEXT, " +
 COLUMN_KEY_IMAGE_RESOURCE_ID + " INTEGER NOT
NULL) ";
 db.execSQL(createKeysTable);
 Log.d(TAG, "Created table: " + TABLE_KEYS);

 // Создание представления "keys_view"
 String createKeysView = "CREATE VIEW IF NOT EXISTS
keys_view AS " +
 "SELECT " + COLUMN_KEY_ID + " AS _id, " +
 COLUMN_KEY_NAME + ", " +
 COLUMN_KEY_DESCRIPTION + ", " +
 COLUMN_KEY_IMAGE_RESOURCE_ID + " FROM " +
TABLE_KEYS;
 db.execSQL(createKeysView);
 Log.d(TAG, "Created view: keys_view");

 // Создание таблицы "Scales"
 String createScalesTable = "CREATE TABLE IF NOT EXISTS "
+ TABLE_SCALES + " (" +
 COLUMN_SCALE_ID + " INTEGER PRIMARY KEY
AUTOINCREMENT, " +

```

```

 COLUMN_SCALE_NAME + " TEXT NOT NULL," +
 COLUMN_SCALE_DESCRIPTION + " TEXT," +
 COLUMN_SCALE_IMAGE_RESOURCE_ID + " INTEGER NOT
NULL) ";
 db.execSQL(createScalesTable);
 Log.d(TAG, "Created table: " + TABLE_SCALES);

 // Создание представления "scales_view"
 String createScalesView = "CREATE VIEW IF NOT EXISTS
scales_view AS " +
 "SELECT " + COLUMN_SCALE_ID + " AS _id, " +
 COLUMN_SCALE_NAME + ", " +
 COLUMN_SCALE_DESCRIPTION + ", " +
 COLUMN_SCALE_IMAGE_RESOURCE_ID + " FROM " +
TABLE_SCALES;
 db.execSQL(createScalesView);
 Log.d(TAG, "Created view: scales_view");

 // Создание таблицы "Tunings"
 String createTuningsTable = "CREATE TABLE IF NOT EXISTS
" + TABLE_TUNINGS + " (" +
 COLUMN_TUNING_ID + " INTEGER PRIMARY KEY
AUTOINCREMENT," +
 COLUMN_TUNING_NAME + " TEXT NOT NULL," +
 COLUMN_TUNING_DESCRIPTION + " TEXT)";
 db.execSQL(createTuningsTable);
 Log.d(TAG, "Created table: " + TABLE_TUNINGS);

 // Создание представления "tunings_view"
 String createTuningsView = "CREATE VIEW IF NOT EXISTS
tunings_view AS " +
 "SELECT " + COLUMN_TUNING_ID + " AS _id, " +
 COLUMN_TUNING_NAME + ", " +
 COLUMN_TUNING_DESCRIPTION + " FROM " +
TABLE_TUNINGS;
 db.execSQL(createTuningsView);
 Log.d(TAG, "Created view: tunings_view");
 }

 @Override
 public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
 Log.d(TAG, "Upgrading database from version " +
oldVersion + " to " + newVersion);

 // Удаляем представления и таблицы и создаем их заново
при обновлении базы данных
 db.execSQL("DROP VIEW IF EXISTS arpeggios_view");

```

```

 db.execSQL("DROP TABLE IF EXISTS " + TABLE_ARPEGGIOS);
 db.execSQL("DROP VIEW IF EXISTS keys_view");
 db.execSQL("DROP TABLE IF EXISTS " + TABLE_KEYS);
 db.execSQL("DROP VIEW IF EXISTS scales_view");
 db.execSQL("DROP TABLE IF EXISTS " + TABLE_SCALES);
 db.execSQL("DROP VIEW IF EXISTS tunings_view");
 db.execSQL("DROP TABLE IF EXISTS " + TABLE_TUNINGS);
 db.execSQL("DROP VIEW IF EXISTS notes_view");
 db.execSQL("DROP TABLE IF EXISTS " + TABLE_NOTES);

 onCreate(db);
 }

 // Методы для работы с таблицей "Arpeggios"
 public void addArpeggio(String name, String description, int
imageResourceId) {
 SQLiteDatabase db = this.getWritableDatabase();
 ContentValues values = new ContentValues();
 values.put(COLUMN_ARPEGGIO_NAME, name);
 values.put(COLUMN_ARPEGGIO_DESCRIPTION, description);
 values.put(COLUMN_ARPEGGIO_IMAGE_RESOURCE_ID,
imageResourceId);
 db.insert(TABLE_ARPEGGIOS, null, values);
 }

 public Cursor getAllArpeggios() {
 SQLiteDatabase db = this.getReadableDatabase();
 return db.rawQuery("SELECT * FROM arpeggios_view",
null);
 }

 // Методы для работы с таблицей "Keys"
 public void addKey(String name, String description, int
imageResourceId) {
 SQLiteDatabase db = this.getWritableDatabase();
 ContentValues values = new ContentValues();
 values.put(COLUMN_KEY_NAME, name);
 values.put(COLUMN_KEY_DESCRIPTION, description);
 values.put(COLUMN_KEY_IMAGE_RESOURCE_ID,
imageResourceId);
 db.insert(TABLE_KEYS, null, values);
 }

 public Cursor getAllKeys() {
 SQLiteDatabase db = this.getReadableDatabase();
 return db.rawQuery("SELECT * FROM keys_view", null);
 }

 // Методы для работы с таблицей "Scales"

```

```

 public void addScale(String name, String description, int
imageResourceId) {
 SQLiteDatabase db = this.getWritableDatabase();
 ContentValues values = new ContentValues();
 values.put(COLUMN_SCALE_NAME, name);
 values.put(COLUMN_SCALE_DESCRIPTION, description);
 values.put(COLUMN_SCALE_IMAGE_RESOURCE_ID,
imageResourceId);
 db.insert(TABLE_SCALES, null, values);
 }

 public Cursor getAllScales() {
 SQLiteDatabase db = this.getReadableDatabase();
 return db.rawQuery("SELECT * FROM scales_view", null);
 }

 // Методы для работы с таблицей "Tunings"
 public void addTuning(String name, String description,
String imageName) {
 SQLiteDatabase db = this.getWritableDatabase();
 ContentValues values = new ContentValues();
 values.put(COLUMN_TUNING_NAME, name);
 values.put(COLUMN_TUNING_DESCRIPTION, description);
 values.put(COLUMN_TUNING_IMAGE_RESOURCE_ID, imageName);
 db.insert(TABLE_TUNINGS, null, values);
 }

 public Cursor getAllTunings() {
 SQLiteDatabase db = this.getReadableDatabase();
 return db.rawQuery("SELECT * FROM tunings_view", null);
 }

 // Методы для работы с таблицей "Notes"
 public void addNote(String name, String description, int
imageResourceId) {
 SQLiteDatabase db = this.getWritableDatabase();
 ContentValues values = new ContentValues();
 values.put(COLUMN_NOTE_NAME, name);
 values.put(COLUMN_NOTE_DESCRIPTION, description);
 values.put(COLUMN_NOTE_IMAGE_RESOURCE_ID,
imageResourceId);
 db.insert(TABLE_NOTES, null, values);
 }

 public Cursor getAllNotes() {
 SQLiteDatabase db = this.getReadableDatabase();
 return db.rawQuery("SELECT * FROM notes_view", null);
 }

```

```
 }
}
```

Главная активность проекта MainActivity:

```
package com.example.guitartoolkit;

import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.widget.Button;

import androidx.appcompat.app.AppCompatActivity;

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class MainActivity extends AppCompatActivity {

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_main);

 DatabaseHelper dbHelper = new DatabaseHelper(this);
 ExecutorService executorService =
Executors.newSingleThreadExecutor();
 Handler handler = new Handler(Looper.getMainLooper());

 Button metronomeButton =
findViewById(R.id.metronomeButton);
 Button tunerButton = findViewById(R.id.tunerButton);
 Button handbookButton =
findViewById(R.id.handbookButton);

 metronomeButton.setOnClickListener(v ->
openMetronomeActivity());
 tunerButton.setOnClickListener(v ->
openTunerActivity());
 handbookButton.setOnClickListener(v ->
openHandbookActivity());
 }

 private void openMetronomeActivity() {
```

```

 Intent intent = new Intent(this,
MetronomeActivity.class);
 startActivity(intent);
 }

 private void openTunerActivity() {
 Intent intent = new Intent(this, TunerActivity.class);
 startActivity(intent);
 }

 private void openHandbookActivity() {
 Intent intent = new Intent(this,
HandbookActivity.class);
 startActivity(intent);
 }
}

```

Makefile activity-main:

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
 xmlns:app="http://schemas.android.com/apk/res-auto"
 xmlns:tools="http://schemas.android.com/tools"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:backgroundTint="@color/green"
 android:gravity="center"
 android:orientation="vertical"
 android:padding="16dp"
 tools:context=".MainActivity">

 <Button
 android:id="@+id/metronomeButton"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_marginBottom="16dp"
 android:text="@string/metronome_button_text"
 android:backgroundTint="@color/dark_green"
 tools:ignore="VisualLintButtonSize" />

 <Button
 android:id="@+id/tunerButton"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_marginBottom="16dp"
 android:text="@string/tuner_button_text"

```

```

 android:backgroundTint="@color/dark_green"
 tools:ignore="VisualLintButtonSize" />

<Button
 android:id="@+id/handbookButton"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_marginBottom="16dp"
 android:text="@string/handbook_button_text"
 android:backgroundTint="@color/dark_green"
 tools:ignore="VisualLintButtonSize" />

</LinearLayout>

```

Активность ArpeggiosActivity:

```

package com.example.guitartoolkit;

import android.annotation.SuppressLint;
import android.content.res.Resources;
import android.database.Cursor;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.widget.ImageView;
import android.widget.ListView;
import android.widget.SimpleCursorAdapter;
import android.widget.TextView;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.content.res.ResourcesCompat;

public class ArpeggiosActivity extends AppCompatActivity {

 private DatabaseHelper dbHelper;
 private TextView arpeggioInfoTextView;
 private ImageView arpeggioImageView;
 private Cursor cursor;
 private Resources resources;

 @Override
 protected void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.activity_arpeggios);

 dbHelper = new DatabaseHelper(getApplicationContext());
 resources = getResources();
 }
}

```

```

 ListView listView = findViewById(R.id.listView);
 arpeggioInfoTextView =
findViewById(R.id.arpeggioInfoTextView);
 arpeggioImageView =
findViewById(R.id.arpeggioImageView);

 addArpeggioRecords();

 cursor = dbHelper.getAllArpeggios();

 String[] fromColumns = {"name", "description"};
 int[] toViews = {android.R.id.text1,
android.R.id.text2};
 SimpleCursorAdapter adapter = new
SimpleCursorAdapter(this, android.R.layout.simple_list_item_2,
cursor, fromColumns, toViews, 0);

 listView.setAdapter(adapter);

 listView.setOnItemClickListener((adapterView, view,
position, id) -> {
 cursor.moveToPosition(position);
 showArpeggio(cursor);
 });
 }

 private void showArpeggio(Cursor cursor) {
 int nameIndex = cursor.getColumnIndex("name");
 int descriptionIndex =
cursor.getColumnIndex("description");
 int imageIndex = cursor.getColumnIndex("image");

 if (nameIndex != -1 && descriptionIndex != -1 &&
imageIndex != -1) {
 String arpeggioName = cursor.getString(nameIndex);
 String arpeggioDescription =
cursor.getString(descriptionIndex);
 String imageName = cursor.getString(imageIndex);

 // Получаем ресурс изображения по его имени
 @SuppressWarnings("DiscouragedApi") int imageResource =
resources.getIdentifier(imageName, "drawable",
getPackageName());

 if (imageResource != 0) {
 Drawable drawable =
ResourcesCompat.getDrawable(resources, imageResource, null);

```

```

arpeggioInfoTextView.setText(getString(R.string.arpeggio_info_format, arpeggioName, arpeggioDescription));
 arpeggioImageView.setImageDrawable(drawable);
 } else {
 // Обработка случая, когда изображение не
 найдено

 arpeggioInfoTextView.setText(getString(R.string.arpeggio_info_not_available));
 arpeggioImageView.setImageDrawable(null);
 }
 } else {
 // Обработка случая, когда столбцы не найдены

 arpeggioInfoTextView.setText(getString(R.string.arpeggio_info_not_available));
 arpeggioImageView.setImageDrawable(null);
 }
}

private void addArpeggioRecords() {
 dbHelper.addArpeggio("Major Triad Arpeggio",
 getString(R.string.major_triad_description), R.drawable.a3);
 dbHelper.addArpeggio("Minor Seventh Arpeggio",
 getString(R.string.minor_seventh_description), R.drawable.c7);
 dbHelper.addArpeggio("Dominant Seventh Arpeggio",
 getString(R.string.dominant_seventh_description),
 R.drawable.a7);
}

@Override
protected void onDestroy() {
 super.onDestroy();
 if (cursor != null) {
 cursor.close();
 }
 if (dbHelper != null) {
 dbHelper.close();
 }
}
}

```

Макет для арпеджио activity-arpeggios:

```

<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
 android:layout_width="match_parent"
 android:layout_height="match_parent">

 <ListView
 android:id="@+id/listView"
 android:layout_width="match_parent"
 android:layout_height="match_parent"
 android:layout_above="@+id/arpeggioInfoTextView"/>

 <TextView
 android:id="@+id/arpeggioInfoTextView"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 android:layout_alignParentBottom="true"
 android:padding="16dp"
 android:text="@string/arpeggio_info_not_available"
 android:textColor="@android:color/black"
 android:textSize="16sp"/>

 <ImageView
 android:id="@+id/arpeggioImageView"
 android:layout_width="wrap_content"
 android:layout_height="wrap_content"
 android:layout_centerHorizontal="true"
 android:layout_above="@id/arpeggioInfoTextView"
 android:src="@drawable/ic_launcher_background"
 android:contentDescription="@null"/>
</RelativeLayout>

```

Обучение классификатора музыкального исполнения

```

import os
import numpy as np
import tensorflow as tf
import tensorflow_hub as hub
import librosa

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Dropout
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.utils import to_categorical

SAMPLE_RATE = 16000

DATASET_PATH = "dataset"

```

```

CLASSES = {
 "clean": 0,
 "rhythm_error": 1,
 "chord_error": 2,
 "noise": 3
}

yamnet_model = hub.load(
 "https://tfhub.dev/google/yamnet/1"
)

def add_noise(signal):
 noise = np.random.randn(len(signal))
 return signal + 0.005 * noise

def time_stretch(signal):
 return librosa.effects.time_stretch(
 signal,
 rate=1.1
)

def pitch_shift(signal):
 return librosa.effects.pitch_shift(
 signal,
 sr=SAMPLE_RATE,
 n_steps=2
)

def volume_change(signal):
 return signal * 1.1

def normalize(signal):
 max_value = np.max(np.abs(signal))

 if max_value == 0:
 return signal

 return signal / max_value

def load_audio(file_path):

 signal, sr = librosa.load(
 file_path,
 sr=SAMPLE_RATE,
 mono=True
)

 signal = normalize(signal)

 return signal

def get_embedding(signal):

```

```

scores, embeddings, spectrogram = yamnet_model(signal)

embedding = tf.reduce_mean(
 embeddings,
 axis=0
)

return embedding.numpy()

X = []
Y = []

for class_name in CLASSES:

 class_path = os.path.join(
 DATASET_PATH,
 class_name
)

 for file_name in os.listdir(class_path):

 if not file_name.endswith(".wav"):
 continue

 file_path = os.path.join(
 class_path,
 file_name
)

 signal = load_audio(file_path)

 embedding = get_embedding(signal)

 X.append(embedding)

 Y.append(CLASSES[class_name])

 signal_noise = add_noise(signal)

 embedding_noise = get_embedding(
 signal_noise
)

 X.append(embedding_noise)

 Y.append(CLASSES[class_name])

 signal_pitch = pitch_shift(signal)

 embedding_pitch = get_embedding(
 signal_pitch

```

```

)

 X.append(embedding_pitch)

 Y.append(CLASSES[class_name])

 signal_stretch = time_stretch(signal)

 embedding_stretch = get_embedding(
 signal_stretch
)

 X.append(embedding_stretch)

 Y.append(CLASSES[class_name])

 signal_volume = volume_change(signal)

 embedding_volume = get_embedding(
 signal_volume
)

 X.append(embedding_volume)

 Y.append(CLASSES[class_name])

X = np.array(X)

Y = np.array(Y)

Y = to_categorical(
 Y,
 num_classes=4
)

dataset_size = len(X)

indices = np.arange(dataset_size)

np.random.shuffle(indices)

X = X[indices]
Y = Y[indices]

split_index = int(
 dataset_size * 0.8
)

X_train = X[:split_index]
Y_train = Y[:split_index]

X_val = X[split_index:]

```

```

Y_val = Y[split_index:]

model = Sequential()

model.add(
 Dense(
 256,
 activation="relu",
 input_shape=(1024,)
)
)

model.add(
 Dropout(
 0.3
)
)

model.add(
 Dense(
 128,
 activation="relu"
)
)

model.add(
 Dropout(
 0.3
)
)

model.add(
 Dense(
 4,
 activation="softmax"
)
)

model.compile(
 optimizer=tf.keras.optimizers.Adam(
 learning_rate=0.0001
),
 loss="categorical_crossentropy",
 metrics=["accuracy"]
)

early_stopping = EarlyStopping(
 monitor="val_loss",
 patience=5,
 restore_best_weights=True
)

reduce_lr = ReduceLRonPlateau(
 monitor="val_loss",

```

```

 factor=0.5,
 patience=3
)
 history = model.fit(
 X_train,
 Y_train,
 validation_data=(
 X_val,
 Y_val
),
 batch_size=32,
 epochs=25,
 callbacks=[
 early_stopping,
 reduce_lr
]
)
 loss, accuracy = model.evaluate(
 X_val,
 Y_val
)
 print("Accuracy:", accuracy)
 model.save(
 "guitar_classifier"
)

```

Генерация рекомендаций:

```

def generate_recommendation(
 predicted_class,
 confidence
):

 if predicted_class == 0:

 return (
 "Исполнение выполнено корректно. "
 "Рекомендуется постепенное увеличение "
 "темпа и усложнение упражнений."
)

 if predicted_class == 1:

 return (
 "Обнаружены ритмические ошибки. "
 "Рекомендуется выполнять упражнения "
 "под метроном в диапазоне "
 "60-80 BPM и постепенно увеличивать темп."
)

 if predicted_class == 2:

```

```

 return (
 "Обнаружены ошибки постановки аккордов. "
 "Рекомендуется выполнять упражнения "
 "на медленную смену аккордов "
 "Am-F-C-G с контролем чистоты звучания."
)

 if predicted_class == 3:

 return (
 "Обнаружено большое количество "
 "посторонних шумов. "
 "Рекомендуется уделить внимание "
 "постановке рук и приглушению струн."
)

 return (
 "Недостаточно данных для анализа."
)
)

```

## **ПРИЛОЖЕНИЕ Б**

**(обязательное)**

Разработка обучающей среды для музыканта с элементами прикладного  
искусственного интеллекта

Презентация

# Разработка обучающей среды для музыканта с элементами прикладного искусственного интеллекта

Студент гр. 24ВВИм2

Трошин А.А

Научный Руководитель:

д.т.н Профессор Бождай А. С.

## Актуальность исследования

### Проблема

Существующие приложения предоставляют:

- тюнер;
- метроном;
- справочные материалы;
- видеоуроки.

### Недостаток:

- Отсутствует интеллектуальная оценка качества исполнения
- Отсутствует автоматическая генерация персональных упражнений



Fender  PLAY

## Цель и задачи работы

Разработка интеллектуального приложения для обучения игре на гитаре.

### Задачи

- анализ предметной области;
- исследование методов ИИ;
- разработка архитектуры;
- реализация мобильного приложения;
- интеграция модели YAMNet;
- генерация персональных рекомендаций;
- тестирование системы.

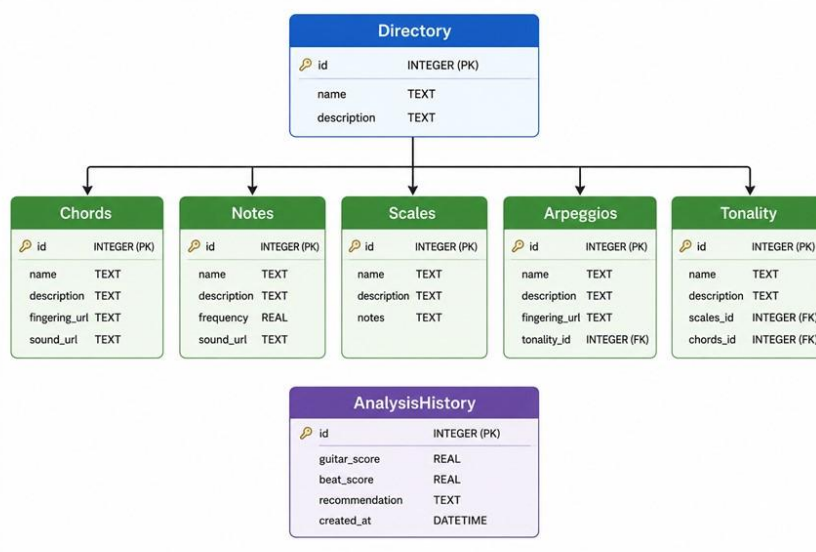
## Структура приложения



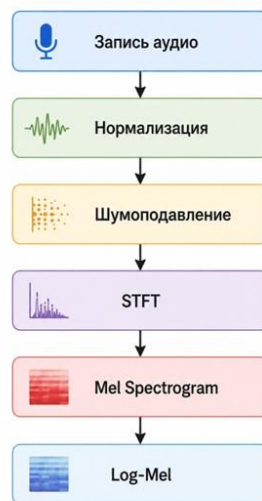
# Архитектура системы



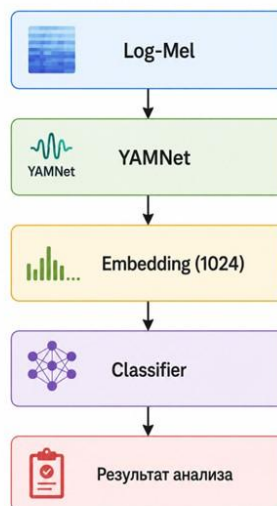
## Модель данных приложения



## Алгоритм анализа аудио



## Интеллектуальный модуль



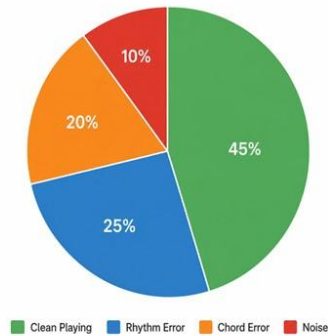
## Набор данных

Класс	Количество фрагментов
Clean Playing	400
Rhythm Error	200
Chord Error	180
Noise	120
Всего	900

- частота дискретизации — 16 кГц;
- длительность фрагмента — 0,96 с;
- формат хранения — WAV;

### разделение выборки:

- обучение — 70 %;
- валидация — 15 %;
- тестирование — 15 %.



## Генерация рекомендаций



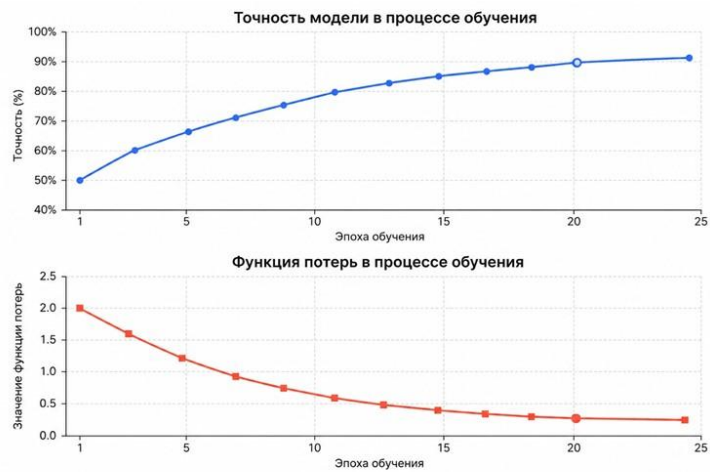
## Результаты обучения

Accuracy — 91.4%

Precision — 90.8%

Recall — 90.1%

F1-score — 90.4%



## Практическая значимость

Преимущества системы:

- Локальный анализ без сервера
- Работа в режиме реального времени
- Персонализированное обучение
- Автоматическая генерация упражнений
- Расширяемость для других инструментов



# Заключение

## Результаты работы

- ✓ разработано Android-приложение
- ✓ реализован интеллектуальный модуль на базе YAMNet
- ✓ реализован музыкальный дневник;
- ✓ разработан справочник музыкальной теории;
- ✓ реализованы тюнер и метроном;
- ✓ обеспечено взаимодействие ученика и преподавателя;
- ✓ достигнута точность 91.4%
- ✓ реализована генерация персональных упражнений
- ✓ подтверждена возможность использования на мобильных устройствах