

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

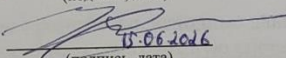
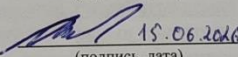
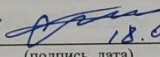
Кафедра
«Вычислительная техника»

Направление подготовки: 09.04.01 «Информатика и вычислительная техника»
Магистерская программа: «Прикладной искусственный интеллект»

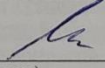
МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

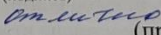
на тему

**ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА ОЦЕНКИ
ЭФФЕКТИВНОСТИ СПОРТСМЕНА В
ИГРОВЫХ ВИДАХ СПОРТА**

Студент	 15.06.2026 (подпись, дата)	Тумасов Вадим Валерьевич (ФИО полностью)
Руководитель	 15.06.2026 (подпись, дата)	Зинкин С.А. (фамилия, инициалы)
Нормоконтролёр	 15.06.2026 (подпись, дата)	Бычков А.С. (фамилия, инициалы)
Рецензент	Доцент ПензГУ  18.06.2026 (должность, место работы) (подпись, дата)	Петерский С.В. (фамилия, инициалы)

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

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

Работа защищена с отметкой  (протокол заседания ГЭК от 23.06.2026 № 26)

Секретарь ГЭК  (подпись) Е.И. Гурин
(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 » 09 2024 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

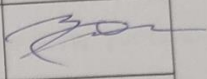
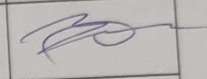
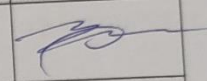
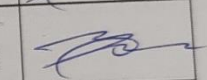
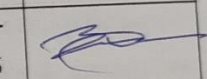
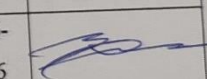
1. Студент Тумасов Вадим Валерьевич гр. 24ВВИм1 факультета ВТ
2. Тема работы «Интеллектуальная система оценки эффективности спортсмена в игровых видах спорта»
- Тема утверждена приказом ПГУ №920/о от "20" сентября 2024г.
3. Руководитель магистерской диссертации д.т.н., профессор кафедры ВТ ПГУ Зинкин С.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.)
Выполнить исследование и разработать программную систему для оценки эффективности спортсмена в игровых видах спорта. Разрабатываемая система должна обеспечивать работу с баскетбольными статистическими данными: добавление карточек игроков, загрузку данных из внешних и локальных источников, хранение информации в базе данных, обработку сезонной статистики, расчет показателей эффективности и отображение результатов анализа в пользовательском интерфейсе. Приложение должно предоставлять возможность просмотра профиля игрока, статистических показателей, сравнительного рейтинга, карты бросков и рекомендаций по развитию игровых характеристик. ML-модуль должен обрабатывать подготовленные признаки игрока, определять вероятные направления улучшения и формировать рекомендации с учетом особенностей баскетбольной предметной области.

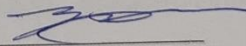
5. Перечень подлежащих разработке вопросов
Требуется провести анализ предметной области спортивной аналитики и определить особенности оценки эффективности спортсмена в игровых видах спорта на примере баскетбола. Необходимо рассмотреть существующие решения, выявить их преимущества и ограничения, определить состав данных для анализа игрока и выбрать способы их получения. В рамках работы требуется спроектировать архитектуру системы RateMyPlayer, включающую клиентское приложение, серверную часть, базу данных, модуль обработки спортивных данных, ML-модуль и модуль формирования рекомендаций. Также необходимо реализовать основные функции приложения:

добавление и загрузку данных об игроках, хранение статистики, расчет рейтинга, построение карты бросков, отображение результатов анализа и формирование рекомендаций по улучшению игровых характеристик. После разработки требуется провести тестирование системы, проверить корректность работы основных модулей и проанализировать полученные результаты.

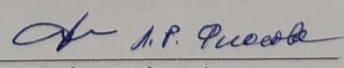
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	11%	20.09.2024-29.01.2025	
2.	Анализ требований	11%	29.01.2025-30.03.2025	
3.	Изучение литературы по теме исследования	22%	30.03.2025-30.05.2025	
4.	Проектирование и реализация системы	27%	01.06.2025-01.09.2025	
5.	Тестирование	17%	02.09.2025-02.10.2025	
6.	Анализ полученных результатов исследования	6%	01.04.2026-29.04.2026	
7.	Оформление пояснительной записки	6%	30.04.2026-25.05.2026	

Дата выдачи задания " 20 " 09 2024 г.

Руководитель магистерской диссертации  Зинкин С.А.
(подпись, дата)

Задание к исполнению принял студент  Тумасов В.В.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)
18. 06. 2026

РЕФЕРАТ

Магистерская диссертация содержит 148 листов, 42 рисунка, 7 таблиц, 20 источников, 2 приложения.

ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА, СПОРТИВНАЯ АНАЛИТИКА, ОЦЕНКА ЭФФЕКТИВНОСТИ СПОРТСМЕНА, МАШИННОЕ ОБУЧЕНИЕ, БАСКЕТБОЛ, ML-МОДУЛЬ, РЕКОМЕНДАЦИИ, FASTAPI, PYSIDE6, SQLITE

Основной целью работы является разработка интеллектуальной системы RateMyPlayer для анализа баскетбольных статистических данных, расчета эффективности игрока, визуализации результатов и формирования рекомендаций с использованием методов машинного обучения.

Объектом исследования является процесс оценки эффективности спортсмена в игровых видах спорта.

Предметом исследования являются методы, алгоритмы и программные средства обработки спортивных данных для расчета эффективности спортсмена и формирования рекомендаций.

В рамках работы проведен анализ предметной области, рассмотрены возможности применения машинного обучения в спортивной аналитике и выполнен обзор существующих аналогов.

Разработана клиент-серверная система RateMyPlayer, включающая пользовательский интерфейс, серверную часть, базу данных, модуль обработки спортивных данных, ML-модуль и модуль рекомендаций. Практическая реализация выполнена на примере баскетбола.

Результаты работы могут использоваться для анализа показателей баскетболистов, расчета рейтинга, визуализации статистики и формирования рекомендаций по улучшению игровых характеристик.

					ПГУ1.090401.15.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Интеллектуальная система оценки эффективности спортсмена в игровых видах спорта	Лит.	Лист	Листов
Разраб.		Тумасов В.В.		15.06.26				
Провер.		Зинкин С.А.		15.06.26			2	148
Н. Контр		Бычков А.С.		15.06.26				
Утверд.		Митрохин М.А.					ФВТ, каф. ВТ гр. 24ВВИм1	

Магистерская диссертация содержит 148 листов, 42 рисунка, 7 таблиц, 20 источников, 2 приложения.

ИНТЕЛЛЕКТУАЛЬНАЯ СИСТЕМА, СПОРТИВНАЯ АНАЛИТИКА, ОЦЕНКА ЭФФЕКТИВНОСТИ СПОРТСМЕНА, МАШИННОЕ ОБУЧЕНИЕ, БАСКЕТБОЛ, ML-МОДУЛЬ, РЕКОМЕНДАЦИИ, FASTAPI, PYSIDE6, SQLITE

Основной целью работы является разработка интеллектуальной системы RateMyPlayer для анализа баскетбольных статистических данных, расчета эффективности игрока, визуализации результатов и формирования рекомендаций с использованием методов машинного обучения.

Объектом исследования является процесс оценки эффективности спортсмена в игровых видах спорта.

Предметом исследования являются методы, алгоритмы и программные средства обработки спортивных данных для расчета эффективности спортсмена и формирования рекомендаций.

В рамках работы проведен анализ предметной области, рассмотрены возможности применения машинного обучения в спортивной аналитике и выполнен обзор существующих аналогов.

Разработана клиент-серверная система RateMyPlayer, включающая пользовательский интерфейс, серверную часть, базу данных, модуль обработки спортивных данных, ML-модуль и модуль рекомендаций. Практическая реализация выполнена на примере баскетбола.

Результаты работы могут использоваться для анализа показателей баскетболистов, расчета рейтинга, визуализации статистики и формирования рекомендаций по улучшению игровых характеристик.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ.....	6
ВВЕДЕНИЕ	7
1 Анализ предметной области.....	10
1.1 Актуальность разработки	10
1.2 Использование машинного обучения в спортивной аналитике	11
1.3 Анализ существующих аналогов.....	12
1.4 Постановка задачи.....	18
1.5 Выводы по разделу.....	19
2 Сбор, хранение и обработка данных	21
2.1 Требования к получаемым данным	21
2.2 Выбор источников данных	22
2.3 Технологии получения доступа к данным.....	24
2.4 Организация хранения данных	26
2.5 Подготовка данных к дальнейшему использованию	28
2.6 Выводы по разделу.....	30
3 Выбор средств разработки.....	30
3.1 Выбор языка программирования	31
3.2 Выбор среды разработки	32
3.3 Выбор системы хранения данных	34
3.4 Выбор серверных технологий.....	36
3.5 Выбор клиентских технологий	38
3.6 Выбор библиотек для обработки данных и машинного обучения.....	40
3.7 Выводы по разделу.....	42
4. Проектирование и разработка программного обеспечения	44
4.1 Общая архитектура программной системы.....	44
4.2 Разработка пользовательского интерфейса	46
4.3 Разработка клиент-серверной части.....	49
4.4 Разработка базы данных	53
4.5 Разработка модуля обработки спортивных данных.....	58

4.6 Разработка ML-модуля оценки эффективности игрока	61
4.7 Разработка модуля формирования рекомендаций.....	72
4.8 Выводы по разделу.....	78
5 Результаты исследования и тестирование системы	79
5.1 Методика тестирования разработанной системы	79
5.2 Проверка работы пользовательского интерфейса и процессов получения, хранения и обработки данных	79
5.3 Проверка работы ML-модуля и формирования рекомендаций	88
5.4 Анализ результатов работы системы.....	100
ЗАКЛЮЧЕНИЕ	102
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	104
ПРИЛОЖЕНИЕ А – Листинг программы	106
ПРИЛОЖЕНИЕ Б - Презентация	136

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей работе применяются следующие сокращения и обозначения:

БД – База данных

ИИ – Искусственный интеллект

МО – Машинное обучение

ПО – Программное обеспечение

СУБД – Система управления базами данных

AI(Artificial Intelligence) – Искусственный интеллект

API(Application Programming Interface) – Программный интерфейс приложения

ASGI(Asynchronous Server Gateway Interface) – Асинхронный серверный шлюзовой интерфейс

CSV(Comma-Separated Values) – Формат хранения табличных данных с разделителями

HTTP(Hypertext Transfer Protocol) – Протокол передачи гипертекста

JSON(JavaScript Object Notation) – Текстовый формат обмена данными

ML(Machine Learning) – Машинное обучение

MLP(Multilayer Perceptron) – Многослойный персептрон

ORM(Object-Relational Mapping) – Объектно-реляционное отображение

REST(Representational State Transfer) – Архитектурный стиль взаимодействия web-сервисов

SQL(Structured Query Language) – Язык структурированных запросов

UI(User Interface) – Пользовательский интерфейс

ВВЕДЕНИЕ

В условиях активного развития цифровых технологий и методов искусственного интеллекта всё большее распространение получают интеллектуальные системы анализа данных, применяемые в различных сферах деятельности человека. Одним из перспективных направлений их использования является спортивная аналитика, где программные средства позволяют собирать, хранить, обрабатывать и анализировать большие объёмы данных о тренировочной и соревновательной деятельности спортсменов[1].

Особую актуальность интеллектуальные системы приобретают в игровых видах спорта, где эффективность спортсмена определяется не одним отдельным показателем, а совокупностью различных статистических характеристик. К таким характеристикам относятся результативность, передачи, подборы, перехваты, блок-шоты, потери, точность игровых действий, стабильность выступлений и вклад игрока в достижение спортивного результата. В рамках данной работы практическая реализация системы выполняется на примере баскетбола, поскольку данный вид спорта имеет развитую систему количественных показателей, пригодных для программной обработки и применения методов машинного обучения.

В настоящее время оценка эффективности спортсмена часто выполняется на основе ручного анализа, экспертного мнения тренера или ограниченного набора статистических данных. Такой подход не всегда дает комплексную картину, поскольку в игре важны не только яркие результативные действия, но и менее заметные эпизоды: сохранение владения, помощь партнерам, работа в защите и надежность на протяжении нескольких матчей. Поэтому в работе разрабатывается интеллектуальная система, которая автоматически обрабатывает баскетбольную статистику, рассчитывает показатели эффективности и формирует рекомендации по улучшению игровых характеристик.

Тема магистерской диссертации — разработка интеллектуальной системы оценки эффективности спортсмена в игровых видах спорта.

Цель работы состоит в создании программной системы RateMyPlayer для сбора, хранения, обработки и анализа баскетбольных статистических данных. Система рассчитывает эффективность игрока, отображает результаты в наглядном виде и формирует рекомендации с применением методов машинного обучения.

Гипотеза исследования заключается в том, что сочетание статистического анализа, ML-модели и экспертных правил позволит сделать оценку баскетболиста более объективной и дать пользователю не просто набор чисел, а понятные рекомендации по развитию игровых показателей.

Объектом исследования является процесс оценки эффективности спортсмена в игровых видах спорта.

Предметом исследования являются методы, алгоритмы и программные средства обработки спортивных данных, применяемые для расчета эффективности спортсмена и формирования рекомендаций по улучшению его игровых характеристик.

Основными задачами работы являются:

- анализ предметной области и обоснование актуальности разработки интеллектуальной системы оценки эффективности спортсмена,
- рассмотрение возможностей применения машинного обучения в спортивной аналитике,
- анализ существующих аналогов и выявление их преимуществ и ограничений,
- определение требований к данным, необходимым для оценки эффективности игрока,
- выбор источников получения баскетбольных статистических данных и технологий доступа к ним,
- организация хранения данных об игроках, статистических показателях и игровых действиях,

- выбор средств разработки программной системы, включая язык программирования, среду разработки, систему хранения данных, серверные и клиентские технологии, библиотеки обработки данных и машинного обучения,
- проектирование и реализация клиент-серверной архитектуры приложения,
- разработка пользовательского интерфейса для работы с данными игроков и результатами анализа,
- разработка базы данных для хранения информации об игроках, сезонной статистике и бросках,
- реализация модуля обработки баскетбольных данных и расчета эффективности игрока,
- разработка ML-модуля оценки эффективности и модуля формирования рекомендаций,
- проведение тестирования разработанной системы и анализ результатов её работы.

Магистерская диссертация направлена на разработку интеллектуальной программной системы, предназначенной для автоматизированной оценки эффективности спортсмена в игровых видах спорта. Практическая реализация системы RateMyPlayer выполнена на примере баскетбола и позволяет объединить хранение спортивных данных, статистическую обработку, расчет рейтинга, визуализацию результатов и формирование рекомендаций в одном программном решении.

1 Анализ предметной области

1.1 Актуальность разработки

В современных условиях спортивная индустрия активно использует информационные технологии для анализа тренировочной и соревновательной деятельности спортсменов. Команды, тренеры и аналитики применяют данные не только для фиксации результатов матчей, но и для оценки игровых действий, выявления сильных и слабых сторон игроков, а также принятия решений при планировании тренировочного процесса.

Особенно важной задачей спортивной аналитики является оценка эффективности игрока в игровых видах спорта. Такая оценка не может быть сведена только к одному показателю, например количеству набранных очков[2]. Для объективного анализа необходимо учитывать не один показатель, а общий игровой профиль спортсмена: результативность, точность действий, участие в атаке и защите, надежность при владении мячом и стабильность выступлений[3]. Такие данные можно хранить в базе, сравнивать между игроками, использовать при расчете рейтинга и передавать в ML-модуль для дальнейшей обработки.

При ручном разборе статистики тренер видит отдельные показатели, но связи между ними приходится оценивать самостоятельно. В баскетболе это особенно заметно: высокий набор очков не всегда означает общую полезность, если он сопровождается частыми потерями, а менее результативный игрок может влиять на игру за счет защиты, передач и командных действий[4].

Поэтому RateMyPlayer рассматривается как инструмент, который собирает данные игрока в единый игровой профиль и помогает быстрее увидеть сильные стороны и проблемные зоны. Машинное обучение в системе используется как дополнительный аналитический механизм: оно не заменяет тренерскую оценку, а помогает выделить направления, на которые стоит обратить внимание при дальнейшем развитии игрока.

Разрабатываемая система RateMyPlayer ориентирована на такую форму анализа, при которой данные игрока рассматриваются не отдельно, а как

единый игровой профиль. Это важно для баскетбола, где вывод о полезности спортсмена должен учитывать не только результативность, но и качество решений, стабильность, защитную работу и влияние на командные действия.

1.2 Использование машинного обучения в спортивной аналитике

В спортивной аналитике машинное обучение используют там, где обычного просмотра статистики уже недостаточно. Игровые показатели связаны между собой: результативность зависит не только от количества бросков, а полезность игрока может проявляться через передачи, подборы, защитные действия, снижение потерь и стабильность на протяжении сезона. ML-модель позволяет работать с такими данными как с единой системой признаков и находить зависимости, которые трудно оценить при ручном анализе[5].

В зависимости от задачи могут применяться разные подходы. Регрессионные модели подходят для прогнозирования числовых показателей, например ожидаемой результативности или изменения рейтинга[7]. Классификация может использоваться для определения уровня игрока или выбора направлений развития. Кластеризация помогает группировать спортсменов со схожим игровым профилем: атакующие защитники, игроки с выраженной защитной ролью, универсальные форварды и другие типы. На практике такие методы применяются для анализа формы игрока, сравнения спортсменов и подготовки рекомендаций для тренировочного процесса.

В системе RateMyPlayer машинное обучение используется как часть анализа баскетбольной статистики. На вход ML-модуля передается не исходная статистика целиком, а подготовленный набор признаков. Он отражает основные стороны игры баскетболиста: результативность, качество владения мячом, бросковую точность, защитную активность и распределение бросков по типам. В результате пользователь получает не только числовые показатели, но и практическую интерпретацию данных.

Для наглядного представления роли машинного обучения в спортивной аналитике используется схема, представленная на рисунке 1.



Рисунок 1 – Применение машинного обучения в спортивной аналитике

Машинное обучение в спортивной аналитике помогает перейти от разрозненной статистики к более прикладному выводу[6]. На его основе можно быстрее обработать показатели игрока, сравнить разные стороны его игры и выделить направления для дальнейшей работы. Для системы RateMyPlayer это особенно важно, поскольку итоговая рекомендация должна опираться не на один показатель, а на совокупность игровых характеристик баскетболиста.

1.3 Анализ существующих аналогов

Спортивная аналитика уже давно вышла за рамки обычной таблицы с результатами матча. В профессиональных клубах используются системы, которые собирают данные о перемещениях игроков, фиксируют игровые эпизоды, считают нагрузку, помогают разбирать тактику и сравнивать спортсменов между собой. С такими инструментами работают тренеры, аналитики, скауты и специалисты по физической подготовке. Их задача — быстрее получить объективную картину игры и принять решение не только на основе личного впечатления, но и на основе данных.

Существующие решения можно разделить на несколько групп: трекинговые системы, видеоаналитические платформы, статистические сервисы и открытые источники спортивных данных. Каждая группа решает свою задачу. Одни системы собирают координаты и физические показатели, другие помогают размечать видео, третьи предоставляют готовую статистику по игрокам, командам и матчам.

К системам трекинга относятся решения, которые фиксируют перемещения игроков, мяча или шайбы во время игры. С их помощью можно определить положение спортсмена на площадке, скорость движения,

ускорения, пройденную дистанцию, зоны активности и интенсивность нагрузки. Такие данные особенно полезны при анализе командного взаимодействия, защитных перестроений и физического состояния игрока. К данной группе можно отнести следующие системы[8]:

– STATS SportVU — система, использующая камеры, установленные на спортивной арене, для отслеживания перемещений игроков и мяча(рисунк 2). Она позволяет анализировать расположение игроков, активность в различных зонах площадки и особенности командного взаимодействия,

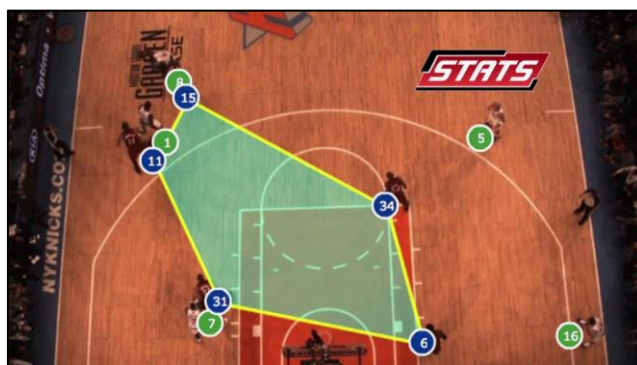


Рисунок 2 – Пример работы SportVU

– Second Spectrum — система спортивной аналитики, применяющая компьютерное зрение и методы искусственного интеллекта для анализа игровых событий, тактических схем и поведения игроков на площадке(рисунк 3)[9],



Рисунок 3 – Пример работы Second Spectrum

– Catapult Tracking Systems — система, использующая носимые датчики для отслеживания физической нагрузки спортсмена, скорости, ускорения, интенсивности движений и других параметров, связанных с физическим состоянием игрока(рисунк 4)[10].



Рисунок 4 – Пример работы Catapult Tracking Systems

Преимуществом систем трекинга является высокая точность и детализация собираемых данных. Однако такие решения требуют специализированного оборудования, камер, датчиков и сложной технической инфраструктуры, поэтому чаще применяются профессиональными клубами.

К видеоаналитическим системам относятся решения, предназначенные для работы с видеозаписями матчей, выделения ключевых моментов, разметки игровых событий и подготовки аналитических отчетов. К данной группе относятся:

– Hudl Sportscode — платформа для разметки игровых эпизодов, анализа действий игроков, формирования видеонарезок и подготовки команды к матчам. Ограничением платформы является то, что разметка игровых эпизодов и подготовка видеоматериалов во многом выполняются аналитиком вручную(рисунок 5)[11],



Рисунок 5 – Пример работы Hudl

– Wyscout — сервис видеоаналитики и скаутинга, в котором доступны записи матчей, статистика игроков и командные показатели. Его используют при подборе спортсменов, сравнении игровых профилей и предварительном анализе соперников(рисунок 6),



Рисунок 6 – Пример работы Wyscout

– LongoMatch — программа для разметки видеозаписей матчей и выделения отдельных игровых эпизодов. Она подходит для базового видеоразбора, однако большая часть действий выполняется вручную, а полноценная интеллектуальная оценка эффективности игрока в системе не предусмотрена(рисунок 7),



Рисунок 7 – Пример работы LongoMatch

– Synergy Sports — аналитическое решение для работы с видеоматериалами, статистикой и игровыми эпизодами, широко применяемое в баскетболе. Система полезна для тренерского анализа и скаутинга, но

относится к коммерческим продуктам и не дает возможности свободно изменять алгоритмы оценки под конкретную задачу(рисунок 8)[12].

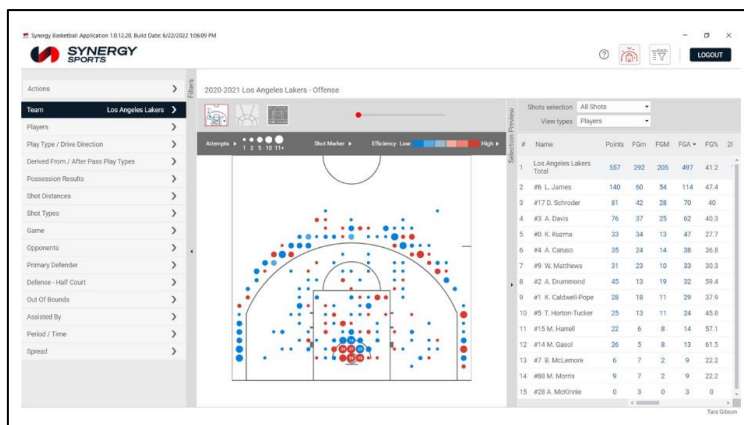


Рисунок 8 – Пример работы Synergy Sports

Видеоаналитические системы помогают разбирать игру по эпизодам, но сильно зависят от качества записи и работы аналитика. Разметка событий, отбор фрагментов и итоговая интерпретация во многих случаях выполняются вручную, поэтому такие решения не всегда подходят для автоматизированной оценки игрока.

Отдельную группу составляют статистические платформы. Они работают не с видеозаписью матча, а с уже собранными спортивными данными: таблицами игроков, командными показателями, расширенными метриками и рейтингами. К данной группе можно отнести следующие решения:

– Opta / Stats Perform — платформа спортивной статистики, используемая клубами, спортивными организациями и медиа. Она содержит большой объем обработанных данных по игрокам, командам и матчам, однако относится к коммерческим решениям, а часть методик расчета показателей остается закрытой(рисунок 9)[13],

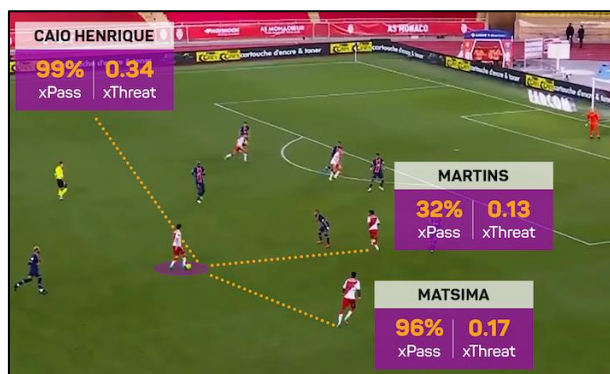


Рисунок 9 – Пример работы Opta Sports

– InStat Sport — система для анализа матчей, просмотра видеоматериалов, получения статистики игроков и формирования отчетов. Пользователь работает преимущественно с уже подготовленными данными, что ограничивает возможность гибкой настройки алгоритмов оценки (рисунок 10),

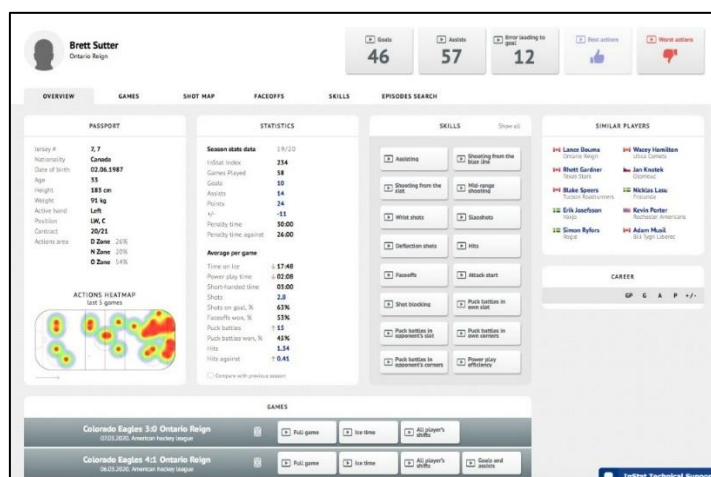


Рисунок 10 – Пример работы InStat Sport

Анализ существующих аналогов показывает, что современные системы спортивной аналитики обладают широкими возможностями, однако имеют ряд ограничений. Системы трекинга требуют дорогостоящего оборудования, видеоаналитические платформы часто зависят от ручной разметки, а статистические сервисы не всегда позволяют адаптировать методику оценки под конкретную задачу.

Таким образом, анализ существующих аналогов подтверждает целесообразность разработки собственной интеллектуальной системы оценки эффективности спортсмена. Разрабатываемая система RateMyPlayer должна

объединять хранение данных, статистическую обработку, расчет эффективности, визуализацию и интеллектуальное формирование рекомендаций в одном программном решении.

1.4 Постановка задачи

Проведенный анализ предметной области и существующих систем спортивной аналитики показал, что современные решения обладают рядом ограничений при применении для локальной оценки эффективности игрока. К основным проблемам относятся высокая стоимость профессиональных платформ, необходимость использования специализированного оборудования, зависимость от внешних сервисов, ручная разметка данных и ограниченная возможность адаптации алгоритмов оценки под конкретную задачу.

Кроме того, многие существующие решения ориентированы преимущественно на профессиональные клубы, скаутские службы или крупные аналитические платформы. Такие системы часто предоставляют готовую статистику или инструменты видеоанализа, но не всегда обеспечивают комплексную локальную обработку данных, расчет эффективности игрока, визуализацию результатов и формирование рекомендаций в рамках одного программного продукта.

В связи с этим возникает необходимость разработки интеллектуальной программной системы, предназначенной для автоматизированной оценки эффективности игрока на основе спортивных статистических данных. В работе баскетбол выбран как прикладная область, так как его игровая статистика хорошо формализуется и подходит для программной обработки: данные можно хранить в базе, использовать при расчете рейтинга, построении карты бросков и формировании признаков для ML-модуля.

Объектом исследования является процесс оценки эффективности спортсмена в игровых видах спорта.

Предметом исследования являются методы, алгоритмы и программные средства обработки спортивных данных, применяемые для расчета

эффективности спортсмена и формирования рекомендаций по улучшению его игровых характеристик.

Целью данной работы является разработка интеллектуальной программной системы RateMyPlayer, обеспечивающей сбор, хранение, обработку и анализ баскетбольных статистических данных, расчет эффективности игрока, визуализацию результатов и формирование рекомендаций с использованием методов машинного обучения.

Для реализации цели работы были выделены следующие задачи:

- проанализировать предметную область спортивной аналитики и определить, почему задача оценки эффективности игрока требует автоматизации,
- рассмотреть, как методы машинного обучения применяются для анализа спортивной статистики и формирования рекомендаций,
- изучить существующие решения в области спортивной аналитики, включая трекинговые системы, видеоаналитические платформы и статистические сервисы,
- определить состав данных, необходимых для анализа игрока: профиль спортсмена, сезонные показатели, сведения о бросках и игровые действия,
- выбрать источники получения данных и способы доступа к ним, включая ESPN Public API, CSV- и JSON-файлы,
- разработать структуру базы данных для хранения игроков, статистики и бросковых попыток,
- обосновать выбор средств разработки: Python, PyCharm, SQLite, FastAPI, PySide6, SQLAlchemy и библиотек для обработки данных и машинного обучения,
- спроектировать клиент-серверную структуру приложения RateMyPlayer,
- реализовать пользовательский интерфейс для просмотра профиля игрока, статистики, сравнительного рейтинга, карты бросков и рекомендаций,
- разработать серверные сервисы обработки спортивных данных,

- реализовать ML-модуль и модуль формирования рекомендаций по улучшению игровых характеристик,
- провести тестирование системы и проанализировать полученные результаты.

Итогом постановки задачи является разработка RateMyPlayer как приложения, в котором данные баскетболиста проходят полный путь: от загрузки и сохранения до отображения в интерфейсе, построения аналитических результатов и формирования рекомендаций.

1.5 Выводы по разделу

В первом разделе показано, почему оценка спортсмена в игровых видах спорта требует комплексного подхода. Для баскетбола важны не только очки, но и передачи, подборы, потери, точность бросков, защитные действия и стабильность.

Рассмотренные аналоги решают отдельные задачи спортивной аналитики: трекинг перемещений, видеоразбор, работу со статистикой или скаутинг. При этом многие из них требуют специального оборудования, ручной разметки или не позволяют гибко менять алгоритм оценки.

На этом основании обоснована разработка RateMyPlayer как локальной системы, которая объединяет обработку баскетбольной статистики, расчет рейтинга, карту бросков и рекомендации на основе ML-модуля.

2 Сбор, хранение и обработка данных

2.1 Требования к получаемым данным

Для корректной работы интеллектуальной системы оценки эффективности спортсмена необходимо определить требования к данным, которые будут использоваться при анализе. Поскольку разрабатываемая система RateMyPlayer предназначена для обработки спортивной статистики, входные данные должны быть структурированными, числовыми, проверяемыми и пригодными для дальнейшего хранения в базе данных, расчета показателей эффективности и использования в ML-модуле.

В рамках данной работы основное внимание уделяется баскетболу как одному из игровых видов спорта, в котором используется большое количество формализованных статистических показателей. Для анализа эффективности игрока необходимы данные, характеризующие как самого спортсмена, так и его игровые результаты. К таким данным относятся сведения о ФИО игрока, команде, позиции, возрасте, росте, весе, сезоне, результативности, передачах, подборках, перехватах, блок-шотах, потерях и процентах реализации бросков.

Получаемые данные должны соответствовать нескольким основным требованиям. Во-первых, они должны иметь единую структуру, чтобы система могла автоматически обрабатывать их независимо от источника поступления. Во-вторых, числовые показатели должны находиться в допустимых диапазонах: например, возраст, рост и вес игрока не должны принимать некорректные значения, а проценты реализации бросков должны быть представлены в диапазоне от 0 до 1. В-третьих, данные должны быть пригодны для хранения в базе данных и последующего извлечения через серверную часть приложения.

Отдельную группу входных данных составляют сведения о бросках игрока. Для построения карты бросков необходимо хранить координаты броска на площадке, результат попытки, тип броска, номер четверти, время игрового эпизода и дату матча. Такие данные позволяют не только отображать

расположение бросков, но и анализировать эффективность игрока в различных зонах площадки.

Качество входных данных напрямую влияет на корректность работы системы. Если данные содержат пропуски, ошибочные значения или отличаются по формату, это может привести к неправильному расчету эффективности игрока, некорректной визуализации или снижению качества работы ML-модуля. Поэтому перед использованием данные должны проходить проверку, очистку и преобразование к единому виду.

Таким образом, получаемые данные должны быть структурированными, полными, достоверными и пригодными для автоматизированной обработки. Соблюдение данных требований необходимо для корректного хранения информации, расчета эффективности игрока, построения визуализаций и формирования рекомендаций в системе RateMyPlayer.

2.2 Выбор источников данных

Для работы интеллектуальной системы оценки эффективности спортсмена необходимо выбрать источники данных, из которых будет поступать информация о спортсменах, их статистических показателях и игровых действиях. От выбора источников зависит полнота анализа, корректность расчета эффективности, возможность построения визуализаций и качество дальнейшего использования данных в ML-модуле.

Для системы RateMyPlayer важны источники, которые дают не только итоговые результаты матчей, но и данные, пригодные для дальнейшей обработки: сведения об игроках, сезонные статистические строки, показатели бросков и отдельные игровые действия. Такая информация может быть сохранена в базе данных, связана с конкретной карточкой спортсмена и использована при расчете рейтинга, построении карты бросков и подготовке признаков для ML-модуля.

В качестве основного внешнего источника данных в системе используется ESPN Public API. Данный источник позволяет получать сведения о спортивных лигах, командах, игроках и их статистике[14]. Использование

API является удобным способом получения структурированных данных, так как информация передается в формате, пригодном для автоматизированной обработки серверной частью приложения. Полученные данные могут быть сохранены в локальной базе данных и использованы для последующего анализа.

Помимо ESPN Public API, RateMyPlayer поддерживает загрузку заранее подготовленных наборов данных. Для этого используются CSV- и JSON-файлы. CSV подходит для табличных записей, например списка игроков или сезонной статистики, а JSON удобен для более связанной структуры данных, где вместе могут храниться сведения об игроке и дополнительные параметры. Такой вариант нужен в тех случаях, когда пользователь работает с собственными данными или не обращается к внешнему API.

Локальная база данных в данном разделе рассматривается как внутренний источник уже сохраненной информации. После импорта или ручного ввода сведения становятся доступными внутри приложения и могут использоваться при дальнейшем анализе без повторного обращения к исходному файлу или внешнему сервису.

Отдельно следует выделить данные о бросках игрока. Они могут поступать из подготовленных файлов или формироваться внутри системы. Такие данные содержат координаты броска на площадке, результат попытки, тип броска и дополнительную информацию об игровом эпизоде. Они необходимы для построения карты бросков и более наглядного анализа игровых действий спортсмена.

Основные источники данных, используемые в системе RateMyPlayer, представлены в таблице 1.

Таблица 1 — Источники данных системы RateMyPlayer

Источник данных	Назначение	Используемые данные
ESPN Public API	Получение спортивной информации из внешнего источника	Данные о лигах, командах, игроках и статистических показателях

Продолжение таблицы 1.

Источник данных	Назначение	Используемые данные
CSV-файлы	Импорт табличных данных в систему	Список игроков, сезонная статистика, показатели эффективности
JSON-файлы	Импорт структурированных данных	Сведения об игроках, статистике и дополнительных параметрах
Локальная база данных	Хранение и повторное использование информации	Игроки, статистика, броски, результаты анализа
Подготовленные данные о бросках	Построение карты бросков и анализ игровых действий	Координаты бросков, результат попытки, тип броска, дата и время эпизода

Выбор нескольких источников данных позволяет повысить гибкость разрабатываемой системы. Внешний API обеспечивает возможность получения спортивной информации из открытого источника, файловый импорт позволяет использовать заранее подготовленные наборы данных, а локальная база данных обеспечивает хранение и повторное использование информации внутри приложения.

Таким образом, в системе RateMyPlayer в качестве источников данных используются внешний API, локальные файлы форматов CSV и JSON, а также собственная база данных. Такой подход позволяет объединить автоматизированное получение информации, ручную загрузку подготовленных данных и локальное хранение результатов для дальнейшей обработки и анализа.

2.3 Технологии получения доступа к данным

В системе RateMyPlayer получение доступа к данным реализуется несколькими способами: через внешний источник ESPN Public API, импорт локальных CSV- и JSON-файлов, а также через взаимодействие клиентской и серверной частей приложения. Такой подход позволяет использовать как

актуальные спортивные данные из внешних источников, так и заранее подготовленные наборы данных.

Основным способом получения внешней информации является обращение к ESPN Public API. Доступ к данным выполняется серверной частью приложения с помощью HTTP-запросов. Для этого используется библиотека `requests`. В ответ сервер получает данные в формате JSON, извлекает необходимые сведения об игроках, командах и статистике, после чего приводит их к внутренней структуре системы.

Файловый импорт используется как отдельный способ загрузки заранее подготовленных наборов данных. CSV-файлы используются для импорта табличной статистики, а JSON-файлы — для загрузки более структурированной информации. Обработка CSV-файлов выполняется с использованием библиотеки `pandas`, а JSON-данные считываются стандартными средствами Python[15].

Взаимодействие между клиентской и серверной частями реализовано через REST API на основе FastAPI. Клиентское приложение отправляет запросы на сервер для получения списка игроков, загрузки данных, расчета аналитики, рейтинга и рекомендаций. Серверная часть выполняет обработку запроса, обращается к внешнему источнику, локальным файлам или базе данных и возвращает результат клиенту в формате JSON.

После получения данные приводятся к единому формату. Это необходимо для того, чтобы сведения из разных источников могли одинаково использоваться при расчете эффективности игрока, построении визуализаций и работе ML-модуля. Обработанные данные сохраняются в локальной базе SQLite, что позволяет повторно использовать их без постоянного обращения к внешним сервисам.

Схема получения доступа к данным в системе RateMyPlayer представлена на рисунке 11.

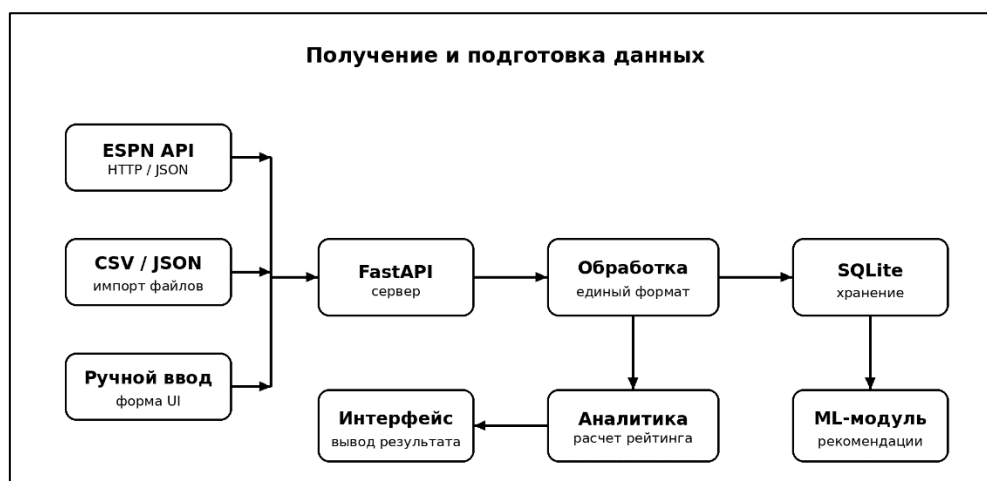


Рисунок 11 – Технологии получения доступа к данным в системе RateMyPlayer

Таким образом, доступ к данным в системе обеспечивается за счет сочетания HTTP-запросов к ESPN Public API, импорта CSV- и JSON-файлов, REST API для взаимодействия клиента и сервера, а также сохранения информации в локальной базе данных.

2.4 Организация хранения данных

Для корректной работы системы RateMyPlayer необходимо обеспечить организованное хранение данных, полученных из внешних источников, локальных файлов и пользовательского интерфейса. Хранение данных требуется для повторного использования информации об игроках, сохранения статистических показателей, построения карты бросков, расчета эффективности и передачи данных в ML-модуль.

В качестве основного средства хранения в системе используется локальная база данных SQLite. Данный вариант подходит для разрабатываемого приложения, поскольку SQLite не требует отдельного серверного развертывания, хранит данные в одном файле и удобна для учебного программного проекта. База данных располагается в каталоге проекта и используется серверной частью приложения для сохранения и получения информации.

Работа с базой данных организована через SQLAlchemy. Использование данной библиотеки позволяет описывать структуру таблиц в виде Python-

классов и выполнять операции добавления, получения, изменения и удаления данных средствами ORM. Такой подход упрощает взаимодействие с базой данных и снижает объем прямых SQL-запросов в коде приложения.

Схема организации хранения данных в системе RateMyPlayer представлена на рисунке 12.

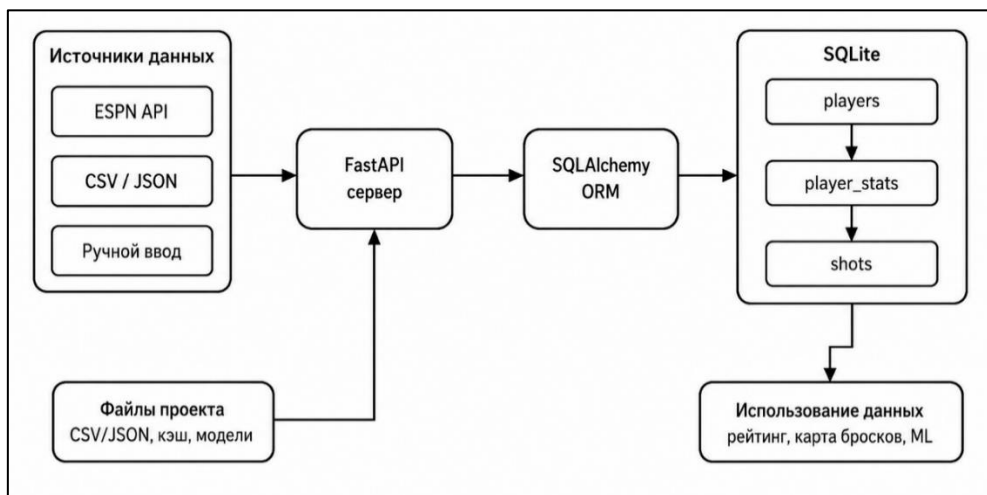


Рисунок 12 – Организация хранения данных в системе RateMyPlayer

Структура базы данных построена вокруг карточки игрока. С ней связаны записи сезонной статистики и сведения о бросковых попытках. Такое разделение позволяет хранить постоянную информацию о спортсмене отдельно от изменяемых игровых данных. Профиль игрока используется как основная сущность, сезонная статистика применяется при расчете рейтинга, а данные о бросках используются для анализа зон площадки и построения визуализации. Благодаря связям между таблицами система может получать полный набор информации по выбранному баскетболисту без дублирования записей.

Организация хранения данных построена таким образом, чтобы между основными сущностями сохранялись связи. Карточка игрока является основной записью, с которой связаны его статистические показатели и броски. Это позволяет получать полную информацию по конкретному спортсмену, анализировать его показатели за определенный сезон и использовать связанные данные для построения визуализаций.

Кроме таблиц SQLite, в проекте используются служебные файловые каталоги. Они не заменяют базу данных, а дополняют ее: в одних хранятся исходные наборы для загрузки, в других — временные ответы внешнего источника, в третьих — артефакты ML-модуля. Такое разделение помогает не смешивать постоянные записи приложения, промежуточные файлы и результаты обучения модели.

Организованное хранение данных позволяет системе выполнять анализ без необходимости повторной загрузки информации из внешних источников. После записи информации в базу пользователь может вернуться к карточке игрока без повторной загрузки данных из ESPN или файлов. В интерфейсе остаются доступны профиль спортсмена, сезонная статистика, карта бросков и рассчитанные аналитические показатели. Это упрощает работу с приложением: уже добавленные данные можно использовать повторно при просмотре, сравнении игроков и формировании рекомендаций.

Такая организация хранения нужна не только для сохранения записей, но и для поддержки дальнейшего анализа. Сервер может получить данные по выбранному игроку, использовать их при расчете рейтинга, построении карты бросков и передаче признаков в ML-модуль без повторной загрузки и ручной подготовки.

2.5 Подготовка данных к дальнейшему использованию

Данные, полученные из ESPN Public API, CSV- и JSON-файлов или введенные пользователем вручную, перед использованием проходят подготовку. Без этого этапа возможны ошибки в расчетах: некорректный процент бросков, пустая позиция игрока, неверный тип значения или координата броска вне площадки могут повлиять на рейтинг, карту бросков и работу ML-модуля.

На этапе проверки система оценивает, достаточно ли данных для создания карточки игрока, добавления сезонной статистики и построения карты бросков. Для каждой группы сведений проверяется наличие обязательных полей и корректность их структуры. Если запись неполная или

не может быть связана с конкретным игроком, она не используется либо обрабатывается с предупреждением.

После проверки структуры выполняется очистка. Из текстовых полей удаляются лишние пробелы, пустые значения заменяются допустимыми вариантами или пропускаются. Например, запись без имени игрока нельзя связать с карточкой спортсмена, поэтому она не должна попадать в базу. Если не указана команда или позиция, система может записать значение «Не указана», чтобы сохранить целостность данных и не прерывать импорт.

Затем значения приводятся к нужным типам. Игровые показатели — очки, передачи, подборы, перехваты, блок-шоты и потери — преобразуются в числовой формат, возраст сохраняется как целое число, а текстовые поля остаются строками. Отдельно обрабатываются проценты реализации бросков. Если значение передано как 45, система приводит его к виду 0,45, так как дальнейшие расчеты используют доли в диапазоне от 0 до 1.

Следующий шаг — проверка допустимых диапазонов. Возраст, рост и вес не должны принимать нереалистичные значения, проценты реализации не могут выходить за пределы от 0 до 1, а координаты бросков должны соответствовать границам баскетбольной площадки. Такая проверка снижает риск ошибок при построении карты бросков и расчете эффективности игрока.

После очистки и проверки данные связываются между собой. Основной записью является карточка игрока, к которой привязываются сезонная статистика и броски. Благодаря этому система может получить полный набор сведений по конкретному спортсмену: профиль, показатели за сезон, бросковую карту, рассчитанный рейтинг и признаки для ML-модуля.

После подготовки данные приводятся к единому виду и могут использоваться в дальнейших расчетах без дополнительной ручной обработки. Это снижает риск ошибок при отображении статистики, построении карты бросков и формировании признаков для ML-модуля.

2.6 Выводы по разделу

Во втором разделе определен порядок работы с данными в RateMyPlayer. Для анализа используются сведения об игроке, сезонная статистика и данные о бросках, которые могут поступать из ESPN Public API, локальных файлов или вводиться пользователем вручную.

После получения данные проходят проверку, очистку и приведение к единому формату. Это необходимо для того, чтобы одна и та же информация корректно использовалась в разных частях системы: при отображении статистики, построении карты бросков, сравнении игроков и подготовке признаков для ML-модуля.

В результате была сформирована логика работы с данными: получение из выбранного источника, сохранение в базе, проверка корректности и дальнейшее использование в аналитических функциях приложения.

3 Выбор средств разработки

3.1 Выбор языка программирования

Выбор языка программирования является важным этапом разработки системы, поскольку от него зависит удобство реализации серверной части, пользовательского интерфейса, базы данных, модулей обработки данных и машинного обучения.

Для разработки системы RateMyPlayer был выбран язык программирования Python. Данный язык широко применяется в задачах анализа данных, машинного обучения, разработки серверных приложений и создания прикладного программного обеспечения. Это делает его подходящим для реализации всех основных компонентов разрабатываемой системы.

Одним из главных преимуществ Python является наличие большого количества библиотек. Для обработки данных могут использоваться pandas и NumPy, для машинного обучения — scikit-learn и PyTorch, для серверной части — FastAPI, для графического интерфейса — PySide6, а для работы с базой данных — SQLAlchemy и SQLite.

Система RateMyPlayer включает клиентскую часть, серверную часть, базу данных, модуль обработки статистики, ML-модуль и модуль рекомендаций. Использование Python позволяет реализовать все эти компоненты в единой технологической среде, что упрощает разработку, отладку и дальнейшее сопровождение проекта.

Сравнение возможных языков программирования для разработки системы представлено в таблице 2.

Таблица 2 – Сравнение языков программирования для разработки системы RateMyPlayer

Параметр сравнения	Python	Java	C++
Анализ данных и ML	Высокая поддержка	Средняя поддержка	Ограниченно

Продолжение таблицы 2.

Параметр сравнения	Python	Java	C++
Серверная часть	FastAPI, Uvicorn	Spring, Spark	Требует больше настройки
Графический интерфейс	PySide6	JavaFX, Swing	Qt
Работа с БД	SQLite, SQLAlchemy	JDBC, ORM	SQL-библиотеки
Скорость разработки	Высокая	Средняя	Низкая
Подход к проекту	Наиболее подходит	Подходит частично	Избыточен

На основе проведенного сравнения можно сделать вывод, что Python является наиболее подходящим языком программирования для разработки системы RateMyPlayer. Он обеспечивает необходимые средства для работы с данными, машинного обучения, серверной логики, пользовательского интерфейса и базы данных.

3.2 Выбор среды разработки

Выбор среды разработки влияет на удобство написания программного кода, отладки, запуска отдельных модулей, работы с зависимостями и сопровождения проекта. RateMyPlayer состоит из нескольких связанных частей: серверного приложения, графического интерфейса, базы данных, модуля обработки статистики и ML-модуля. Поэтому среда разработки должна позволять удобно работать со структурой Python-проекта, запускать отдельные компоненты, отлаживать код и управлять зависимостями.

При выборе среды разработки рассматривались PyCharm, Visual Studio Code и Jupyter Notebook. Эти инструменты поддерживают работу с Python, однако рассчитаны на разные задачи. Для RateMyPlayer требовалась среда, в которой удобно вести проект с несколькими модулями: серверной частью,

графическим интерфейсом, базой данных, обработкой статистики и ML-компонентом.

PyCharm подходит для такой структуры лучше всего, поскольку ориентирован именно на Python-разработку. В нем удобно просматривать дерево проекта, быстро переходить между файлами, запускать отдельные части приложения, работать с виртуальным окружением и выполнять отладку. Это упрощает разработку RateMyPlayer, где код распределен между сервером, клиентской частью, базой данных и модулем машинного обучения.

Visual Studio Code также подходит для Python-разработки, но требует дополнительной настройки расширений. Данная среда является гибким редактором кода с большим количеством расширений, встроенным терминалом и поддержкой Git. Однако для полноценной работы с Python-проектом требуется установка и настройка дополнительных расширений, тогда как PyCharm изначально ориентирован на разработку Python-приложений.

Jupyter Notebook удобен для анализа данных и проведения экспериментов с моделями машинного обучения. Его можно использовать как вспомогательный инструмент при исследовании данных, проверке признаков и тестировании отдельных алгоритмов. Однако он не подходит в качестве основной среды разработки клиент-серверного приложения с графическим интерфейсом и базой данных.

Сравнение рассмотренных сред разработки представлено в таблице 3.

Таблица 3 — Сравнение сред разработки

Среда разработки	Назначение	Основные преимущества	Вывод для проекта
PyCharm	Основная среда для Python-проекта	Поддержка Python, отладчик, виртуальные окружения, удобная структура проекта	Выбрана для разработки RateMyPlayer

Продолжение таблицы 3.

Среда разработки	Назначение	Основные преимущества	Вывод для проекта
Visual Studio Code	Универсальный редактор кода	Гибкость настройки, расширения, терминал, поддержка Git	Может использоваться как альтернатива
Jupyter Notebook	Инструмент для анализа данных и ML-экспериментов	Пошаговый запуск кода, графики, проверка моделей	Используется как вспомогательный инструмент

На основе проведенного сравнения в качестве основной среды разработки была выбрана PyCharm. Данный выбор обусловлен тем, что PyCharm является специализированной средой для Python, обеспечивает удобную работу со структурой проекта, поддерживает отладку, виртуальные окружения и запуск отдельных компонентов системы. Это делает ее наиболее подходящей для разработки и сопровождения программной системы RateMyPlayer.

3.3 Выбор системы хранения данных

Для RateMyPlayer требовалось локальное хранилище, которое поддерживает связи между карточкой игрока, сезонной статистикой и бросковыми попытками. База данных должна была работать вместе с серверной частью приложения, быстро возвращать сведения по выбранному спортсмену и не требовать отдельного развертывания удаленной СУБД. Такой вариант подходит для проекта, где данные загружаются из файлов или внешнего источника, сохраняются внутри приложения и затем повторно используются при построении рейтинга, карты бросков и рекомендаций.

Для реализации хранения данных была выбрана SQLite. Данная система управления базами данных подходит для учебного программного проекта, поскольку не требует отдельной установки серверного программного обеспечения, хранит данные в одном файле и легко интегрируется с Python-

приложениями. Такой подход упрощает развертывание системы и позволяет использовать базу данных непосредственно внутри проекта.

Для RateMyPlayer система хранения должна поддерживать постоянное сохранение данных между запусками приложения и быстрое получение информации по выбранному игроку. Важно, чтобы серверная часть могла обращаться к сохраненным записям при расчете рейтинга, построении карты бросков и подготовке данных для ML-модуля без повторного импорта из файлов или внешнего API.

Для взаимодействия с базой данных используется SQLAlchemy. Данная библиотека позволяет описывать таблицы базы данных в виде Python-классов и выполнять операции с данными средствами ORM. Благодаря этому уменьшается необходимость написания большого количества прямых SQL-запросов, а работа с данными становится более удобной и понятной в рамках серверной части приложения.

В качестве альтернативных вариантов были рассмотрены PostgreSQL, MySQL, а также хранение данных только в CSV- и JSON-файлах. PostgreSQL и MySQL являются полноценными серверными СУБД и подходят для крупных многопользовательских систем, однако для локального учебного приложения они являются избыточными. CSV- и JSON-файлы удобны для импорта и экспорта данных, но не обеспечивают полноценную работу со связанными таблицами и не подходят в качестве основного хранилища. Сравнение рассмотренных средств хранения данных представлено в таблице 4.

Таблица 4 — Сравнение средств хранения данных

Средство хранения	Основные особенности	Применение в проекте
SQLite	Локальная БД в одном файле; не требует отдельного сервера; хорошо интегрируется с Python и SQLAlchemy.	Выбрана как основное хранилище данных об игроках, статистике и бросках.
PostgreSQL	Надежная серверная СУБД для больших и многопользовательских систем.	На текущем этапе избыточна; может использоваться при дальнейшем развитии системы.

Продолжение таблицы 4.

Средство хранения	Основные особенности	Применение в проекте
MySQL	Распространенная серверная СУБД для web-приложений.	Не выбрана, так как локальное приложение не требует отдельного сервера БД.
CSV/JSON-файлы	Простые форматы для импорта, экспорта и обмена данными.	Используются как дополнительные форматы загрузки, но не как основное хранилище.

На основе проведенного сравнения в качестве основного средства хранения данных была выбрана SQLite. Данное решение соответствует требованиям проекта RateMyPlayer, так как обеспечивает локальное хранение данных, не требует сложного развертывания, поддерживает SQL-запросы и хорошо интегрируется с Python и SQLAlchemy. CSV- и JSON-файлы при этом используются как дополнительные форматы для загрузки и обмена данными.

3.4 Выбор серверных технологий

Разрабатываемая система RateMyPlayer имеет клиент-серверную архитектуру, поэтому для ее реализации необходимо выбрать серверные технологии, обеспечивающие обработку запросов, взаимодействие с базой данных, получение спортивных данных из внешних источников и передачу результатов анализа в клиентское приложение.

Серверная часть системы должна выполнять несколько основных функций: принимать запросы от пользовательского интерфейса, обрабатывать данные игроков, обращаться к базе данных, выполнять расчет статистических показателей, взаимодействовать с ML-модулем и возвращать результаты клиентской части в удобном формате. Для решения этих задач требуется серверный фреймворк, поддерживающий создание API, работу с JSON-данными и интеграцию с Python-модулями.

В качестве основной серверной технологии был выбран FastAPI. Данный фреймворк предназначен для разработки web API на языке Python и хорошо подходит для построения серверной части приложения RateMyPlayer. FastAPI

позволяет создавать маршруты для работы с игроками, статистикой, внешними спортивными источниками и ML-модулем. Также он поддерживает автоматическую генерацию документации API, что упрощает проверку и тестирование серверных методов.

Для запуска серверного приложения используется Uvicorn. Он является ASGI-сервером и обеспечивает запуск FastAPI-приложения. Использование Uvicorn позволяет обрабатывать HTTP-запросы от клиентской части и передавать их соответствующим маршрутам серверного приложения.

Взаимодействие между клиентской и серверной частями организовано по принципу REST API. Клиентское приложение отправляет HTTP-запросы к серверу, а сервер обрабатывает их и возвращает ответ в формате JSON. Такой подход позволяет отделить пользовательский интерфейс от логики обработки данных. В результате клиентская часть отвечает за отображение информации и взаимодействие с пользователем, а серверная часть — за обработку данных, работу с базой данных и выполнение аналитических функций.

Для описания и проверки структуры передаваемых данных используется Pydantic. Он позволяет задавать схемы данных, проверять корректность входных параметров и формировать ответы API в едином виде. Это особенно важно при работе с данными игроков, сезонной статистикой, бросками и результатами анализа.

В качестве альтернативных серверных технологий были рассмотрены Flask и Django. Flask является простым и гибким фреймворком для создания web-приложений и API, однако для реализации валидации данных, документации и структуры проекта потребовалось бы больше ручной настройки. Django является более крупным фреймворком и подходит для сложных web-систем, но для локального API, обслуживающего desktop-приложение, он является избыточным.

Сравнение рассмотренных серверных технологий представлено в таблице 5.

Таблица 5 — Сравнение серверных технологий

Технология	Назначение	Итог для проекта
FastAPI	Разработка REST API серверной части.	Выбран как основной backend-фреймворк.
Uvicorn	Запуск FastAPI-приложения.	Используется для запуска сервера.
REST API	Обмен данными между клиентом и сервером.	Используется для связи PySide6-клиента с backend.
Flask	Создание небольших API.	Не выбран: требует больше ручной настройки.
Django	Разработка крупных web-приложений.	Не выбран: избыточен для локального desktop-проекта.

На основе проведенного сравнения для серверной части системы RateMyPlayer были выбраны FastAPI и Uvicorn. FastAPI используется для создания REST API и обработки запросов, а Uvicorn — для запуска серверного приложения. Данный выбор позволяет реализовать удобную, структурированную и расширяемую серверную часть, обеспечивающую взаимодействие между пользовательским интерфейсом, базой данных, внешними источниками данных и ML-модулем.

3.5 Выбор клиентских технологий

Клиентская часть системы RateMyPlayer предназначена для взаимодействия пользователя с программой. Через пользовательский интерфейс выполняется просмотр информации об игроках, добавление и редактирование данных, запуск анализа, отображение статистики, просмотр результатов расчета эффективности и получение рекомендаций. Поэтому выбранная клиентская технология должна обеспечивать создание удобного графического интерфейса и хорошо интегрироваться с серверной частью приложения.

Для реализации клиентской части была выбрана библиотека PySide6. Она представляет собой набор Python-привязок к фреймворку Qt и позволяет

создавать desktop-приложения с графическим интерфейсом. PySide6 подходит для данного проекта, так как дает возможность реализовать полноценное оконное приложение, использовать кнопки, таблицы, формы, вкладки, списки, графические элементы и другие компоненты интерфейса.

Важным преимуществом PySide6 является то, что клиентская часть может быть реализована на том же языке программирования, что и остальные части системы. Это упрощает разработку, так как серверная часть, обработка данных, ML-модуль и интерфейс создаются в единой Python-среде. Кроме того, PySide6 позволяет удобно организовать взаимодействие с серверной частью через HTTP-запросы к FastAPI.

В качестве альтернативных вариантов были рассмотрены PyQt5/PyQt6, Tkinter и web-интерфейс. PyQt имеет схожие возможности с PySide6, так как также основан на Qt, однако для проекта был выбран PySide6 как официальная Python-привязка к Qt. Tkinter входит в стандартную поставку Python и подходит для простых интерфейсов, но его возможностей недостаточно для удобного оформления более сложного приложения. Web-интерфейс позволил бы реализовать клиентскую часть в браузере, однако это потребовало бы отдельной разработки frontend-части с использованием HTML, CSS и JavaScript. Сравнение клиентских технологий представлено в таблице 6.

Таблица 6 – Сравнение клиентских технологий

Технология	Назначение	Оценка для проекта
PySide6	Создание desktop-интерфейса на Python/Qt.	Выбрана: подходит для локального приложения RateMyPlayer и интеграции с серверной частью.
PyQt5 / PyQt6	Создание графических приложений на Qt.	Не выбрана: возможности схожи с PySide6, но в проекте используется PySide6.
Tkinter	Создание простых интерфейсов на Python.	Не выбрана: недостаточно удобна для сложного интерфейса с вкладками и визуализацией.

Продолжение таблицы 6.

Технология	Назначение	Оценка для проекта
Web-интерфейс	Клиентская часть в браузере.	Не выбран: требует отдельной frontend-разработки; для проекта удобнее desktop-интерфейс.

На основе проведенного сравнения для разработки клиентской части системы RateMyPlayer была выбрана библиотека PySide6. Данный выбор обусловлен возможностью создания полноценного desktop-интерфейса, удобной интеграцией с Python-кодом, поддержкой графических компонентов и соответствием требованиям разрабатываемого приложения.

3.6 Выбор библиотек для обработки данных и машинного обучения

Для реализации системы RateMyPlayer необходимо выбрать библиотеки, обеспечивающие обработку спортивных данных, подготовку признаков, обучение моделей машинного обучения, сохранение обученных моделей и визуализацию результатов. Поскольку система разрабатывается на языке Python, выбор библиотек осуществлялся среди инструментов, которые хорошо интегрируются с Python-проектами и широко применяются в задачах анализа данных и машинного обучения.

Для обработки табличных данных в проекте используется библиотека pandas. Она позволяет загружать данные из CSV-файлов, выполнять фильтрацию, преобразование, очистку и подготовку наборов данных для дальнейшего анализа. В системе RateMyPlayer pandas применяется при импорте статистики, обработке табличных показателей и формировании признаков для ML-модуля.

Для выполнения числовых расчетов используется библиотека NumPy. NumPy используется для работы с числовыми данными и массивами. В проекте эта библиотека нужна при нормализации статистических показателей, подготовке обучающей выборки и выполнении вспомогательных расчетов.

Она хорошо сочетается с `pandas` и библиотеками машинного обучения, поэтому подходит для обработки признаков ML-модуля[16].

Основная модель машинного обучения в `RateMyPlayer` реализована с использованием `PyTorch`. В проекте применяется нейросетевая MLP-модель, которая обрабатывает подготовленные признаки игрока и рассчитывает вероятности возможных направлений развития. Модель использует многозадачный подход: общая нейросетевая часть формирует внутреннее представление игрового профиля, а отдельные выходные слои используются для оценки рекомендаций, общего рейтинга, атакующего и защитного профилей[17].

`Scikit-learn` в системе выполняет вспомогательную роль. Средства этой библиотеки применяются для подготовки признаков, масштабирования числовых данных, кодирования позиции игрока, разделения выборки и расчета метрик качества. Также на основе `scikit-learn` реализован дополнительный вариант MLP-модели, который используется для сравнения результатов и проверки работы ML-модуля[19].

Для сохранения обученных моделей применяется `joblib`. С ее помощью модель и связанные с ней объекты обработки данных сохраняются в файл и затем загружаются повторно без нового обучения. Это ускоряет запуск ML-модуля и делает работу системы удобнее.

Для построения графиков и отчетных материалов используется `matplotlib`. Библиотека подходит для отображения метрик качества, кривых обучения, сравнения моделей и визуального анализа статистики[20]. В `RateMyPlayer` она применяется при подготовке графиков, связанных с обучением и проверкой ML-модуля.

Также в проекте используется `Pillow`. Эта библиотека предназначена для базовой работы с изображениями и может применяться при обработке графических материалов, используемых в интерфейсе или отчетах.

Сравнение выбранных библиотек для обработки данных и машинного обучения представлено в таблице 7.

Таблица 7 — Библиотеки для обработки данных и машинного обучения

Библиотека	Назначение	Использование в RateMyPlayer
pandas	Работа с табличными данными	Импорт CSV, очистка данных, подготовка признаков
NumPy	Числовые вычисления	Расчеты признаков и подготовка данных для ML
scikit-learn	Машинное обучение	Обучение модели, нормализация, оценка качества
PyTorch	Нейросетевые модели	Дополнительное обучение MLP и графики метрик
joblib	Сохранение моделей	Загрузка и повторное использование обученных моделей
matplotlib	Визуализация данных	Графики обучения, метрик и результатов анализа
Pillow	Работа с изображениями	Вспомогательная обработка графических материалов

На основе проведенного анализа для системы RateMyPlayer были выбраны библиотеки pandas, NumPy, scikit-learn, PyTorch, joblib, matplotlib и Pillow. Данный набор инструментов обеспечивает полный цикл работы с данными: загрузку, очистку, преобразование, подготовку признаков, обучение моделей, сохранение результатов и визуализацию. Это позволяет реализовать ML-модуль и модуль анализа спортивной статистики в единой программной среде Python.

3.7 Выводы по разделу

В третьем разделе выбран стек разработки для RateMyPlayer. Основным языком стал Python, поскольку он подходит для серверной логики, обработки данных, машинного обучения и разработки интерфейса.

Для работы над проектом выбрана PyCharm. Хранение данных организуется в SQLite через SQLAlchemy, серверная часть строится на FastAPI и Uvicorn, а клиентский интерфейс реализуется с использованием PySide6.

Для анализа данных и ML-модуля используются pandas, NumPy, scikit-learn, PyTorch, joblib, matplotlib и Pillow. Такой набор инструментов закрывает

основные задачи проекта: загрузку данных, расчет признаков, обучение модели, сохранение результатов и визуализацию.

4. Проектирование и разработка программного обеспечения

4.1 Общая архитектура программной системы

Разрабатываемая система RateMyPlayer представляет собой программное приложение для оценки эффективности спортсмена в игровых видах спорта. В рамках данной работы основное внимание уделяется анализу баскетбольных данных, так как баскетбол имеет развитую систему статистических показателей, пригодных для программной обработки и дальнейшего применения методов машинного обучения.

Программная система построена по клиент-серверной архитектуре. Такой подход позволяет разделить пользовательский интерфейс, бизнес-логику, работу с базой данных и интеллектуальный модуль. Клиентская часть отвечает за взаимодействие с пользователем и отображение информации, а серверная часть выполняет обработку запросов, работу с данными, расчет показателей и обращение к ML-модулю.

Общая архитектура системы RateMyPlayer представлена на рисунке 13.

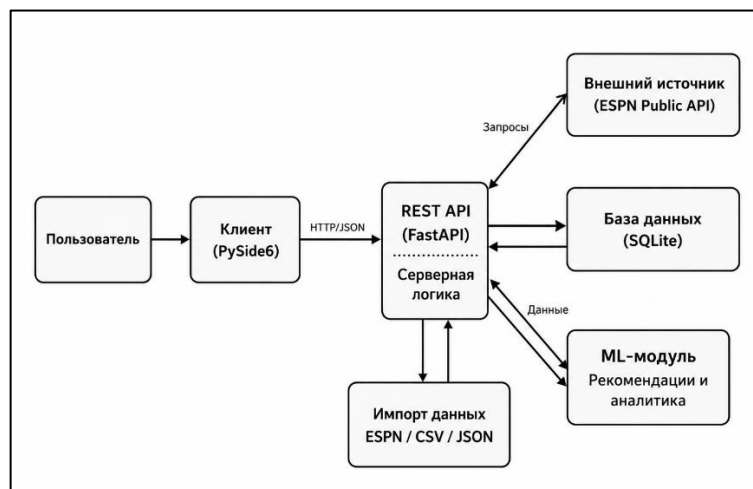


Рисунок 13 – Общая архитектура программной системы RateMyPlayer
В состав системы входят следующие основные компоненты:

- клиентская часть,
- серверная часть,
- база данных,
- модуль получения и импорта данных,
- модуль обработки спортивной статистики,

- ML-модуль оценки эффективности игрока,
- модуль формирования рекомендаций.

Клиентская часть в общей архитектуре выполняет роль слоя взаимодействия с пользователем. Она реализована как desktop-интерфейс на PySide6 и передает действия пользователя серверной части: выбор игрока, запуск операций с данными и запрос результатов анализа. Подробная структура интерфейса и порядок обмена с сервером рассматриваются далее в разделах разработки пользовательского интерфейса и клиент-серверной части.

Серверная часть реализована с использованием FastAPI и запускается с помощью Uvicorn. Она является центральным компонентом системы, поскольку принимает запросы от клиента, взаимодействует с базой данных, вызывает сервисы обработки данных, получает информацию из внешних источников и передает данные в ML-модуль. Серверная часть также обеспечивает единый формат обмена данными между компонентами системы.

База данных используется для хранения сведений об игроках, их статистике и бросках. В качестве системы хранения используется SQLite, а взаимодействие с ней выполняется через SQLAlchemy. Данные сохраняются локально, что позволяет повторно использовать информацию без необходимости постоянного обращения к внешним источникам.

Модуль получения и импорта данных обеспечивает загрузку информации из ESPN Public API, а также импорт подготовленных CSV- и JSON-файлов. После получения данные приводятся к внутренней структуре системы и сохраняются в базе данных. Такой подход позволяет использовать как внешние спортивные источники, так и заранее подготовленные наборы данных.

Модуль обработки спортивной статистики выполняет расчет показателей игрока, подготовку данных для визуализации, построение карты бросков и формирование аналитической информации. На основе сохраненной статистики система может рассчитывать эффективность игрока и представлять результаты пользователю в удобной форме.

ML-модуль используется для интеллектуальной оценки показателей игрока и формирования рекомендаций. Он получает подготовленные признаки, выполняет обработку данных и возвращает результат, который затем отображается в клиентском интерфейсе. Благодаря этому система не ограничивается простым хранением статистики, а выполняет интеллектуальный анализ игровых характеристик.

Таким образом, архитектура системы RateMyPlayer построена таким образом, чтобы разделить функции интерфейса, серверной логики, хранения данных, обработки статистики и машинного обучения. Это делает систему более структурированной, удобной для сопровождения и дальнейшего расширения.

4.2 Разработка пользовательского интерфейса

Пользовательский интерфейс системы RateMyPlayer разработан для удобного взаимодействия пользователя с основными функциями программы. Через интерфейс пользователь может просматривать список игроков, добавлять новые карточки, загружать статистические данные, получать информацию из внешнего источника ESPN, анализировать статистику, просматривать рейтинг, карту бросков и рекомендации, сформированные интеллектуальным модулем.

В пользовательском интерфейсе RateMyPlayer PySide6 используется уже на уровне реализации конкретных окон и виджетов. Основным элементом является главное окно, в котором объединены боковая панель выбора игрока, область сведений о спортсмене и вкладки с результатами анализа. Такая структура позволяет разделить навигацию, просмотр данных и запуск основных операций по разным зонам интерфейса.

Главное окно приложения построено на основе класса MainWindow, наследуемого от QMainWindow. В нем формируется общая структура интерфейса: боковая панель со списком игроков и кнопками управления, а также основная область отображения данных выбранного игрока. Главное окно системы RateMyPlayer представлено на рисунке 14.

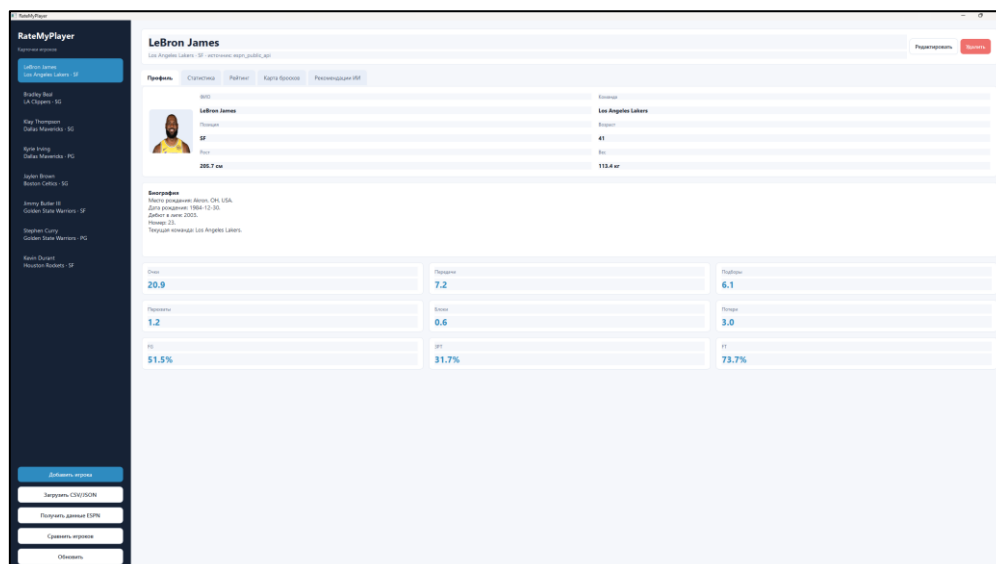


Рисунок 14 — Главное окно системы RateMyPlayer

Боковая панель предназначена для управления списком игроков. В ней отображаются карточки добавленных спортсменов и расположены основные кнопки: «Добавить игрока», «Загрузить CSV/JSON», «Получить данные ESPN», «Сравнить игроков» и «Обновить». Такое расположение элементов позволяет пользователю быстро выполнять основные действия, связанные с загрузкой, обновлением и сравнением данных.

Основная область интерфейса предназначена для отображения подробной информации о выбранном игроке. В верхней части окна выводятся имя игрока, команда, позиция и источник данных. Также предусмотрены кнопки редактирования и удаления карточки игрока.

Для структурированного представления информации используются вкладки:

- «Профиль»,
- «Статистика»,
- «Рейтинг»,
- «Карта бросков»,
- «Рекомендации ИИ».

Фрагмент создания вкладок пользовательского интерфейса представлен ниже.

```
self.tabs = QWidget()
```

```

self.profile_tab, self.profile_layout = self._scroll_tab()
self.stats_tab, self.stats_layout = self._scroll_tab()
self.rating_tab, self.rating_layout = self._scroll_tab()
self.shot_tab, self.shot_layout = self._scroll_tab()
self.recommendations_tab, self.recommendations_layout =
self._scroll_tab()

self.tabs.addTab(self.profile_tab, "Профиль")
self.tabs.addTab(self.stats_tab, "Статистика")
self.tabs.addTab(self.rating_tab, "Рейтинг")
self.tabs.addTab(self.shot_tab, "Карта бросков")
self.tabs.addTab(self.recommendations_tab, "Рекомендации
ИИ")

```

Вкладка «Профиль» предназначена для отображения общей информации об игроке: ФИО, команды, позиции, возраста, роста, веса, биографии и основных статистических показателей. Вкладка «Статистика» используется для просмотра сезонных показателей и анализа динамики результатов.

Вкладка «Рейтинг» отображает рассчитанную оценку эффективности игрока, отдельные статистические метрики и качественную интерпретацию результата. Вкладка «Карта бросков» предназначена для визуального отображения бросков игрока на баскетбольной площадке, что позволяет оценить распределение и результативность бросков по зонам.

Вкладка «Рекомендации ИИ» предназначена для вывода результатов интеллектуального анализа. Здесь пользователь видит ML-профиль игрока, вероятности по направлениям развития, текстовые рекомендации и план дальнейшей работы над игровыми показателями.

Для ввода и изменения данных используются отдельные диалоговые окна. В системе предусмотрены формы для создания карточки игрока, добавления сезонной статистики, внесения информации о броске и сравнения спортсменов. Это разгружает главное окно и делает работу с приложением более последовательной: каждое действие выполняется в своем окне.

Часть операций выполняется в фоновом режиме. Это относится к загрузке данных из ESPN, отправке запросов на сервер, получению рекомендаций и построению аналитики. Такие действия могут занимать время, поэтому в интерфейсе используются фоновые задачи и пул потоков. За счет этого окно приложения не зависает, а пользователь может продолжать работу.

Пользовательский интерфейс RateMyPlayer объединяет основные функции системы: работу с карточками игроков, загрузку данных, просмотр статистики, расчет рейтинга, отображение карты бросков и получение рекомендаций ML-модуля.

4.3 Разработка клиент-серверной части

Клиент-серверная часть является одной из основных составляющих системы RateMyPlayer, так как обеспечивает взаимодействие между пользовательским интерфейсом, серверной логикой, базой данных, внешними источниками данных и ML-модулем. Разделение приложения на клиентскую и серверную части позволяет отделить отображение данных от их обработки и сделать структуру проекта более удобной для сопровождения.

В клиент-серверном взаимодействии клиентское приложение не выполняет расчеты самостоятельно. Его задача — сформировать запрос по действию пользователя и передать его серверной части. После получения ответа интерфейс обновляет нужный экран: карточку игрока, статистику, рейтинг, карту бросков или рекомендации.

Серверная часть выступает точкой обработки таких запросов. В зависимости от выбранного маршрута она обращается к базе данных, сервисам импорта, модулю расчета статистики или ML-модулю, после чего возвращает клиенту подготовленный результат в едином формате. Такое разделение позволяет отделить отображение данных от их обработки и упростить дальнейшее расширение приложения.

Схема клиент-серверного взаимодействия представлена на рисунке 14.

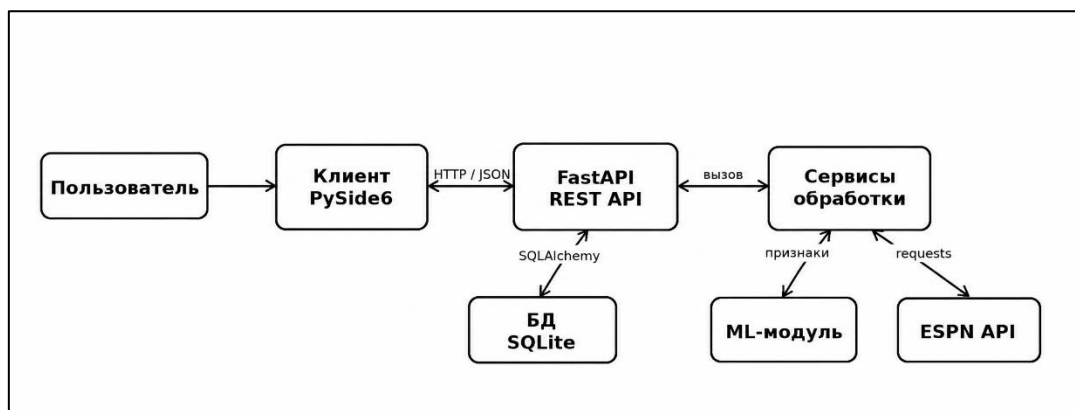


Рисунок 15 – Схема клиент-серверного взаимодействия в системе RateMyPlayer

Взаимодействие организовано следующим образом: пользователь выполняет действие в интерфейсе, клиентская часть отправляет HTTP-запрос к соответствующему endpoint серверной части, FastAPI-сервер обрабатывает запрос, обращается к базе данных или аналитическим модулям и возвращает результат в формате JSON. После этого клиентское приложение отображает полученные данные пользователю.

В серверной части создается объект FastAPI и подключаются основные маршруты приложения.

```

app = FastAPI(
    title="RateMyPlayer",
    description="Интеллектуальная система оценки баскетболиста с FastAPI backend.",
    version="0.1.0",
    lifespan=lifespan,
)

app.include_router(players_router)
app.include_router(espn_router)
app.include_router(ml_router)

```

Маршруты `players_router` отвечают за работу с игроками, статистикой, бросками, рейтингом, аналитикой и рекомендациями. Маршруты `espn_router` используются для получения данных из внешнего источника ESPN, а `ml_router` обеспечивает работу с ML-модулем.

Для проверки доступности серверной части используется endpoint `/health`, который позволяет убедиться, что сервер запущен и готов к обработке запросов.

```
@app.get("/health")
def health() -> dict:
    return {"status": "ok", "service": "ratemyplayer"}
```

Основная группа endpoint связана с обработкой данных игроков. Серверная часть позволяет получить список игроков, создать карточку игрока, получить аналитический отчет, рейтинг, карту бросков и рекомендации.

```
@router.get("", response_model=list[PlayerRead])
def api_list_players(db: Session = Depends(get_db)) -> list:
    return list_players(db)

@router.post("", response_model=PlayerRead, status_code=201)
def api_create_player(payload: PlayerCreate, db: Session =
Depends(get_db)):
    return create_player(db, payload)

@router.get("/{player_id}/analytics",
response_model=AnalyticsReport)
def api_get_analytics(player_id: int, db: Session =
Depends(get_db)):
    player = _get_player_or_404(db, player_id)
    return analyze_player(player, latest_stats(player))
```

Для получения рекомендаций используется отдельный endpoint. Он извлекает данные игрока из базы, определяет актуальную статистику, передает данные в рекомендательный модуль и возвращает результат клиентской части.

```
@router.get("/{player_id}/recommendations",
response_model=RecommendationResponse)
def api_get_recommendations(player_id: int, db: Session =
Depends(get_db)):
    player = _get_player_or_404(db, player_id)
    stats = latest_stats(player)
```

```

        recommender = HybridRecommender()
        return recommender.recommend_for_player(player, stats,
player.shots)

```

Клиентская часть взаимодействует с сервером через класс `ApiClient`. В нем задается базовый адрес сервера и реализуются методы, соответствующие основным действиям пользователя.

```

class ApiClient:
    def __init__(self, base_url: str =
"http://127.0.0.1:8000") -> None:
        self.base_url = base_url.rstrip("/")
        self.timeout = 12

    def list_players(self) -> list[dict]:
        return self._request("GET", "/players")

    def get_analytics(self, player_id: int) -> dict:
        return self._request("GET",
f"/players/{player_id}/analytics")

    def get_recommendations(self, player_id: int) -> dict:
        return self._request("GET",
f"/players/{player_id}/recommendations")

```

Непосредственная отправка HTTP-запросов выполняется с использованием библиотеки `requests`. При возникновении ошибки соединения или некорректном ответе сервера клиентская часть формирует исключение, которое затем может быть отображено пользователю в интерфейсе. Централизация запросов в классе `ApiClient` упрощает структуру клиентской части, так как остальные элементы интерфейса используют готовые методы и не обращаются к серверу напрямую.

Запуск серверной части выполняется с помощью `Uvicorn`. В проекте предусмотрен общий запуск приложения, при котором сначала стартует FastAPI-сервер, а затем открывается клиентское приложение. Это позволяет пользователю работать с системой как с единым программным продуктом.

Таким образом, клиент-серверная часть системы RateMyPlayer обеспечивает обмен данными между интерфейсом и основными функциональными модулями. Клиентская часть отвечает за действия пользователя и отображение результатов, а серверная часть выполняет обработку запросов, работу с базой данных, получение внешних данных, расчет аналитики и обращение к ML-модулю.

4.4 Разработка базы данных

В разработанной системе база данных описывает структуру хранения через связанные таблицы. Центральной сущностью выступает игрок, к которому привязываются сезонные показатели и бросковые попытки. За счет такой связи сервер может получить не разрозненные записи, а полный набор данных по выбранному спортсмену. Это важно для дальнейших расчетов: статистика используется при формировании рейтинга, а сведения о бросках — при построении карты площадки и подготовке данных для аналитических модулей.

В качестве системы хранения данных используется SQLite. База данных хранится в локальном файле `basketball_ai.sqlite3`, расположенном в каталоге `data`. Такой вариант подходит для разрабатываемого приложения, поскольку SQLite не требует развертывания отдельного сервера базы данных и позволяет использовать систему как локальное desktop-приложение с серверной частью.

Подключение к базе данных реализовано в модуле `server/db/database.py`. В нем задается путь к файлу базы данных, формируется строка подключения и создается объект `engine`, через который SQLAlchemy взаимодействует с SQLite.

```
DATABASE_PATH = DATA_DIR / "basketball_ai.sqlite3"
DATABASE_URL = os.getenv(
    "BASKETBALL_AI_DATABASE_URL",
    f"sqlite:/// {DATABASE_PATH.as_posix()} "
)

engine = create_engine(DATABASE_URL, **engine_kwargs)
```

```

SessionLocal = sessionmaker(
    bind=engine,
    autoflush=False,
    autocommit=False,
    future=True
)

```

Использование SQLAlchemy позволяет описывать таблицы базы данных в виде Python-классов, задавать связи между ними и выполнять операции добавления, получения, изменения и удаления данных средствами ORM. Это снижает количество прямых SQL-запросов в коде и делает работу с данными более структурированной.

Структура базы данных системы RateMyPlayer представлена на рисунке 16.

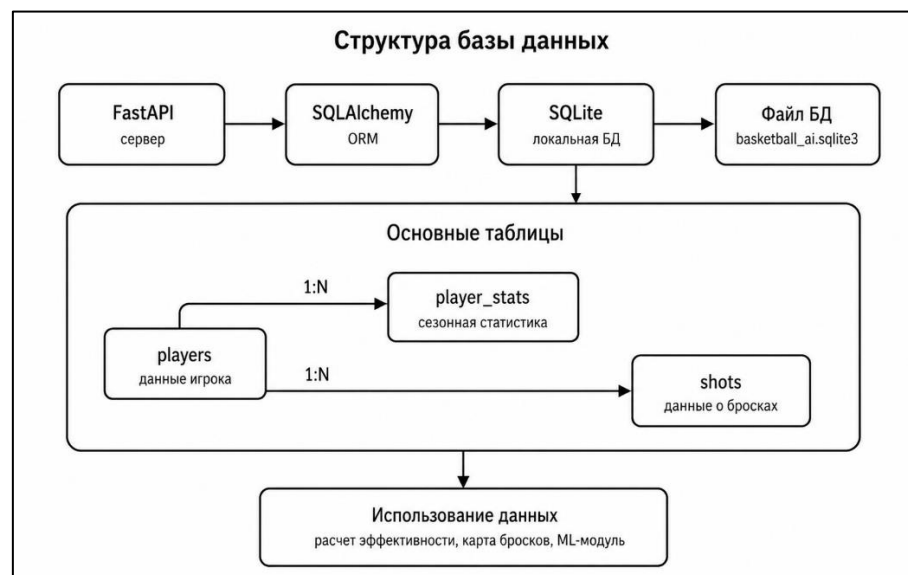


Рисунок 16 – Структура базы данных системы RateMyPlayer

В базе данных реализованы три основные таблицы:

- players,
- player_stats,
- shots.

Таблица players является основной таблицей базы данных. Она предназначена для хранения общей информации об игроках. В ней сохраняются идентификатор игрока, внешний идентификатор, ФИО, команда,

позиция, возраст, рост, вес, биография, путь к изображению, источник получения данных и дата создания записи. Каждая запись данной таблицы соответствует одному игроку.

Фрагмент описания модели игрока в файле `server/db/models.py` представлен ниже.

```
class Player(Base):
    __tablename__ = "players"

    id: Mapped[int] = mapped_column(Integer,
primary_key=True, index=True)
    external_id: Mapped[str | None] = mapped_column(String(60), nullable=True, index=True)
    full_name: Mapped[str] = mapped_column(String(160),
nullable=False, index=True)
    team: Mapped[str] = mapped_column(String(120),
default="Не указана")
    position: Mapped[str] = mapped_column(String(60),
default="Не указана")
    age: Mapped[int | None] = mapped_column(Integer,
nullable=True)
    height: Mapped[float | None] = mapped_column(Float,
nullable=True)
    weight: Mapped[float | None] = mapped_column(Float,
nullable=True)
    biography: Mapped[str | None] = mapped_column(Text,
nullable=True)
    source: Mapped[str] = mapped_column(String(60),
default="manual")
```

Таблица `player_stats` предназначена для хранения сезонной статистики игрока. Она связана с таблицей `players` через внешний ключ `player_id`. В данной таблице сохраняются сезон, очки, передачи, подборы, перехваты, блок-шоты, потери, процент попаданий с игры, процент трехочковых бросков и процент

штрафных бросков. Эти показатели используются при расчете эффективности игрока, построении аналитики и формировании признаков для ML-модуля.

```
class PlayerStats(Base):
    __tablename__ = "player_stats"

    id: Mapped[int] = mapped_column(Integer,
primary_key=True, index=True)
    player_id: Mapped[int] = mapped_column(
        ForeignKey("players.id", ondelete="CASCADE"),
        index=True
    )
    season: Mapped[str] = mapped_column(String(40),
default="2025/26")
    points: Mapped[float] = mapped_column(Float, default=0.0)
    assists: Mapped[float] = mapped_column(Float,
default=0.0)
    rebounds: Mapped[float] = mapped_column(Float,
default=0.0)
    steals: Mapped[float] = mapped_column(Float, default=0.0)
    blocks: Mapped[float] = mapped_column(Float, default=0.0)
    turnovers: Mapped[float] = mapped_column(Float,
default=0.0)
    fg_pct: Mapped[float] = mapped_column(Float, default=0.0)
    three_pct: Mapped[float] = mapped_column(Float,
default=0.0)
    ft_pct: Mapped[float] = mapped_column(Float, default=0.0)
```

Таблица shots предназначена для хранения данных о бросках игрока. Она также связана с таблицей players через поле player_id. В таблице сохраняются сезон, координаты броска на площадке, результат попытки, тип броска, четверть, время игрового эпизода и дата матча. Эти данные используются для построения карты бросков и анализа результативности игрока в различных зонах площадки.

```
class Shot(Base):
    __tablename__ = "shots"
```

```

        id: Mapped[int] = mapped_column(Integer,
primary_key=True, index=True)
        player_id: Mapped[int] = mapped_column(
            ForeignKey("players.id", ondelete="CASCADE"),
            index=True
        )
        season: Mapped[str] = mapped_column(String(40),
default="2025/26", index=True)
        x: Mapped[float] = mapped_column(Float, nullable=False)
        y: Mapped[float] = mapped_column(Float, nullable=False)
        result: Mapped[str] = mapped_column(String(20),
nullable=False)
        shot_type: Mapped[str] = mapped_column(String(12),
nullable=False)
        quarter: Mapped[int | None] = mapped_column(Integer,
nullable=True)
        time: Mapped[str | None] = mapped_column(String(16),
nullable=True)
        game_date: Mapped[str | None] = mapped_column(String(20),
nullable=True, index=True)

```

Между таблицами базы данных реализованы связи типа «один ко многим». Один игрок может иметь несколько записей сезонной статистики и несколько записей о бросках. Для этого в модели Player задаются связи со статистикой и бросками. При удалении игрока связанные с ним данные также удаляются, что позволяет сохранять целостность базы данных.

```

stats: Mapped[list["PlayerStats"]] = relationship(
    "PlayerStats",
    back_populates="player",
    cascade="all, delete-orphan"
)

shots: Mapped[list["Shot"]] = relationship(
    "Shot",

```

```

        back_populates="player",
        cascade="all, delete-orphan"
    )

```

Инициализация базы данных выполняется при запуске серверной части приложения. Для этого используется метод `create_all`, который создает таблицы на основе описанных ORM-моделей. Также в проекте предусмотрена возможность расширения структуры базы данных при дальнейшем развитии системы.

```

def init_db() -> None:
    Base.metadata.create_all(bind=engine)
    run_lightweight_migrations()

```

Таким образом, разработанная структура базы данных обеспечивает хранение всех основных данных, необходимых для работы системы RateMyPlayer. Таблица `players` содержит сведения об игроках, таблица `player_stats` хранит сезонные статистические показатели, а таблица `shots` используется для хранения бросков и построения карты бросков. Связи между таблицами позволяют получать полную информацию по каждому игроку и использовать ее в модулях обработки данных, расчета эффективности, визуализации и машинного обучения.

4.5 Разработка модуля обработки спортивных данных

Модуль обработки спортивных данных в системе RateMyPlayer используется после того, как сведения об игроке, сезонной статистике и бросках уже сохранены в базе данных. На этом этапе данные не загружаются заново, а извлекаются серверной частью и передаются в отдельные сервисы, отвечающие за отображение статистики, построение сравнительного рейтинга, анализ бросков и подготовку признаков для ML-модуля.

Схема обработки спортивных данных в системе RateMyPlayer представлена на рисунке 17.

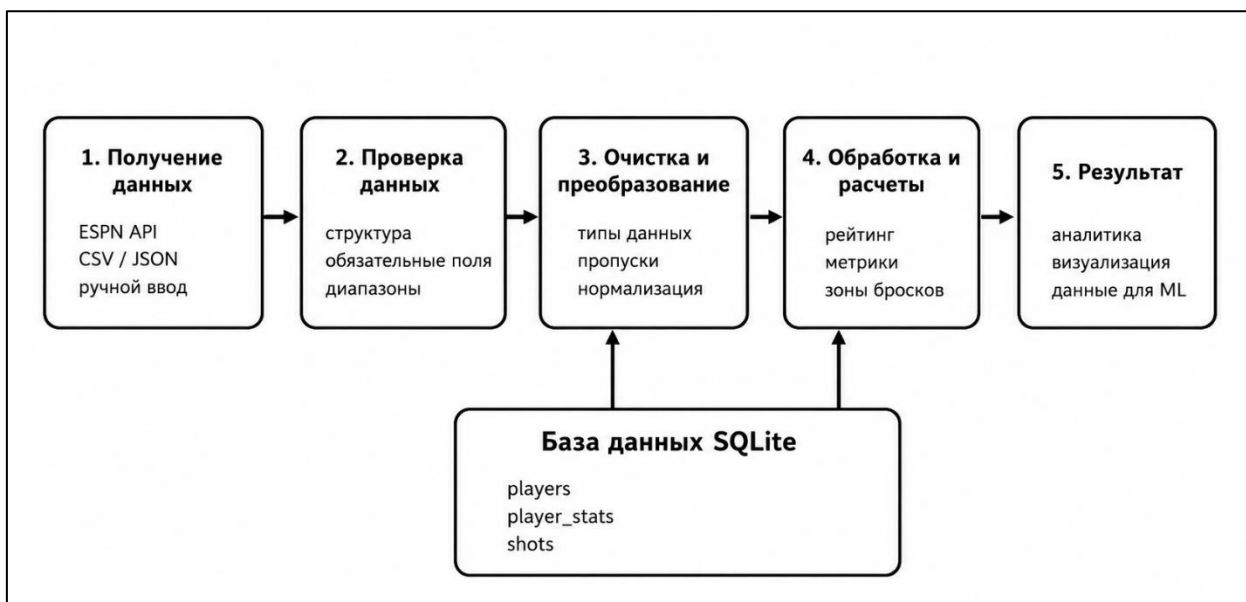


Рисунок 17 – Схема обработки спортивных данных в системе RateMyPlayer

Для отображения статистики игрока используется сервис сравнения показателей. Он получает сохраненную сезонную статистику, выбирает нужный сезон и сопоставляет показатели игрока со справочными средними значениями. В проекте это реализовано в функции `build_stats_comparison`, расположенной в файле `server/services/league_average_service.py`.

```
def build_stats_comparison(player: Player, season: str | None = None) -> dict[str, Any]:
```

Результат работы данного сервиса используется во вкладке «Статистика». Пользователь видит не только значения по выбранному игроку, но и сравнение с условным средним уровнем. За счет этого отдельные показатели получают контекст: можно понять, находится игрок выше, ниже или примерно на уровне среднего значения.

Отдельно реализована обработка динамики статистики. За нее отвечает функция `build_stats_trend` из файла `server/services/stats_trend_service.py`. Она формирует данные для графика изменения выбранного показателя. Если пользователь выбирает период «вся карьера», точки строятся по сезонам. Если выбран конкретный сезон, система использует доступные сведения о бросках или строит оценочную помесечную динамику вокруг сезонного среднего.

```
def build_stats_trend(player: Player, metric: str = "points",
period: str = "career") -> dict[str, Any]:
```

Такой подход позволяет использовать даже неполные данные. Если в базе нет детальной статистики по каждому матчу, система все равно может показать примерную динамику, но при этом источник данных указывается в ответе сервера. Это важно, так как пользователь должен понимать, какие графики построены по фактическим данным, а какие имеют оценочный характер.

Для вкладки «Рейтинг» используется отдельный сервис `rating_service.py`. Он не рассчитывает формульную оценку эффективности, а формирует сравнительный рейтинг игрока относительно других спортсменов. Основная функция сервиса — `build_player_rating`.

```
def feature_row_from_stats(stats: Any, shots: Iterable[Any],
position: str,
history: Iterable[Any] | None = None) -> dict[str, Any]:
```

В набор признаков входят исходные игровые показатели, сведения о бросках, тренды, сравнение со средними значениями и производные характеристики. Дополнительно рассчитываются отношение передач к потерям, бросковая точность, защитная активность, атакующая нагрузка и вспомогательный показатель эффективности. Эти признаки затем используются в ML-модуле для анализа игрового профиля и формирования рекомендаций.

Таким образом, пункт обработки спортивных данных в `RateMyPlayer` описывает не один отдельный расчет, а слой серверных сервисов, который связывает сохраненную статистику с аналитическими функциями приложения. На этом уровне данные подготавливаются для вкладок «Статистика», «Рейтинг» и «Карта бросков», а также преобразуются в признаки для ML-модуля. За счет такого разделения интерфейс отвечает за

отображение результата, база данных — за хранение сведений, а серверные сервисы — за обработку и подготовку данных к дальнейшему анализу.

4.6 Разработка ML-модуля оценки эффективности игрока

ML-модуль в RateMyPlayer отвечает за ту часть анализа, где одних расчетных формул уже недостаточно. Он получает статистику баскетболиста, данные о бросках и позицию игрока, после чего преобразует их в признаки для модели. В этот набор входят не только базовые показатели вроде очков, передач, подборов и потерь, но и производные значения: отношение передач к потерям, бросковая точность, защитная активность и распределение бросков по типам.

Работа модуля связана с двумя другими частями системы: обработкой спортивных данных и формированием рекомендаций. Сначала сервер извлекает из базы нужную информацию по выбранному игроку. Затем подготовленные данные передаются в модель машинного обучения, которая рассчитывает вероятности по возможным направлениям развития. После этого результат дополняется экспертными правилами: они помогают учесть очевидные случаи, например низкий процент штрафных, частые потери или слабую защитную активность.

Схема работы ML-модуля представлена на рисунке 18.

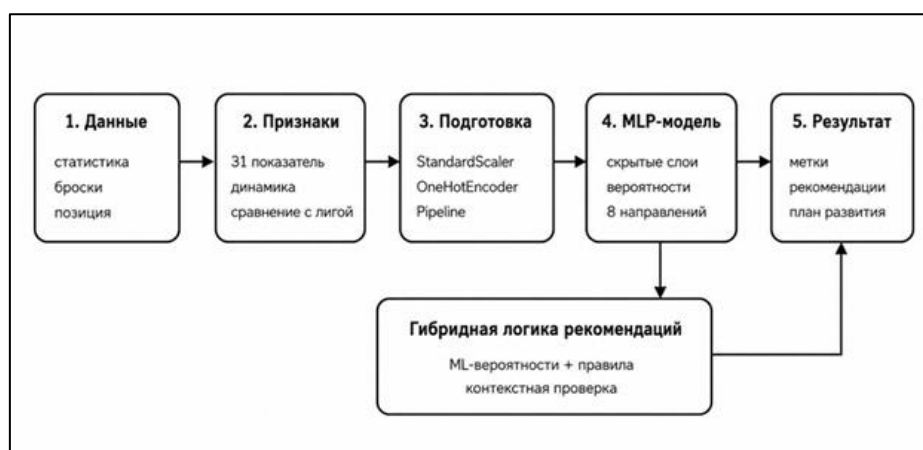


Рисунок 18 – Схема работы ML-модуля системы RateMyPlayer

На уровне реализации для этой задачи выбран нейросетевой вариант многослойного персептрона на PyTorch. После подготовки признаков входной

вектор передается в полносвязные слои модели, где формируется внутреннее представление игрового профиля[18]. Далее отдельные выходы модели рассчитывают вероятности по каждому направлению развития. Такая структура подходит для многометочной классификации, поскольку рекомендации не исключают друг друга и могут выдаваться одновременно.

В системе предусмотрены следующие направления рекомендаций:

- улучшение трехочкового броска,
- снижение количества потерь,
- улучшение штрафных бросков,
- увеличение атак в краску,
- развитие защитной работы,
- улучшение принятия решений,
- развитие физической готовности,
- развитие паса и видения площадки.

Для работы ML-модуля используется не исходная статистика в разрозненном виде, а единый вектор признаков, сформированный после предварительной обработки данных. В него входят несколько групп характеристик: сезонные показатели игрока, сведения о бросковых попытках, динамика изменения отдельных метрик и сравнение с условными средними значениями. Такой формат позволяет модели анализировать игровой профиль не по одному параметру, а по совокупности признаков, отражающих атакующие действия, защитную активность, качество владения мячом и особенности бросковой карты.

Для приведения итоговых рейтинговых значений к единой шкале используется функция ограничения(1):

$$C(x) = \min(100, \max(0, x)), \quad (1)$$

где x — рассчитанное значение показателя,

$C(x)$ — значение, приведенное к диапазону от 0 до 100.

Использование данной функции позволяет отображать атакующий рейтинг, защитный рейтинг, общую эффективность и итоговый профиль игрока в единой шкале.

Атакующий рейтинг игрока рассчитывается по формуле 2:

$$O = C(P * 2,0 + AST * 3,0 + FG * 25,0 + 3PT * 18,0 + FT * 10,0 - TOV * 3,5), \quad (2)$$

где O — атакующий рейтинг;

P — очки;

AST — передачи;

FG — процент попаданий с игры;

$3PT$ — процент трехочковых бросков;

FT — процент штрафных бросков;

TOV — потери.

Данная формула отражает вклад игрока в нападение. Очки и передачи имеют положительные веса, так как напрямую связаны с созданием результата команды. Проценты реализации бросков имеют увеличенные коэффициенты, поскольку сами значения представлены в диапазоне от 0 до 1 и без дополнительных весов оказывали бы слишком слабое влияние на итоговый показатель. Потери учитываются с отрицательным коэффициентом, так как они приводят к завершению владения без результата.

Защитный рейтинг рассчитывается по формуле 3:

$$D = C(REB * 3,0 + STL * 12,0 + BLK * 10,0), \quad (3)$$

где D — защитный рейтинг;

REB — подборы;

STL — перехваты;

BLK — блок-шоты.

В данной формуле подборы имеют меньший вес, так как являются более частым статистическим действием. Перехваты и блок-шоты получают больший вес, поскольку они напрямую отражают активное защитное воздействие на атаку соперника и встречаются реже.

Общая эффективность игрока рассчитывается по формуле 4:

$$E = C(P + REB * 1,2 + AST * 1,4 + STL * 3,0 + BLK * 2,5 - TOV * 2,2 + FG * 20,0 + 3PT * 10,0 + FT * 8,0), \quad (4)$$

где E — показатель общей эффективности.

Данная формула объединяет атакующие, защитные и бросковые характеристики игрока. Потери уменьшают эффективность, поскольку отрицательно влияют на качество владений.

Итоговый рейтинг игрока формируется как взвешенная сумма отдельных компонентов(5):

$$R = C(O * 0,45 + D * 0,25 + E * 0,20 + S * 0,10), \quad (5)$$

где R — общий рейтинг игрока;

O — атакующий рейтинг;

D — защитный рейтинг;

E — общая эффективность;

S — стабильность.

В итоговой оценке больший вклад отводится атакующему рейтингу — 0,45. Это связано с тем, что в доступной баскетбольной статистике лучше всего представлены именно атакующие действия: набранные очки, передачи и точность бросков. Защитный рейтинг имеет вес 0,25, поскольку защита влияет на общую полезность игрока, но описывается меньшим числом показателей. Общая эффективность учитывается с весом 0,20 и помогает объединить разные игровые действия в один дополнительный показатель. Стабильность

получила вес 0,10, так как она используется как корректирующий фактор и не должна перекрывать основные игровые характеристики.

Перед передачей данных в ML-модуль дополнительно рассчитываются производные признаки. Один из них — отношение результативных передач к потерям 6:

$$\frac{AST}{TOV} = \frac{AST}{\max(TOV, 0,15)}, \quad (6)$$

AST/TOV показывает, насколько аккуратно игрок распоряжается мячом. Если значение высокое, значит баскетболист чаще создает моменты для партнеров, чем теряет владение. Низкое значение, наоборот, указывает на проблемы с контролем мяча или выбором решения в атаке.

Также рассчитывается защитная активность 7:

$$DA = STL + BLK + REB * 0,18, \quad (7)$$

где DA — защитная активность.

В защитной активности учитываются перехваты, блок-шоты и часть подборов. Эти действия по-разному отражают работу игрока в обороне: перехват прерывает атаку соперника, блок-шот мешает завершить бросок, а подбор помогает вернуть владение своей команде.

Для построения ML-профиля игрока отдельно рассчитываются атакующий и защитный профили. Атакующий профиль определяется по формуле 8:

$$OP = C \left(\frac{P}{34} \right) * 40 + \left(\frac{AST}{11} \right) * 22 + \left(\frac{FG}{0,62} \right) * 20 + \left(\frac{3PT}{0,48} \right) * 10 + \\ + \left(\frac{\frac{AST}{TOV}}{4,5} \right) * 8), \quad (8)$$

где OP — атакующий профиль игрока.

В формуле атакующего профиля основной вклад дают очки, поскольку они напрямую отражают результативность игрока в нападении. Передачи учитываются следующим по значимости компонентом: они показывают, насколько игрок участвует в создании моментов для партнеров. Процент попаданий с игры нужен для оценки качества завершения атак. Трехочковый процент и отношение передач к потерям имеют меньший вес, но позволяют точнее описать стиль игрока: его угрозу с дальней дистанции и аккуратность при владении мячом.

Защитный профиль рассчитывается по формуле 9:

$$DP = C \left(\frac{REB}{15} \right) * 40 + \left(\frac{STL}{2,8} \right) * 25 + \left(\frac{BLK}{3,2} \right) * 20 + \left(\frac{DA}{6} \right) * 15), \quad (9)$$

где DP — защитный профиль игрока.

В защитном профиле основную роль играют подборы. Этот показатель встречается чаще остальных защитных действий и хорошо показывает, насколько игрок участвует в борьбе за владение после броска соперника. Перехваты и блок-шоты имеют меньшую частоту, но высокую значимость: перехват сразу срывает атаку, а блок-шот мешает сопернику завершить бросок. Показатель защитной активности используется как дополнительная сводная характеристика, которая объединяет несколько оборонительных действий в одну оценку.

Общий ML -профиль игрока рассчитывается как взвешенная сумма атакующего и защитного профилей(10):

$$MP = OP * 0,58 + DP * 0,42, \quad (10)$$

где MP — общий профиль игрока.

Атакующий профиль получает больший вес, поскольку в используемых данных лучше раскрыты именно действия в нападении: очки, передачи, точность бросков и соотношение передач к потерям. Эти показатели напрямую влияют на итоговую оценку игрока. Защитный профиль при этом не исключается из расчета, так как вклад баскетболиста определяется не только атакой. Подборы, перехваты, блок-шоты и общая защитная активность также показывают, насколько игрок полезен команде без мяча.

Формирование признаков выполняется в функции `feature_row_from_stats`. Она получает статистику игрока, данные о бросках, позицию и историю выступлений, после чего формирует единый словарь признаков для ML-модуля.

```
def feature_row_from_stats(
    stats: Any,
    shots: Iterable[Any],
    position: str,
    history: Iterable[Any] | None = None,
) -> dict[str, Any]:
    summary = shot_summary(shots)
    averages = league_averages_for_season(getattr(stats,
"season", None))
    row = {
        "points": float(getattr(stats, "points", 0.0) or
0.0),
        "assists": float(getattr(stats, "assists", 0.0) or
0.0),
        "rebounds": float(getattr(stats, "rebounds", 0.0) or
0.0),
        "steals": float(getattr(stats, "steals", 0.0) or
0.0),
        "blocks": float(getattr(stats, "blocks", 0.0) or
0.0),
        "turnovers": float(getattr(stats, "turnovers", 0.0)
or 0.0),
```

```

        "fg_pct": float(getattr(stats, "fg_pct", 0.0) or
0.0),
        "three_pct": float(getattr(stats, "three_pct", 0.0)
or 0.0),
        "ft_pct": float(getattr(stats, "ft_pct", 0.0) or
0.0),
        "shot_distribution": summary["three_rate"],
        "made_shots": summary["made_shots"],
        "missed_shots": summary["missed_shots"],
        "position": str(position or "SF").upper(),
    }
    return complete_derived_features(row)

```

После формирования базовых признаков рассчитываются дополнительные производные показатели. Они позволяют модели учитывать не только исходную статистику, но и более содержательные характеристики игрока: отношение передач к потерям, бросковую точность, защитную активность, атакующую нагрузку и приближенную оценку эффективности.

Для предварительной обработки данных используется `ColumnTransformer`. Числовые признаки масштабируются с помощью `StandardScaler`, а категориальный признак позиции игрока кодируется с помощью `OneHotEncoder`. Это необходимо для того, чтобы модель могла корректно работать как с числовыми показателями, так и с позиционными особенностями игрока.

```

numeric_features = [column for column in FEATURE_COLUMNS if
column != "position"]
preprocessor = ColumnTransformer(
    transformers=[
        ("num", StandardScaler(), numeric_features),
        ("cat", OneHotEncoder(handle_unknown="ignore",
sparse_output=False), ["position"]),
    ]
)

```

Модель машинного обучения реализована как конвейер Pipeline, объединяющий предварительную обработку данных и классификатор. В качестве классификатора используется MLPClassifier с несколькими скрытыми слоями. Такой подход позволяет хранить все этапы обработки и прогнозирования в одной модели.

```

classifier = MLPClassifier(
    hidden_layer_sizes=(128, 64, 32),
    activation="relu",
    alpha=0.0008,
    learning_rate_init=0.0015,
    max_iter=650,
    early_stopping=True,
    validation_fraction=0.14,
    n_iter_no_change=24,
    random_state=42,
)

self.pipeline = Pipeline(
    steps=[
        ("preprocessor", preprocessor),
        ("classifier", classifier),
    ]
)

```

Для определения направлений развития используется многометочная классификация. ML-модель получает вектор признаков X и рассчитывает вероятность необходимости каждой рекомендации:

$$p_{MLj} = P(y_j = 1 | X), \quad (11)$$

где p_{MLj} — вероятность выбора j -го направления развития;

X — вектор признаков игрока;

y_j — бинарная метка рекомендации.

Рекомендация считается выбранной, если вероятность превышает заданный порог:

$y_j = 1$, если $p_{MLj} \geq t_j$,

где t_j — порог выбора рекомендации.

В системе используется базовый порог, позволяющий модели выбирать несколько направлений развития одновременно и не отбрасывать потенциально важные рекомендации при многометочной классификации.

Результатом работы модели являются вероятности по каждому направлению рекомендации. Для этого используется метод `predict_scores`, который возвращает список направлений, их вероятности и признак выбора.

```
def predict_scores(self, row: dict[str, Any], threshold:
float | None = None) -> list[dict[str, Any]]:
    frame = pd.DataFrame([_normalize_row(row)],
columns=FEATURE_COLUMNS)
    probabilities = self._predict_probabilities(frame)
    return [
        {
            "label": label,
            "probability": round(float(probability), 4),
            "selected": bool(float(probability) >=
float(threshold_value)),
        }
        for label, probability, threshold_value
        in zip(RECOMMENDATION_LABELS, probabilities,
self.thresholds, strict=True)
    ]
```

В `RateMyPlayer` рекомендации формируются не только по результатам ML-модели. Система использует гибридную схему: вероятности, рассчитанные моделью, дополняются экспертными правилами по ключевым баскетбольным показателям. Это делает модуль устойчивее. Если модель еще не обучена или данных по игроку недостаточно, система всё равно может выдать полезную рекомендацию на основе понятных статистических условий.

Экспертные правила срабатывают в ситуациях, которые легко интерпретировать без сложной модели. Низкий процент трехочковых бросков

указывает на необходимость работы над дальним броском. Большое количество потерь показывает проблемы с контролем мяча и выбором решения в атаке. Слабый процент штрафных приводит к рекомендации уделить внимание броскам с линии. Если у игрока низкие показатели перехватов и блок-шотов, система выделяет защитную работу как отдельное направление развития.

```
if stats.three_pct < 0.33 and position.upper() != "C":
    labels.append("improve_three_point")
if stats.turnovers > 3:
    labels.extend(["reduce_turnovers",
"improve_decision_making"])
if stats.ft_pct < 0.70:
    labels.append("improve_free_throws")
if stats.steals < 0.8 and stats.blocks < 0.6:
    labels.append("develop_defense")
```

Итоговый список рекомендаций формируется как объединение направлений, выбранных ML-моделью и экспертными правилами. Повторяющиеся рекомендации удаляются, после чего технические метки преобразуются в текстовые рекомендации, понятные пользователю.

```
rule_labels, _rule_texts = self.rule_based.recommend(stats,
shot_list, position)
if self.model and stats is not None:
    model_scores = self.model.predict_scores(row)
    model_labels = [
        str(score["label"])
        for score in model_scores
        if score.get("selected")
    ]
    labels = _deduplicate([*rule_labels, *model_labels])
    recommendations = [LABEL_TEXTS[label] for label in labels if
label in LABEL_TEXTS]
```

На выходе ML-модуль возвращает вероятности по направлениям развития игрока. Эти значения не являются готовым текстом рекомендации:

они передаются дальше в модуль формирования рекомендаций, где объединяются с экспертными правилами и преобразуются в понятные пользователю выводы.

Итоговая задача ML-модуля — подготовить признаки, рассчитать профиль игрока и оценить вероятность каждого направления развития.

4.7 Разработка модуля формирования рекомендаций

Модуль рекомендаций используется на финальном этапе анализа в RateMyPlayer. Он получает уже обработанные данные игрока: сезонную статистику, сведения о бросках, рассчитанные признаки и результаты ML-модуля. На основе этих данных система формирует не просто итоговую оценку, а практические подсказки: над чем игроку стоит работать — дальним броском, контролем мяча, штрафными, защитой, принятием решений или развитием паса.

После получения результатов ML-модуля система переходит к формированию итогового списка рекомендаций. На этом этапе внутренние метки направлений развития сопоставляются с текущими статистическими показателями игрока и преобразуются в понятный для пользователя текст. Модуль не просто выводит результат модели, а проверяет, насколько выбранное направление подтверждается игровым профилем баскетболиста. Благодаря этому рекомендация связывается с конкретной причиной: недостаточной точностью дальних бросков, большим числом потерь, слабой реализацией штрафных или низкой защитной активностью.

Схема формирования рекомендаций представлена на рисунке 19.

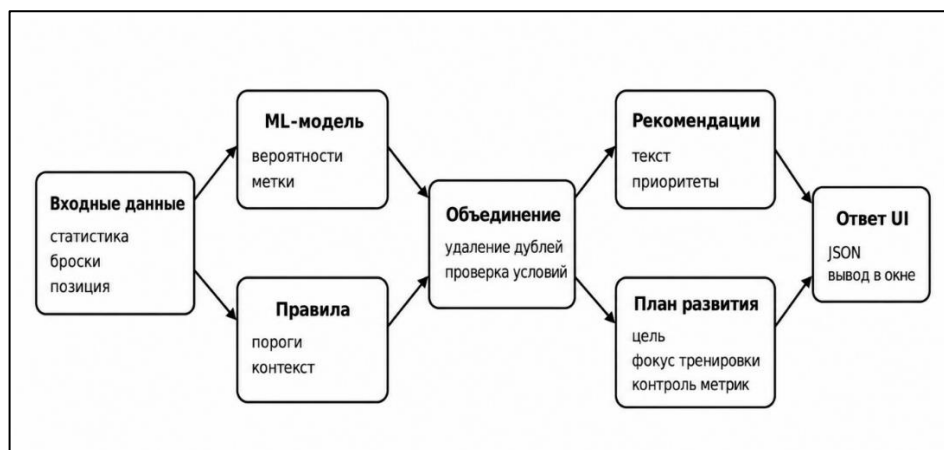


Рисунок 19 – Схема формирования рекомендаций в системе RateMyPlayer

Модуль рекомендаций работает с уже подготовленными результатами анализа и преобразует их в понятные направления развития игрока. На этом этапе внутренние метки рекомендаций сопоставляются с текстовыми пояснениями, повторяющиеся варианты исключаются, а итоговый список приводится к форме, удобной для вывода в интерфейс.

Набор возможных направлений развития задается внутри рекомендательного модуля и используется при интерпретации результатов модели и экспертных правил. Такой подход позволяет не выводить пользователю технические метки, а формировать рекомендации в виде понятных формулировок, связанных с конкретными игровыми показателями баскетболиста.

Для каждого направления в системе задано краткое название и текст рекомендации. Это позволяет преобразовать техническую метку модели в понятное сообщение, которое отображается пользователю в интерфейсе.

```

LABEL_TITLES = {
    "improve_three_point": "Трехочковый бросок",
    "reduce_turnovers": "Контроль мяча",
    "improve_free_throws": "Штрафные броски",
    "attack_paint_more": "Атака краски",
    "develop_defense": "Защитная работа",
    "improve_decision_making": "Принятие решений",

```

```

        "improve_conditioning": "Физическая готовность",
        "develop_playmaking": "Пас и видение площадки",
    }

```

Тексты рекомендаций вынесены отдельно от технических меток модели. За счет этого система не показывает пользователю сухой результат классификации, а сразу формирует понятное пояснение. Например, если у игрока выявлена проблема с дальним броском, рекомендация связывается не просто с меткой `improve_three_point`, а с конкретными действиями: улучшить работу ног, ускорить выпуск мяча и чаще отрабатывать броски из углов.

```

LABEL_TEXTS = {
    "improve_three_point": (
        "Уделить отдельный блок тренировок трехочковому
броску: "
        "работа с ногами, быстрый выпуск и серии из углов."
    ),
    "reduce_turnovers": (
        "Снизить количество потерь: добавить упражнения на
контроль мяча "
        "под давлением и чтение ловушек."
    ),
    "improve_free_throws": (
        "Улучшить штрафные: зафиксировать рутину перед
броском "
        "и вести учет серий после нагрузки."
    ),
}

```

Формирование рекомендаций сначала выполняется по экспертным правилам. За этот этап отвечает класс `RuleBasedRecommender`, который проверяет статистику игрока по заранее заданным условиям. Если у баскетболиста процент трехочковых ниже 0,33 и он не играет на позиции центрового, система отмечает необходимость работы над дальним броском. При количестве потерь больше 3 за игру добавляются рекомендации, связанные с контролем мяча и выбором решений в атаке.

```

if stats.three_pct < 0.33 and position.upper() != "C":
    labels.append("improve_three_point")
if stats.turnovers > 3:
    labels.extend(["reduce_turnovers",
"improve_decision_making"])
if stats.ft_pct < 0.70:
    labels.append("improve_free_throws")
if stats.steals < 0.8 and stats.blocks < 0.6:
    labels.append("develop_defense")

```

На втором этапе подключается ML-модель. Если обученный файл модели найден и совместим с текущим набором признаков, система рассчитывает вероятность для каждого направления развития. Рекомендация попадает в итоговый список только в том случае, если ее вероятность выше заданного порога. После этого выполняется дополнительная проверка по текущей статистике игрока: например, рекомендация по трехочковому броску не должна появляться, если процент дальних бросков находится на хорошем уровне.

```

if self.model and stats is not None:
    model_scores = self.model.predict_scores(row)
    model_labels = [
        str(score["label"])
        for score in model_scores
        if score.get("selected")
        and
_label_is_contextually_supported(str(score.get("label")), row)
    ]

```

Контекстная проверка отсекает рекомендации, которые не подтверждаются статистикой игрока. Например, совет по работе над трехочковым броском остается в итоговом списке только тогда, когда процент дальних попаданий ниже заданного порога или заметно хуже среднего уровня по лиге. Такой фильтр снижает риск случайных рекомендаций и делает результат более связанным с реальными показателями баскетболиста.

```

def _label_is_contextually_supported(label: str, row:
dict[str, Any]) -> bool:
    if label == "improve_three_point":
        return (
            float(row.get("three_pct") or 0.0) < 0.35
            or float(row.get("three_pct_vs_league") or 0.0)
< -0.06
        )
    if label == "reduce_turnovers":
        return (
            float(row.get("turnovers") or 0.0) >= 3.0
            or float(row.get("ast_tov_ratio") or 0.0) < 2.0
        )
    return True

```

После работы модели и правил система собирает найденные направления в единый список. Если одна и та же рекомендация появилась из двух источников, она оставляется только один раз. Затем внутренние метки преобразуются в понятный текст для пользователя. Такой порядок позволяет совместить результаты обученной модели с логикой предметной области и избежать дублирования рекомендаций.

```

labels = _deduplicate([*rule_labels, *model_labels])
recommendations = [
    LABEL_TEXTS[label]
    for label in labels
    if label in LABEL_TEXTS
]

```

Дополнительно для каждой рекомендации формируется оценка важности. Она зависит от вероятности, рассчитанной ML-моделью, и силы срабатывания экспертного правила. Если рекомендация выбрана и моделью, и правилом, она получает источник hybrid. Если она выбрана только правилами, источник обозначается как rules, а если только моделью — как ml.

```

if not is_selected:
    source = "not_selected"

```

```

elif is_rule and model_probability > 0:
    source = "hybrid"
elif is_rule:
    source = "rules"
else:
    source = "ml" if self.model else "not_selected"

```

Кроме текстовых рекомендаций, система формирует план развития игрока. Для каждого выбранного направления указываются приоритет, цель, тренировочный фокус и контрольные метрики. Например, для рекомендации по снижению потерь целью является уменьшение количества потерь до уровня ниже 2,5 за игру, а контрольными метриками являются turnovers, AST/TOV и динамика потерь.

```

"reduce_turnovers": {
    "target": "снизить потери до уровня ниже 2.5 за игру",
    "training_focus": (
        "ball pressure drills, чтение ловушек, "
        "первый пас после пик-н-ролла"
    ),
    "control_metrics": ["turnovers", "AST/TOV",
"turnovers_trend"],
}

```

Итоговая работа модуля выполняется в классе HybridRecommender. Он получает данные игрока, вызывает модуль правил, при наличии модели получает ML-оценки, объединяет результаты, формирует профиль игрока, список рекомендаций и план развития. Результат возвращается серверной части, а затем передается клиентскому интерфейсу в формате JSON.

```

return RecommendationResponse(
    player_id=player.id,
    player_name=player.full_name,
    source="hybrid_ml_rules" if self.model else "rule_based",
    labels=labels,
    recommendations=recommendations,
    scores=scores,

```

```
model_info=model_info,  
ml_profile=ml_profile,  
development_plan=development_plan,  
model_artifacts=model_artifacts,  
) .model_dump()
```

Модуль формирования рекомендаций связывает расчетные показатели, ML-оценки и экспертные правила в один итоговый вывод. Пользователь получает не только общий рейтинг, но и список конкретных направлений развития с пояснением причин выбора.

4.8 Выводы по разделу

В четвертом разделе описана реализация RateMyPlayer: клиент-серверная структура, интерфейс, база данных, обработка статистики, ML-модуль и модуль рекомендаций.

Серверная часть на FastAPI работает с базой SQLite через SQLAlchemy, получает данные и передает результаты клиенту. Интерфейс на PySide6 позволяет пользователю работать с карточками игроков, рейтингом, картой бросков и рекомендациями.

Основной результат раздела — описание того, как отдельные компоненты системы связаны в единое приложение: данные сохраняются, обрабатываются, используются в расчетах и передаются в ML-модуль для формирования рекомендаций.

5 Результаты исследования и тестирование системы

5.1 Методика тестирования разработанной системы

После завершения разработки основных компонентов системы RateMyPlayer было выполнено тестирование, направленное на проверку корректности работы пользовательского интерфейса, серверной части, базы данных, модулей обработки спортивных данных, ML-модуля и модуля формирования рекомендаций.

В рамках работы применялось функциональное тестирование. Оно было направлено на проверку основных действий пользователя: запуск приложения, добавление и редактирование игроков, загрузку данных, получение информации из внешнего источника ESPN, просмотр статистики, расчет эффективности, отображение карты бросков и получение рекомендаций.

Тестирование выполнялось по основным частям системы. Сначала проверялся клиентский интерфейс: запуск приложения, работа кнопок, переключение вкладок, открытие диалоговых окон и ввод данных через формы. После этого тестировалась серверная часть: доступность FastAPI-сервера, обработка HTTP-запросов и корректность JSON-ответов, которые получает клиентское приложение.

Отдельное внимание было уделено базе данных. Проверялись операции добавления, изменения, удаления и повторного получения сведений об игроках, сезонной статистике и бросках. Также контролировалось, сохраняется ли связь между карточкой игрока и относящимися к нему статистическими записями.

Далее проверялся полный путь данных внутри приложения: загрузка из выбранного источника, сохранение в базе, повторное открытие карточки игрока и вывод результатов в интерфейсе. Для импортируемых данных оценивалось, корректно ли система считывает файл или ответ внешнего источника и связывает полученные сведения с нужным игроком.

Отдельно проверялась аналитическая часть: построение статистики, рейтинга, карты бросков и рекомендаций для выбранного спортсмена.

Результат считался корректным, если данные отображались в соответствующих вкладках, карта строилась по координатам бросков, а рекомендации были связаны с показателями игрового профиля.

Основными критериями проверки стали: запуск клиента и сервера без ошибок, успешная передача данных между ними, сохранение сведений в базе, корректный импорт файлов, построение визуализаций и получение результата от ML-модуля. Такая методика позволила проверить не отдельную функцию, а полный сценарий работы RateMyPlayer от загрузки данных до вывода аналитического результата пользователю.

5.2 Проверка работы пользовательского интерфейса и процессов получения, хранения и обработки данных

Проверка пользовательского интерфейса выполнялась с целью убедиться, что клиентская часть системы RateMyPlayer корректно запускается, отображает основные элементы управления и обеспечивает доступ к функциям программы. Через интерфейс пользователь работает с данными игроков, запускает анализ и просматривает результаты работы интеллектуального модуля.

Проверка интерфейса началась с запуска приложения и просмотра главного окна. После открытия системы пользователь видит список игроков, панель управления, область с подробной информацией о выбранном спортсмене и вкладки с результатами анализа.

Отдельно проверялась боковая панель (рисунки 20). Через нее пользователь выбирает игрока, обновляет список, добавляет новую карточку, загружает данные из файлов и запускает получение информации из ESPN. Все основные действия должны быть доступны без перехода к программному коду.

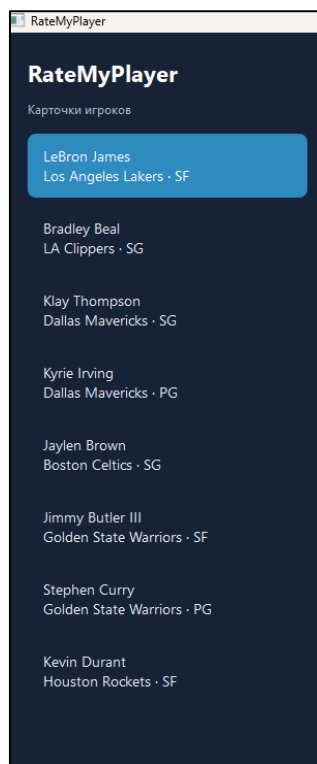


Рисунок 20 – Боковая панель

Затем была проверена центральная область окна(рисунок 21). При выборе игрока в ней отображаются имя, команда, позиция, источник данных и доступные действия. Также проверялись операции редактирования и удаления карточки. После изменения данных интерфейс обновлял информацию без повторного запуска приложения.

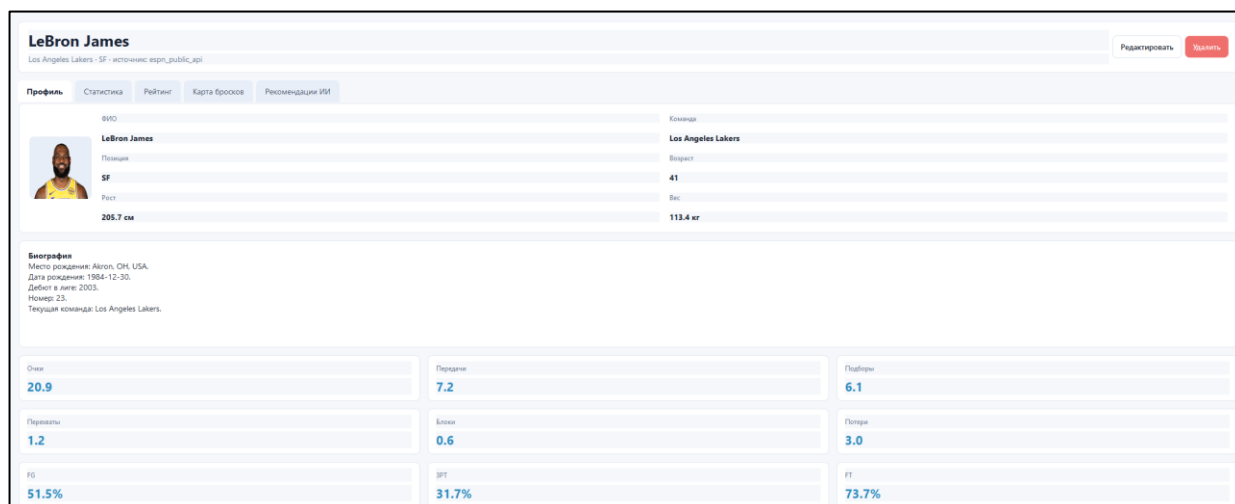


Рисунок 21 – Центральная область окна

Вкладки «Профиль»(рисунок 21), «Статистика»(рисунок 22), «Рейтинг»(рисунок 23), «Карта бросков»(рисунок 24) и «Рекомендации ИИ»

проверялись отдельно. При переключении между ними отображались соответствующие данные: общие сведения об игроке, сезонные показатели, рассчитанный рейтинг, визуализация бросков и рекомендации ML-модуля.

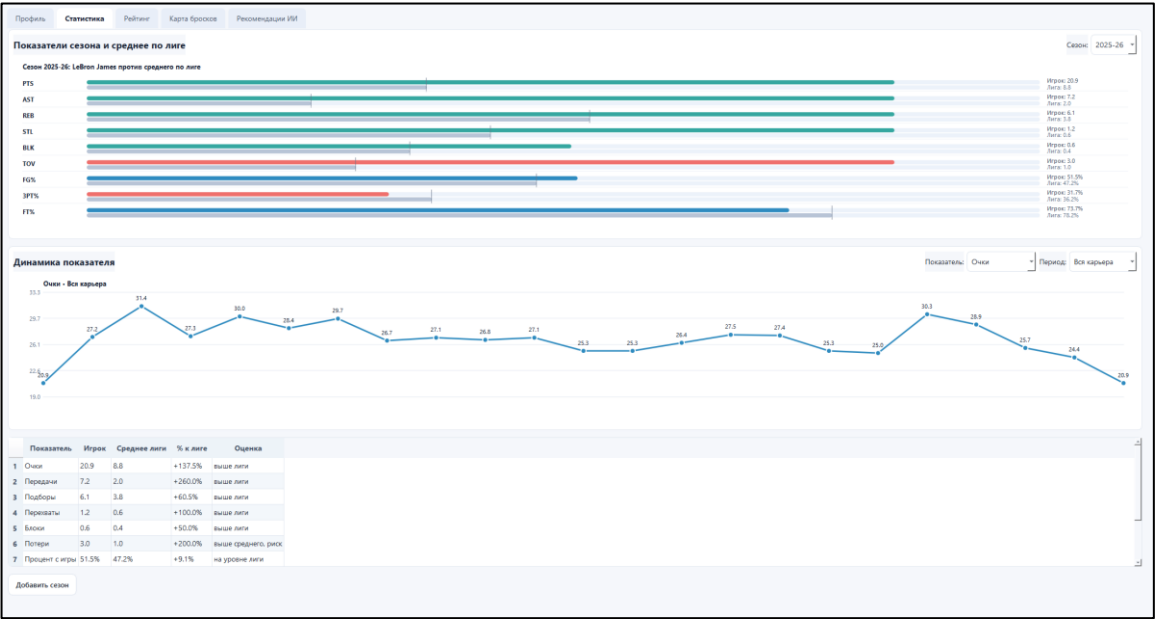


Рисунок 22 – Вкладка «Статистика»

Профиль

Статистика

Рейтинг

Карта бросков

Рекомендации ИИ

Рейтинг игрока в базе RateMyPlayer

Итоговое место: #5 из 7 Игроков в расчете: 7

	Показатель	Игрок	Место	Лидер	Значение лидера	Оценка
1	Очки	20.9	#4 из 7	Jaylen Brown	28.7	середина
2	Передачи	7.2	#1 из 7	LeBron James	7.2	топ базы
3	Подборы	6.1	#2 из 7	Jaylen Brown	6.9	выше среднего
4	Перехваты	1.2	#2 из 7	Jimmy Butler III	1.4	выше среднего
5	Блоки	0.6	#2 из 7	Kevin Durant	0.9	выше среднего
6	Потери	3.0	#5 из 7	Klay Thompson	0.9	середина
7	Процент с игры	51.5%	#3 из 7	Kevin Durant	52.0%	середина
8	Трехочковые	31.7%	#7 из 7	Kevin Durant	41.3%	ниже среднего
9	Штрафные	73.7%	#7 из 7	Stephen Curry	92.3%	ниже среднего

Пороги оценки: топ базы - место в верхних 10% рейтинга; выше среднего - верхние 35%; середина - до 70%; ниже среднего - ниже 70% по выбранному показателю.

Рисунок 23 – Вкладка «Рейтинг»

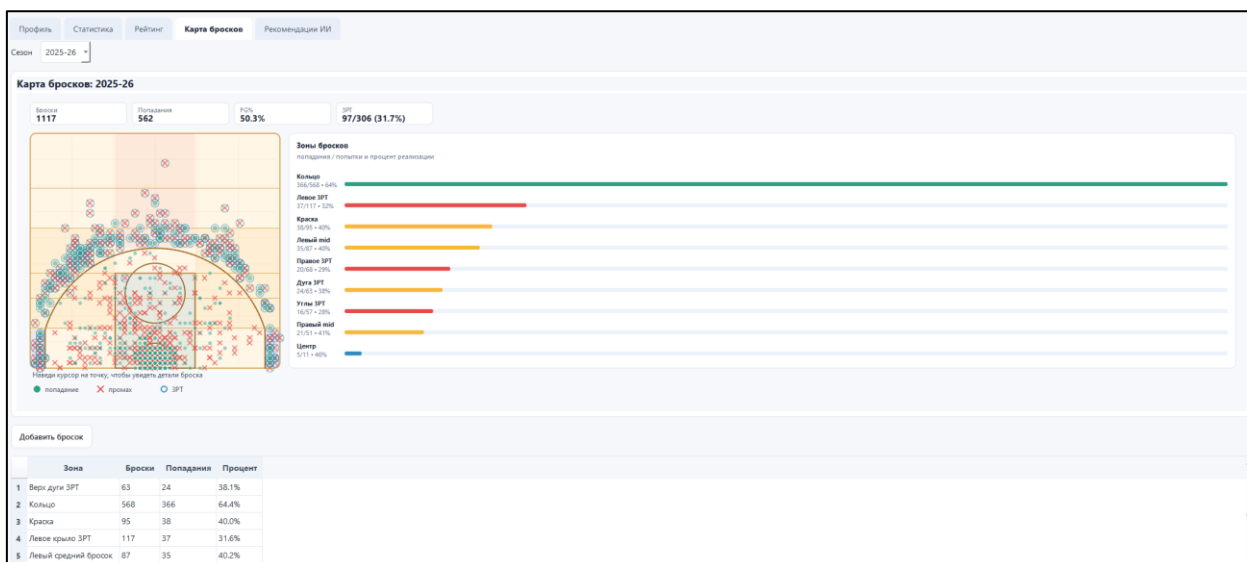


Рисунок 24 – Вкладка «Карта бросков»

Также тестировались диалоговые окна для добавления и редактирования игрока, загрузки данных, внесения сезонной статистики и добавления бросков. Введенная информация передавалась на сервер, сохранялась в базе данных и затем отображалась в интерфейсе.

При проверке ручного добавления данных создавалась новая карточка игрока. В форме указывались основные сведения о спортсмене: имя, команда, позиция, возраст, рост и вес. После сохранения запись передавалась на серверную часть и добавлялась в базу данных. Результат проверки представлен на рисунке 25.

Добавить игрока

ФИО

Иванов Иван Иванович

Команда

"Сурские Крылья"

Позиция

PG

Возраст

20

Рост, см

190

Вес, кг

85

Фото

Выбрать

Биография

1. Родился 18.10.2000
2. Город - Пенза

Сезонная статистика

Добавить сезон

Удалить сезон

	Сезон	Очки	Передачи	Подборы	Перехваты	Блоки	Потери	FG	ЗРТ	FT
1	2025/26	10	4	3	2	1	3	5	1,4	3

Можно добавить любое количество сезонов. Проценты вводятся в формате 0.472 или 47.2.

Сохранить

Отмена

Рисунок 25 – Ручное добавление карточки игрока

После добавления карточки проверялось, что новый игрок отображается в общем списке и может быть выбран для дальнейшей работы. При выборе записи в центральной области интерфейса выводились сведения о спортсмене и становились доступны действия для редактирования, удаления, просмотра статистики и анализа. Результат отображения добавленного игрока представлен на рисунке 26.

Иванов Иван Иванович

Редактировать

Удалить

"Сурские Крылья" - PG - источник: талисман

Профиль

Статистика

Рейтинг

Карта бросков

Рекомендации ИИ

ФИО

Иванов Иван Иванович

Команда

"Сурские Крылья"

Позиция

PG

Возраст

20

Рост

190.0 см

Вес

85.0 кг

Биография

1. Родился 18.10.2000
2. Город - Пенза

Очки

10.0

Передачи

4.0

Подборы

3.0

Перехваты

2.0

Блоки

1.0

Потери

3.0

FG

5.0%

ЗРТ

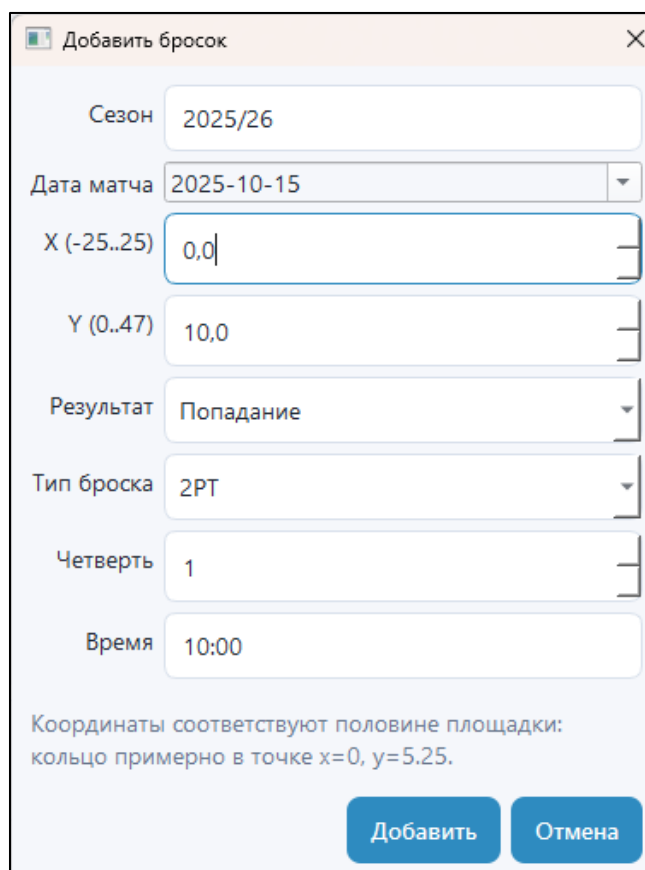
1.4%

FT

3.0%

Рисунок 26 – Профиль созданного игрока

Отдельно проверялось добавление данных о бросках игрока. Для этого использовалось диалоговое окно, в котором указываются сезон, дата матча, координаты броска на площадке, результат попытки, тип броска, четверть и время игрового эпизода(рисунок 27).



Добавить бросок

Сезон 2025/26

Дата матча 2025-10-15

X (-25..25) 0,0

Y (0..47) 10,0

Результат Попадание

Тип броска 2PT

Четверть 1

Время 10:00

Координаты соответствуют половине площадки:
кольцо примерно в точке $x=0$, $y=5.25$.

Добавить Отмена

Рисунок 27 – Окно «Добавление броска»

На вкладке «Карта бросков» отображается схема баскетбольной площадки с нанесенными бросковыми попытками выбранного игрока. На рисунке 28 видно, что попадания и промахи отмечены разными маркерами, а ниже представлена сводная информация по броскам. Такой вид позволяет визуально оценить расположение попыток на площадке и увидеть, в каких зонах у игрока накоплено больше бросков.

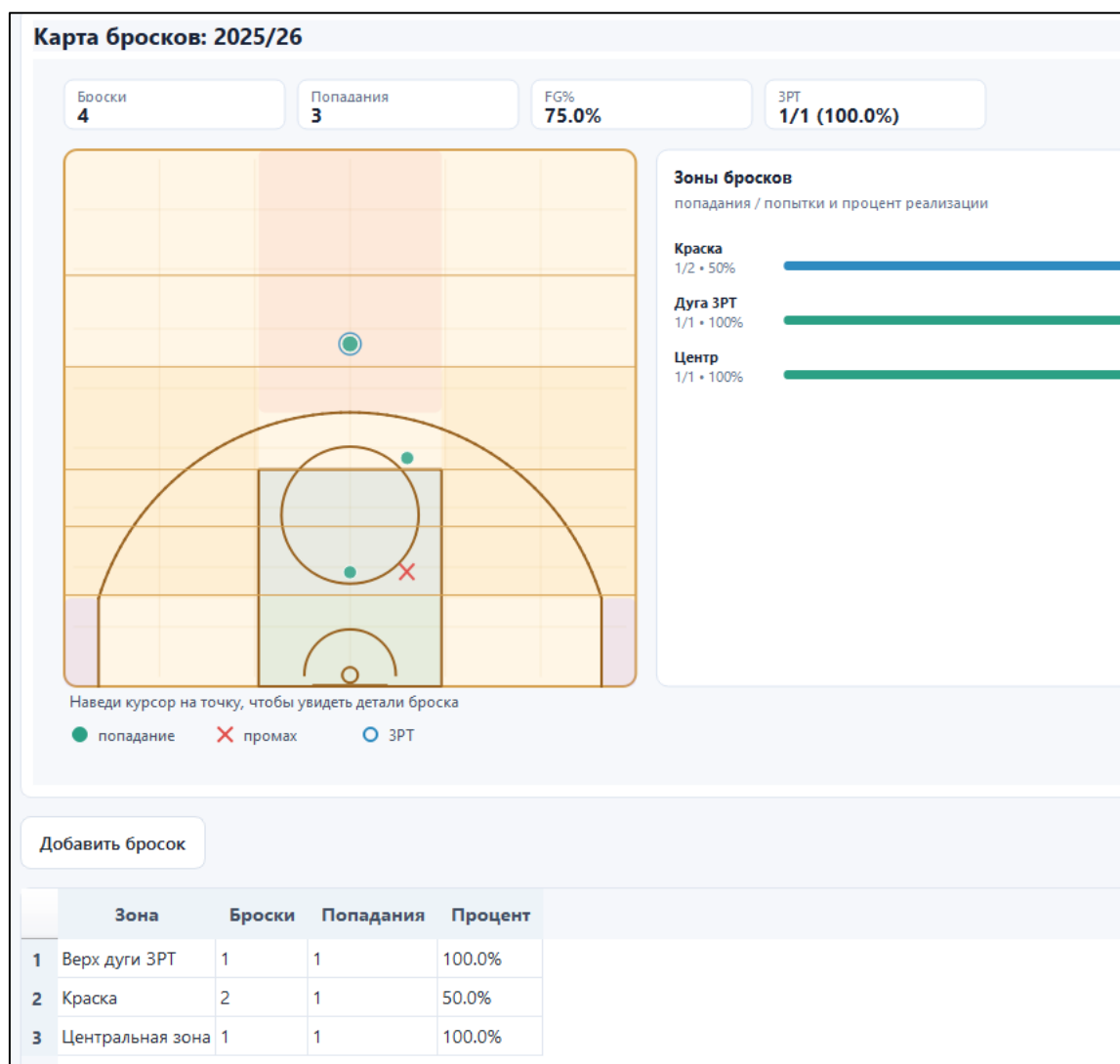


Рисунок 28 – Окно «Карта бросков», после ручного добавления

Дополнительно проверялась работа интерфейса при обращении к серверу. При расчете аналитики, получении рекомендаций и загрузке данных клиентская часть отправляла запросы к FastAPI-серверу и после ответа обновляла нужные элементы окна. При ошибке соединения или некорректном ответе пользователю выводилось сообщение.

Также проверялась загрузка подготовленного CSV-файла со статистическими данными. На рисунке 29 представлен фрагмент файла, содержащий сведения об игроках, командах, позициях, сезонах и основных статистических показателях. Наличие нескольких строк по одному игроку позволяет проверить не только импорт карточки спортсмена, но и сохранение сезонной статистики.

player_name	team	position	age	height	weight	season	points	assists	rebounds	steals	blocks	turnovers	fg_pct	three_pct	ft_pct	biography
Иван Петров	Московские Соколы	PG	24	188	84	2020-21	12.4	6.8	3.1	1.2	0.2	2.6	0.438	0.341	0.781	"Тестовый разыгрывающий для проверки CSV-импорта: сильный пас и развивающийся трехочковый бросок."
Иван Петров	Московские Соколы	PG	24	188	84	2021-22	14.1	7.3	3.4	1.3	0.2	2.9	0.446	0.352	0.795	"Тестовый разыгрывающий для проверки CSV-импорта: сильный пас и развивающийся трехочковый бросок."
Иван Петров	Московские Соколы	PG	24	188	84	2022-23	16.3	7.8	3.7	1.4	0.3	3.1	0.452	0.365	0.812	"Тестовый разыгрывающий для проверки CSV-импорта: сильный пас и развивающийся трехочковый бросок."
Иван Петров	Московские Соколы	PG	24	188	84	2023-24	18.0	8.1	4.0	1.5	0.3	3.4	0.459	0.371	0.826	"Тестовый разыгрывающий для проверки CSV-импорта: сильный пас и развивающийся трехочковый бросок."
Иван Петров	Московские Соколы	PG	24	188	84	2024-25	19.2	8.6	4.2	1.6	0.3	3.6	0.464	0.379	0.838	"Тестовый разыгрывающий для проверки CSV-импорта: сильный пас и развивающийся трехочковый бросок."
Алексей Морозов	Казанские Волки	SF	27	203	99	2020-21	20.5	3.1	6.4	0.9	0.6	2.1	0.482	0.389	0.742	"Тестовый легкий форвард для проверки CSV-импорта: результативный скорер с хорошим подбором."
Алексей Морозов	Казанские Волки	SF	27	203	99	2021-22	21.7	3.4	6.8	1.0	0.7	2.3	0.491	0.401	0.758	"Тестовый легкий форвард для проверки CSV-импорта: результативный скорер с хорошим подбором."
Алексей Морозов	Казанские Волки	SF	27	203	99	2022-23	23.4	3.7	7.1	1.1	0.8	2.5	0.503	0.412	0.771	"Тестовый легкий форвард для проверки CSV-импорта: результативный скорер с хорошим подбором."
Алексей Морозов	Казанские Волки	SF	27	203	99	2023-24	24.8	4.0	7.5	1.2	0.8	2.7	0.511	0.407	0.786	"Тестовый легкий форвард для проверки CSV-импорта: результативный скорер с хорошим подбором."
Алексей Морозов	Казанские Волки	SF	27	203	99	2024-25	26.1	4.2	7.9	1.3	0.9	2.8	0.518	0.398	0.801	"Тестовый легкий форвард для проверки CSV-импорта: результативный скорер с хорошим подбором."
Никита Соколов	Уральские Медведи	C	22	213	112	2020-21	8.2	1.1	7.4	0.4	1.1	1.8	0.548	0.182	0.612	"Тестовый центровой для проверки CSV-импорта: защита кольца, подбор и нестабильные штрафные."
Никита Соколов	Уральские Медведи	C	22	213	112	2021-22	9.6	1.3	8.1	0.5	1.4	1.9	0.563	0.201	0.633	"Тестовый центровой для проверки CSV-импорта: защита кольца, подбор и нестабильные штрафные."
Никита Соколов	Уральские Медведи	C	22	213	112	2022-23	11.1	1.5	8.9	0.6	1.7	2.0	0.574	0.214	0.651	"Тестовый центровой для проверки CSV-импорта: защита кольца, подбор и нестабильные штрафные."
Никита Соколов	Уральские Медведи	C	22	213	112	2023-24	12.4	1.7	9.6	0.6	1.9	2.2	0.589	0.228	0.668	"Тестовый центровой для проверки CSV-импорта: защита кольца, подбор и нестабильные штрафные."
Никита Соколов	Уральские Медведи	C	22	213	112	2024-25	13.6	1.9	10.3	0.7	2.1	2.4	0.601	0.241	0.684	"Тестовый центровой для проверки CSV-импорта: защита кольца, подбор и нестабильные штрафные."

Рисунок 29 – CSV-файл с данными игроков

После выполнения импорта проверялось, что загруженные игроки появились в боковой панели приложения. На рисунке 30 видно, что в списке отображаются карточки игроков, полученные из файла, а выбранная запись подсвечивается в интерфейсе. Это подтверждает, что данные были считаны из файла, переданы на сервер и сохранены в базе данных.

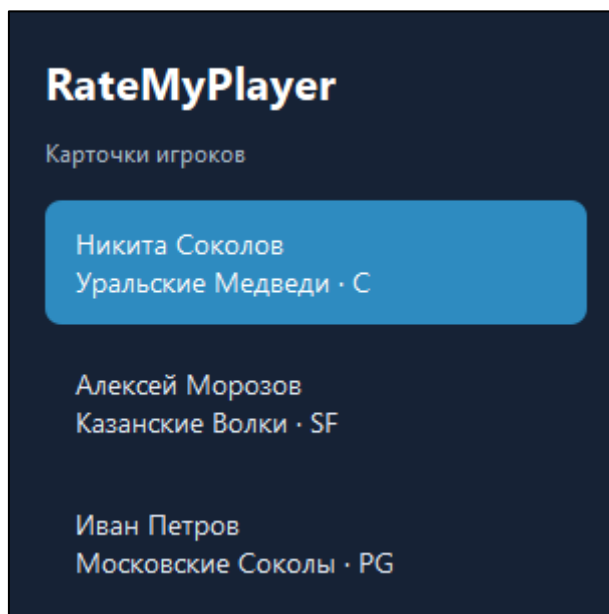


Рисунок 30 – Отображение игроков после импорта CSV-файла

Затем был выбран один из импортированных игроков для проверки корректности отображения сохраненных сведений. На рисунке 31 показан профиль игрока, где отображаются его ФИО, команда, позиция, рост, вес, биография и основные статистические показатели. Это подтверждает, что после импорта данные доступны не только в списке игроков, но и в подробной карточке спортсмена.

ИИО		Команда	
Никита Сиволов		Уральские Медведи	
НС	Позиция	Возраст	
	С	22	
	Рост	Вес	
	213.0 см	112.0 кг	
Биография			
Тестовый центр для проверки CSV-импорта: защита кода, подбор и нестабильные графики.			
Очки		Порядок	
13.6		1.9	
Потери		Блоки	
0.7		2.1	
FG		FT	
60.1%		24.1%	
		FT	
		68.4%	

Рисунок 31 – Профиль игрока, загруженного из CSV-файла

Проверка показала, что интерфейс RateMyPlayer корректно обеспечивает доступ к основным функциям системы: работе с карточками игроков, загрузке данных, просмотру статистики, рейтинга, карты бросков и рекомендаций.

5.3 Проверка работы ML-модуля и формирования рекомендаций

Проверка ML-модуля была направлена на то, чтобы убедиться, что система правильно готовит признаки, передает их в модель и получает рекомендации, связанные со статистикой игрока. Этот этап важен, потому что именно он отличает RateMyPlayer от обычного приложения для хранения спортивных данных.

На вход ML-модуля подаются сезонная статистика, позиция игрока и данные о бросках. Дополнительно рассчитываются производные признаки: отношение передач к потерям, бросковая точность, защитная активность, атакующая нагрузка, динамика показателей и отклонение от средних значений по лиге.

Во время тестирования проверялось, что данные корректно извлекаются из базы, приводятся к нужному формату, масштабируются и передаются в модель. Отдельно контролировалась обработка позиции игрока как категориального признака.

Модель решает задачу многометочной классификации: одному баскетболисту может соответствовать сразу несколько направлений развития. В системе проверялись рекомендации по дальнему броску, потерям,

штрафным, атакам в краску, защите, принятию решений, физической готовности и развитию паса.

Итоговый результат формируется по гибридной схеме. Модель рассчитывает вероятности по направлениям, а экспертные правила проверяют очевидные статистические ситуации: низкий процент трехочковых, большое число потерь, слабую реализацию штрафных или недостаточную защитную активность.

Результаты выводятся во вкладке «Рекомендации ИИ». Пользователь видит ML-профиль, график вероятностей, таблицу выбранных направлений, план развития и кнопки управления моделью.

Результат работы ML-модуля и формирования рекомендаций представлен на рисунке 32.

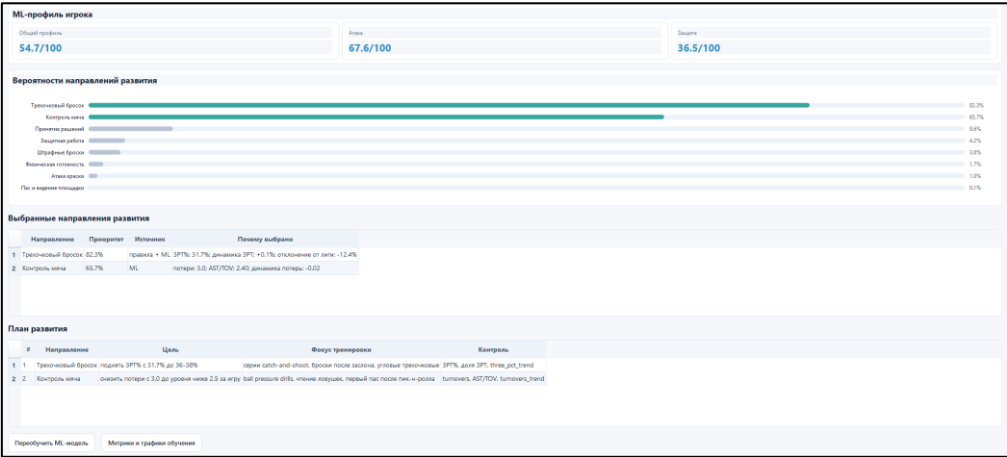


Рисунок 32 — Результат работы ML-модуля и формирования рекомендаций

Вкладка начинается с ML-профиля игрока. В нем выводятся общий, атакующий и защитный профили по шкале от 0 до 100. Эти значения дают быстрый ориентир: за счет каких сторон игры баскетболист выглядит сильнее.

Ниже расположен график вероятностей направлений развития. Каждая строка соответствует одной рекомендации, а значение показывает степень уверенности модели. Например, низкий процент дальних бросков повышает вероятность направления «Трехочковый бросок», а большое количество потерь — направлений «Контроль мяча» и «Принятие решений».

После графика отображается таблица выбранных направлений. В ней указаны приоритет, источник рекомендации и причина выбора. Это помогает понять, пришел ли результат от ML-модели, экспертных правил или их объединения.

Таблица «План развития» связывает рекомендацию с практическими действиями. Для каждого направления задаются цель, тренировочный фокус и контрольные метрики. Например, для контроля мяча отслеживаются `turnovers`, `AST/TOV` и динамика потерь, а для трехочкового броска — процент попаданий и доля дальних попыток.

Кнопка «Переобучить ML-модель» запускает повторное обучение на подготовленном наборе данных. После этого параметры модели обновляются, а вероятности по направлениям развития могут измениться. Такой механизм полезен при добавлении новых игроков, сезонов и бросков.

Кнопка «Метрики и графики обучения» открывает отдельное окно с отчетом по модели. В нем собраны графики обучения, качества классификации, распределения меток и значимости признаков. Это окно используется для проверки ML-модуля и анализа корректности рекомендаций.

Окно с метриками и графиками обучения ML-модели представлено на рисунке 33.

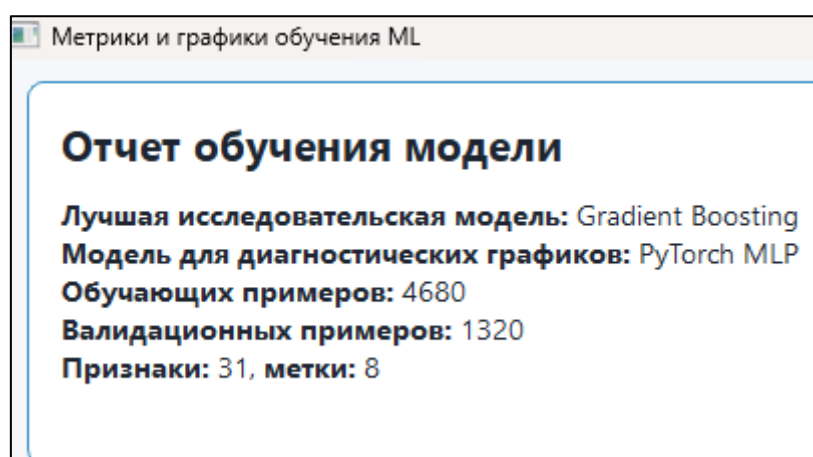


Рисунок 33 — Окно метрик и графиков обучения ML-модели

В верхней части окна отображается краткая информация об отчете обучения. В ней указывается лучшая исследовательская модель, модель,

использованная для диагностических графиков, количество обучающих и валидационных примеров, количество признаков и количество целевых меток. Эти сведения позволяют оценить, на каком объеме данных обучалась модель и сколько направлений рекомендаций она должна определять.

Одним из основных графиков является «Кривые обучения PyTorch MLP» (рисунок 34). На нем показано, как менялись показатели модели по эпохам обучения. В левой части графика отображаются две кривые функции потерь: для обучающей и валидационной выборок. В начале обучения обе кривые быстро снижаются, что показывает активное уменьшение ошибки модели на первых эпохах. Затем снижение становится более плавным, а значения функции потерь постепенно стабилизируются. При этом валидационная кривая не начинает резко расти, поэтому по графику не наблюдается явного переобучения.

В центральной части рисунка показаны значения Micro-F1 и Macro-F1. Micro-F1 растет быстрее и достигает более высокого уровня, так как эта метрика учитывает общий результат по всем меткам. Macro-F1 увеличивается медленнее, поскольку сильнее отражает качество по отдельным направлениям рекомендаций, в том числе по менее частым меткам. Постепенный рост обеих метрик показывает, что модель со временем лучше распознает направления развития игрока.

В целом график показывает нормальный ход обучения PyTorch MLP: ошибка на обучающей и валидационной выборках снижается, а F1-метрики растут и выходят на устойчивый уровень. Это означает, что модель не просто запоминает обучающие данные, а сохраняет качество на валидационной выборке и может использоваться для дальнейшего формирования рекомендаций.

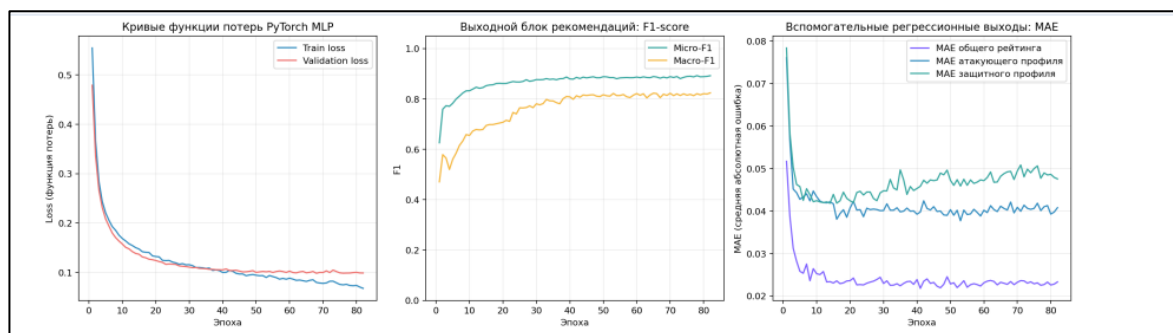


Рисунок 34 — Кривые обучения Pytorch MLP

Отдельно на рисунке 35 представлен график «MAE вспомогательных регрессионных выходов». Он показывает изменение средней абсолютной ошибки для трех числовых выходов модели: общего рейтинга, атакующего профиля и защитного профиля. По оси X отложены эпохи обучения, по оси Y — значение MAE. Чем ниже линия на графике, тем меньше среднее отклонение предсказанного значения от целевого.

На графике видно, что ошибка по общему рейтингу быстро снижается в первые эпохи и затем остается на самом низком уровне среди трех выходов. Это означает, что модель лучше всего приближает именно общий числовой профиль игрока. Ошибка атакующего профиля после начального снижения стабилизируется примерно на одном уровне, без резких скачков. Защитный профиль показывает более заметные колебания и остается выше остальных кривых, что говорит о более сложном приближении защитных характеристик по имеющемуся набору признаков.

По данному графику можно сделать вывод, что вспомогательные регрессионные выходы обучаются с разной точностью. Общий рейтинг предсказывается наиболее устойчиво, атакующий профиль сохраняет приемлемый уровень ошибки, а защитный профиль остается самым сложным для модели. Это связано с тем, что защитные действия в статистике представлены меньшим числом показателей и могут требовать более подробных данных для повышения точности.

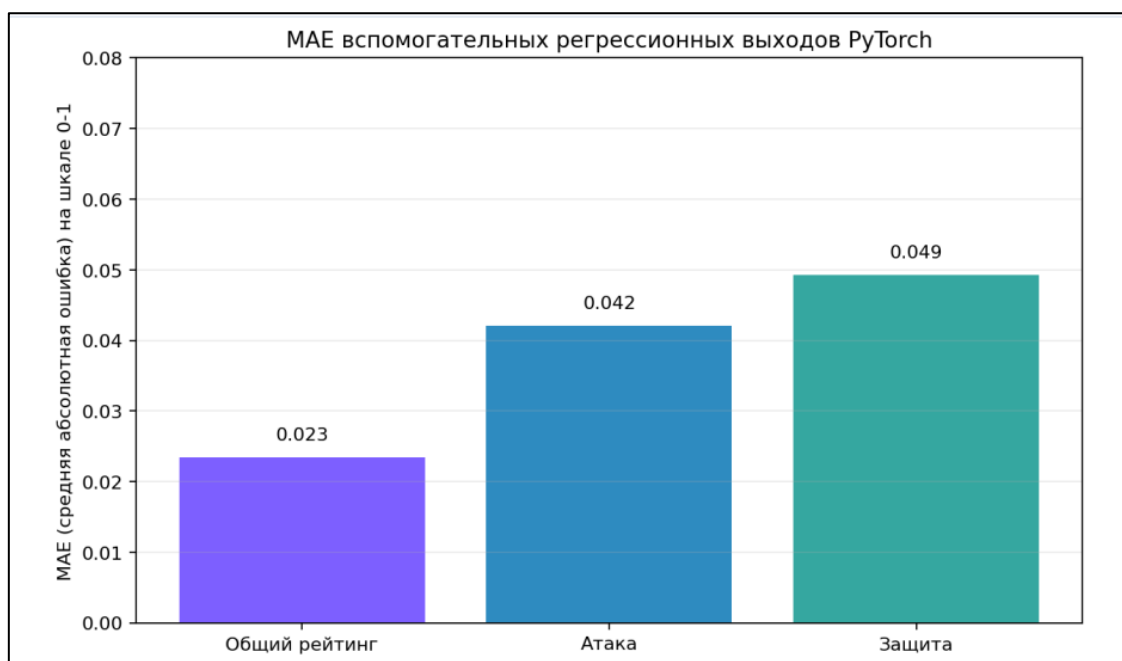


Рисунок 35 — MAE вспомогательных регрессионных выходов обучения

График «Сравнение моделей multi-label классификации» (рисунок 36) показывает сопоставление нескольких алгоритмов по качеству определения направлений рекомендаций. По оси X расположены сравниваемые модели: градиентный бустинг, случайный лес, MLP PyTorch, MLP Scikit-learn и логистическая регрессия. Для каждой модели приведены значения Micro-F1, Macro-F1 и Hamming score.

По графику видно, что наилучшие результаты показывает градиентный бустинг: его значения по всем трем метрикам находятся ближе всего к единице. Случайный лес также демонстрирует высокий уровень качества и незначительно уступает лидирующей модели. MLP PyTorch занимает промежуточное положение: результат ниже, чем у ансамблевых моделей, но выше, чем у MLP Scikit-learn и логистической регрессии. Наименьшие значения наблюдаются у логистической регрессии, что указывает на ее меньшую пригодность для данной задачи.

Сравнение показывает, что ансамблевые модели лучше справляются с тестовой задачей многометочной классификации. При этом в рабочем ML-модуле RateMyPlayer используется MLP PyTorch, поскольку она соответствует

выбранной нейросетевой архитектуре системы и может быть расширена при увеличении объема обучающих данных.

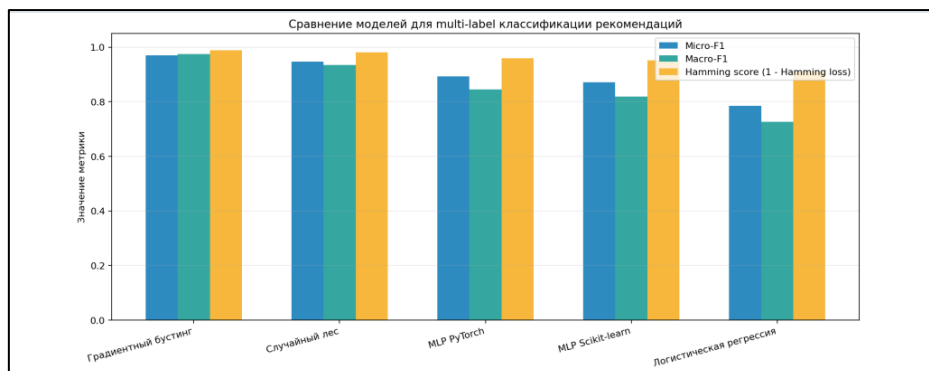


Рисунок 36 — Сравнение моделей multi-label классификации

График «Precision, Recall и F1-score по целевым меткам» (рисунок 37) показывает качество работы MLP-модели PyTorch отдельно по каждому направлению рекомендаций. По оси X расположены целевые метки, а для каждой из них показаны три столбца: Precision, Recall и F1-score. Это позволяет увидеть, какие направления модель определяет уверенно, а где качество ниже.

Наиболее высокие значения наблюдаются для направления «ЗРТ-бросок»: все три столбца находятся выше 0,9, поэтому модель хорошо распознает случаи, связанные с необходимостью работы над дальним броском. Близкие к высоким значения также видны у направлений «Контроль мяча» и «Штрафные»: показатели находятся примерно в диапазоне от 0,86 до 0,94, без резкого провала одной из метрик.

Более слабые результаты заметны у направления «Атака кольца». Здесь Precision ниже, чем Recall, поэтому модель чаще находит такие случаи, но часть выбранных рекомендаций может быть менее точной. Похожая ситуация видна у направления «Принятие решений»: Recall высокий, однако Precision заметно ниже, из-за чего итоговый F1-score оказывается средним. Направление «Защитный вклад» выглядит наиболее ровным: три столбца почти одинаковые и находятся примерно на одном уровне.

По графику можно сделать вывод, что модель лучше всего определяет направления, связанные с трехочковым броском, контролем мяча и

штрафными. Направления «Атака кольца» и «Принятие решений» требуют большего внимания, так как по ним качество менее устойчивое. Это может указывать на необходимость расширения обучающих примеров именно по этим типам рекомендаций.

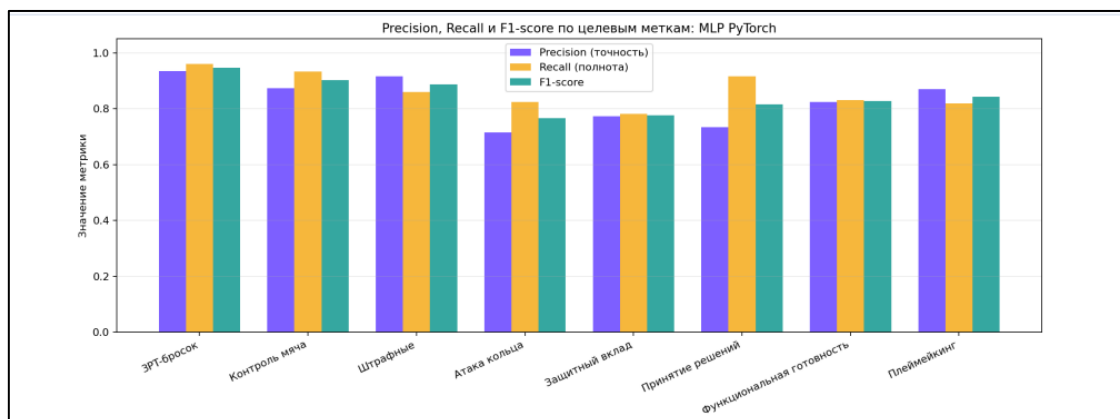


Рисунок 37 — Precision, Recall и F1-score по целевым меткам

График «Precision-Recall кривые» (рисунок 38) показывает качество работы модели по отдельным направлениям рекомендаций при изменении порога отбора. На рисунке для каждой целевой метки построена своя кривая, поэтому можно увидеть, насколько уверенно модель разделяет положительные и отрицательные примеры по каждому направлению.

По графику видно, что часть кривых проходит ближе к верхней границе поля и дольше сохраняет высокие значения precision при росте recall. Это говорит о более уверенном распознавании соответствующих рекомендаций. Наиболее устойчиво выглядят кривые для направлений, которые и на предыдущем графике показывали высокое качество, в частности для трехочкового броска и контроля мяча. Для них площадь под кривой больше, а снижение точности происходит более плавно.

Для некоторых направлений кривые расположены ниже и раньше начинают снижаться. Это означает, что при попытке охватить больше положительных примеров модель быстрее теряет точность. На графике такие различия особенно заметны у менее устойчиво определяемых рекомендаций, например связанных с атакой кольца или принятием решений. По ним качество остается приемлемым, но разделение классов выражено слабее.

На этом графике повторяется та же закономерность: трехочковый бросок и контроль мяча модель определяет устойчивее, а для направлений «Атака кольца» и «Принятие решений» точность снижается быстрее при расширении охвата положительных примеров. Поэтому при дальнейшей настройке модели именно эти направления требуют дополнительной проверки.

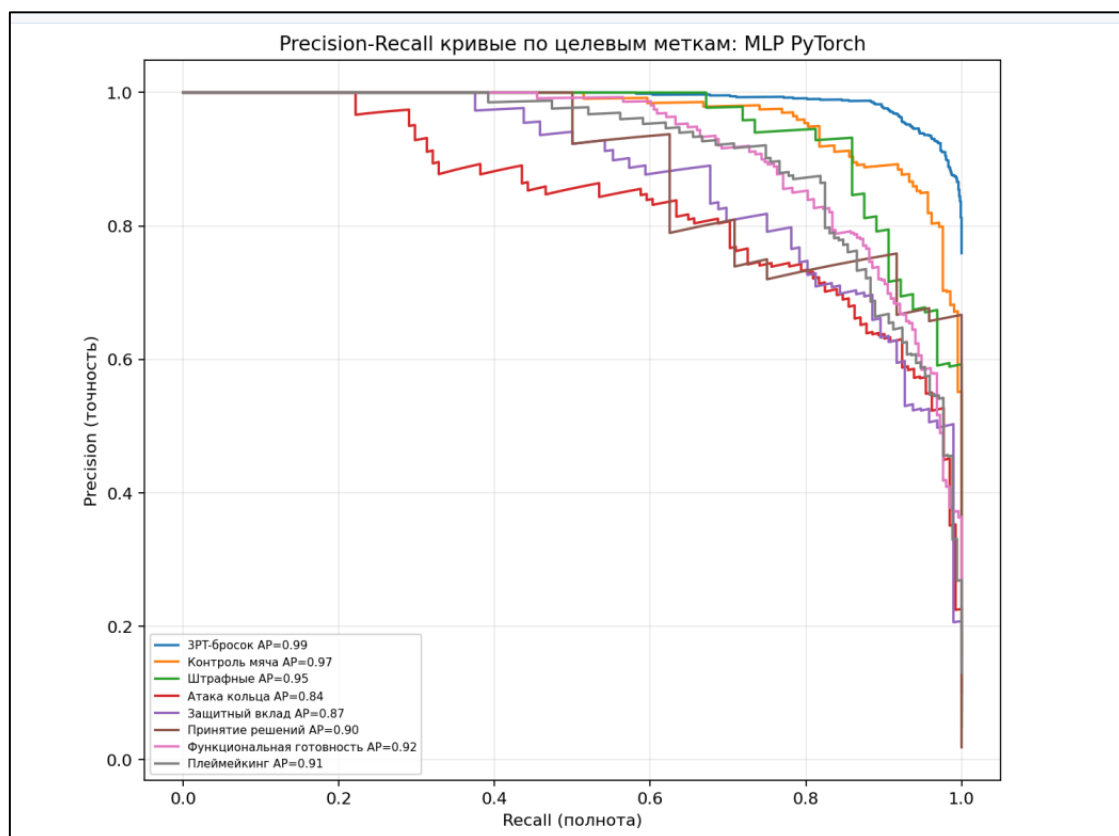


Рисунок 38 — Precision-Recall кривые

На рисунке 39 представлены ROC-кривые для отдельных направлений рекомендаций. Большая часть линий проходит выше диагонали случайного выбора, поэтому модель различает положительные и отрицательные примеры лучше случайного варианта.

Выше остальных расположены кривые для направлений «ЗРТ-бросок», «Контроль мяча» и «Штрафные». Они быстрее поднимаются к верхней части графика и дольше сохраняют высокий уровень, что подтверждает более уверенное распознавание этих рекомендаций. Кривые для «Атаки кольца» и «Принятия решений» проходят ниже: модель распознает эти направления, но делает это менее устойчиво.

ROC-график подтверждает выводы предыдущих проверок: технически выраженные проблемы, связанные с броском и контролем мяча, определяются лучше, чем более комплексные игровые ситуации.

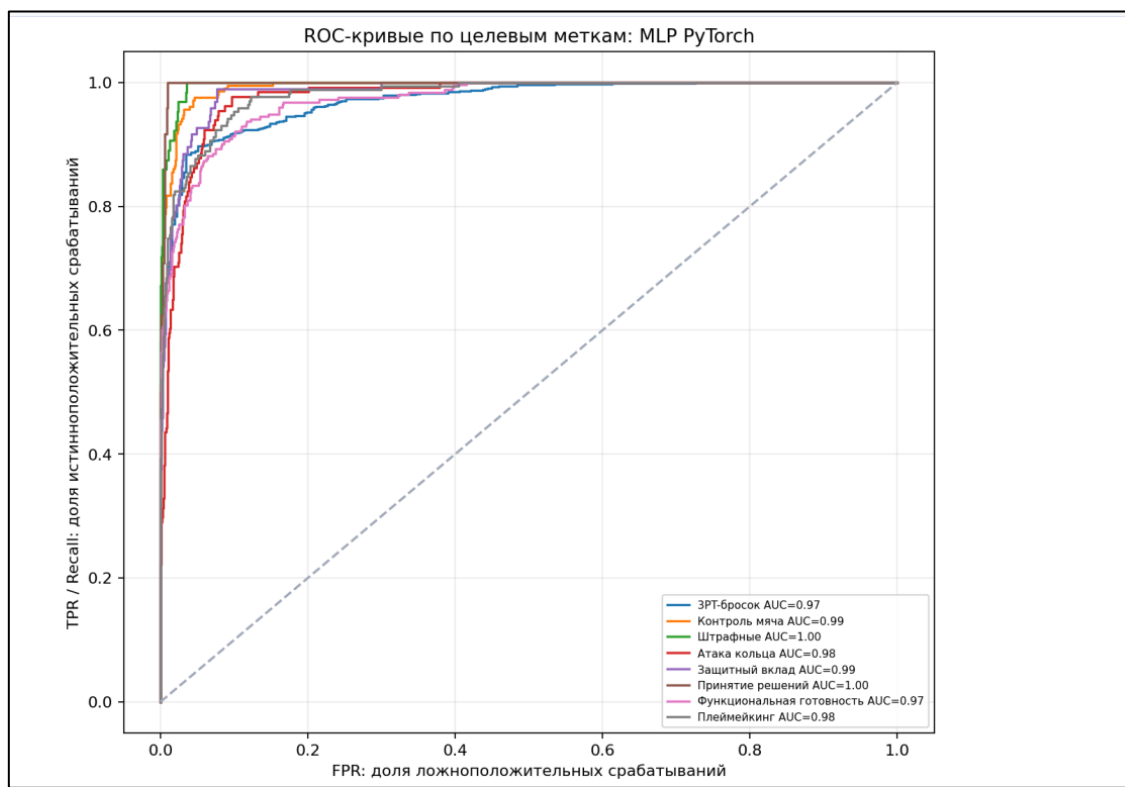


Рисунок 39 — ROC-кривые по направлениям рекомендаций

График «Информативность признаков» (рисунок 40) показывает, какие входные показатели сильнее остальных связаны с формированием рекомендаций. На рисунке признаки расположены по убыванию значимости: самые важные находятся в верхней части графика, а менее значимые — ниже. За счет этого сразу видно, на какие статистические характеристики модель опирается в первую очередь.

По графику видно, что наибольший вклад вносят показатели, непосредственно связанные с качеством игровых действий: процент трехочковых бросков, количество потерь, отношение передач к потерям, точность бросков и показатели защитной активности. Более высокое положение этих признаков означает, что именно они чаще всего влияют на выбор направлений развития игрока. Ниже располагаются признаки, которые

тоже участвуют в анализе, но оказывают меньшее влияние на итоговый результат.

Этот график позволяет связать рекомендации с конкретными данными игрока. Если среди наиболее значимых признаков находятся `three_pct`, `turnovers` и `ast_tov_ratio`, то это хорошо согласуется с рекомендациями по улучшению дальнего броска, контролю мяча и принятию решений. За счет этого работа ML-модуля выглядит более прозрачной: видно, что выбор рекомендаций основан на статистических показателях, а не формируется произвольно.

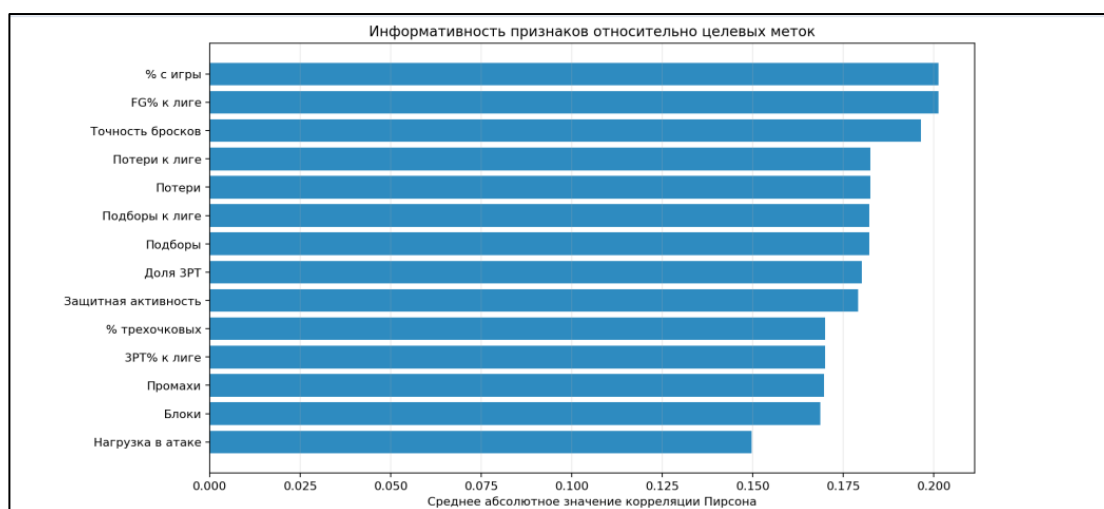


Рисунок 40 — Информативность признаков

График «Распределение целевых меток» (рисунок 41) показывает, сколько обучающих примеров относится к каждому направлению рекомендаций. По оси X расположены целевые метки, а по оси Y — количество примеров. Столбцы на графике имеют разную высоту, поэтому видно, что обучающий набор распределен по направлениям неравномерно.

Наибольшее число примеров приходится на направления, связанные с контролем мяча, трехочковым броском и принятием решений. Эти метки представлены лучше остальных, поэтому модель получает больше данных для их распознавания. Ниже расположены столбцы, связанные с менее частыми направлениями, например развитием физической готовности или отдельными игровыми компонентами, которые встречаются реже.

По рисунку видно, что в обучающей выборке есть дисбаланс между направлениями рекомендаций. Это объясняет различия качества, которые были заметны на предыдущих графиках: по более представленным меткам модель работает устойчивее, а по редким направлениям может показывать менее стабильный результат.

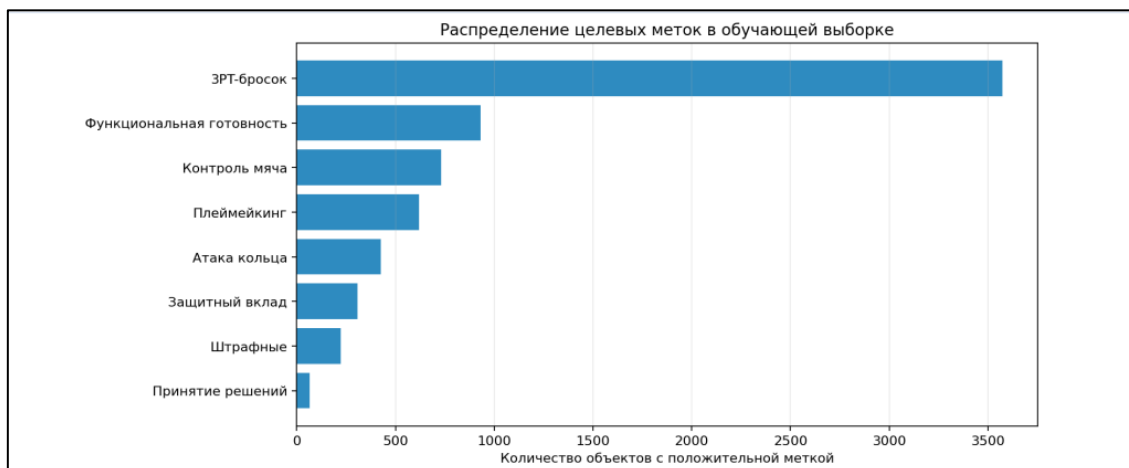


Рисунок 41 — Распределение целевых меток

График «Матрицы ошибок бинарных классификаторов» (рисунок 42) показывает результаты работы модели отдельно по каждому направлению рекомендаций. Для каждой метки на рисунке приведена своя матрица ошибок, в которой видно, сколько случаев модель определила правильно и где допустила ошибки. По этим матрицам можно понять, какие направления распознаются устойчиво, а по каким модель чаще ошибается.

Если в матрице преобладают значения на главной диагонали, это означает, что модель в большинстве случаев верно определяет наличие или отсутствие соответствующей рекомендации. Такие матрицы указывают на более надежную работу по данному направлению. Если же заметна повышенная доля ошибок вне диагонали, это говорит о менее точной классификации. В одних случаях модель чаще добавляет лишнюю рекомендацию, в других — пропускает ситуацию, когда рекомендация действительно нужна.

Матрицы ошибок подтверждают результаты предыдущих графиков. Для направлений «ЗРТ-бросок», «Контроль мяча» и «Штрафные» основная часть

значений сосредоточена на главной диагонали, то есть модель чаще дает правильный результат. Для «Атаки кольца» и «Принятия решений» ошибок вне диагонали больше. Эти направления требуют дополнительной настройки, так как они зависят не от одного статистического показателя, а от более сложного сочетания признаков.

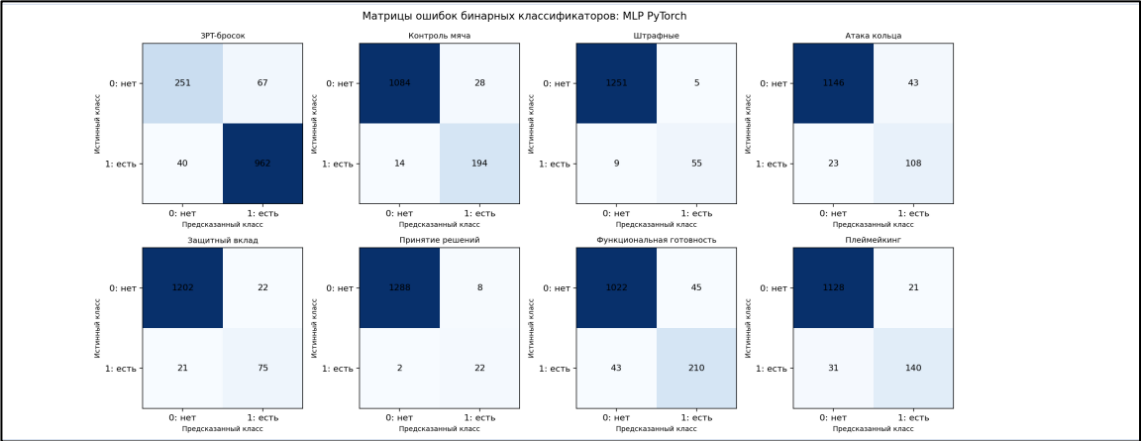


Рисунок 42 — Матрицы ошибок бинарных классификаторов

В ходе тестирования отдельно проверялось окно «Метрики и графики обучения». Оно открывается из вкладки «Рекомендации ИИ» и выводит подготовленные изображения отчета прямо в интерфейсе приложения, без ручного открытия файлов из каталога проекта.

Графики помогают оценить работу ML-модуля до просмотра итоговых рекомендаций. По ним видно, как проходило обучение модели, какие направления рекомендаций определяются лучше, какие признаки сильнее влияют на результат и где чаще возникают ошибки. Это дает возможность связать выданные рекомендации с качеством работы модели, а не рассматривать их как необъяснимый итог.

5.4 Анализ результатов работы системы

Проверка показала, что в RateMyPlayer работает полный сценарий анализа игрока: данные добавляются в систему, сохраняются в базе и повторно используются без дополнительного импорта. При выборе спортсмена приложение получает связанные с ним сведения и передает их в нужные разделы интерфейса.

Результаты тестирования подтвердили корректную работу основных аналитических функций. Система отображает профиль и статистику игрока, формирует сравнительный рейтинг, строит карту бросков по координатам попыток и выводит рекомендации через вкладку «Рекомендации ИИ». Отдельная проверка диагностических графиков показала, что результаты обучения ML-модуля можно просмотреть внутри приложения и сопоставить с итоговыми рекомендациями.

В целом разработанная система выполняет задачи, поставленные в работе: хранит баскетбольные данные, обрабатывает их на серверной стороне, отображает результаты анализа и формирует рекомендации по игровому профилю баскетболиста. Это подтверждает работоспособность RateMyPlayer как локального инструмента для оценки эффективности спортсмена.

ЗАКЛЮЧЕНИЕ

В ходе магистерской диссертации была разработана система RateMyPlayer для оценки эффективности спортсмена в игровых видах спорта. Практическая часть выполнена на примере баскетбола, где результат игрока можно описать набором формализованных показателей: очками, передачами, подборами, потерями, точностью бросков и защитными действиями.

В работе рассмотрена предметная область спортивной аналитики и существующие решения: трекинговые системы, видеоаналитические платформы, статистические сервисы и открытые источники данных. Анализ показал, что такие инструменты часто требуют оборудования, зависят от ручной разметки или не позволяют свободно менять алгоритмы оценки под локальную задачу.

Для RateMyPlayer определены источники данных и структура хранения. Информация об игроках, сезонной статистике и бросках сохраняется в SQLite, а взаимодействие с базой организовано через SQLAlchemy. Такой подход позволяет повторно использовать данные при расчете рейтинга, построении карты бросков и работе ML-модуля.

Система реализована как клиент-серверное приложение. FastAPI и Uvicorn используются для серверной части, PySide6 — для графического интерфейса. Пользователь может работать с карточками игроков, загружать данные, просматривать статистику, рейтинг, карту бросков и рекомендации.

Разработанный аналитический модуль рассчитывает атакующий и защитный рейтинги, общую эффективность и стабильность. ML-модуль формирует признаки на основе статистики, позиции и бросков игрока, а затем определяет вероятные направления развития. Итоговые рекомендации строятся по гибридной схеме: результаты модели уточняются экспертными правилами.

Тестирование подтвердило работоспособность основных функций: загрузки и хранения данных, расчета рейтинга, визуализации бросков, работы ML-модуля и отображения рекомендаций. Дополнительно были рассмотрены

метрики и графики обучения, которые помогают оценить качество интеллектуального компонента.

Практическая значимость RateMyPlayer заключается в том, что система помогает анализировать индивидуальные показатели баскетболистов, находить сильные стороны и зоны роста, сравнивать игроков и выбирать направления для дальнейшей тренировки.

Дальнейшее развитие системы может быть связано с расширением набора спортивных данных, увеличением обучающей выборки, улучшением модели рекомендаций, добавлением других игровых видов спорта и доработкой интерфейса.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Альмар, Б. Спортивная аналитика: руководство для тренеров, менеджеров и лиц, принимающих решения / Б. Альмар. – New York: Columbia University Press, 2013. – 152 p.
2. Оливер, Д. Баскетбол на бумаге: правила и инструменты анализа эффективности / Д. Оливер. – Washington: Potomac Books, 2004. – 392 p.
3. Кубатко, Дж. Отправная точка для анализа баскетбольной статистики / Дж. Кубатко, Д. Оливер, К. Пелтон, Д. Т. Розенбаум // Journal of Quantitative Analysis in Sports. – 2007. – Vol. 3, № 3. – P. 1–24. – DOI: 10.2202/1559-0410.1070.
4. Безмылов, Н. Н. Оценка соревновательной деятельности баскетболистов высокого класса в игровом сезоне: монография / Н. Н. Безмылов, О. А. Шинкарук. – Киев: НВП Полиграфсервис, 2013. – 144 с.
5. Касюк, С. Т. Использование нейронных сетей для анализа и прогнозирования данных в физической культуре и спорте / С. Т. Касюк, Е. М. Вахтомова // [электронный ресурс]. – URL: <https://cyberleninka.ru/article/n/ispolzovanie-neyronnyh-setey-dlya-analiza-i-prognozirovaniya-dannyh-v-fizicheskoy-kulture-i-sporte>.
6. Бойко, Г. М. Применение искусственного интеллекта и его помощь игрокам и тренерам в спорте / Г. М. Бойко, М. Г. Пурыгина // Молодой ученый. – 2021. – № 50 (392). – С. 578–581. – URL: <https://moluch.ru/archive/392/86722>.
7. Банкер, Р. П. Применение методов машинного обучения для прогнозирования результатов в командных видах спорта: обзор / Р. П. Банкер, Т. Сусняк // Journal of Artificial Intelligence Research. – 2022. – Vol. 73. – P. 1285–1322. – DOI: 10.1613/jair.1.13509.
8. Гудмундссон, Й. Пространственно-временной анализ командных видов спорта / Й. Гудмундссон, М. Хортон // ACM Computing Surveys. – 2017. – Vol. 50, № 2. – Article 22. – DOI: 10.1145/3054132.
9. Second Spectrum: официальный сайт. – URL: <https://www.secondspectrum.com>.

10. Catapult Athlete Monitoring: официальный сайт. – URL: <https://www.catapult.com>.
11. Hudl Sportscode; Wyscout Platform: официальные сайты. – URL: <https://www.hudl.com/products/sportscode>; URL: <https://wyscout.hudl.com>.
12. Synergy Sports: официальный сайт. – URL: <https://www.sportradar.com/synergy-sports>.
13. Stats Perform / Opta: официальный сайт. – URL: <https://www.statsperform.com/opta>.
14. ESPN Public API Documentation: репозиторий. – URL: <https://github.com/pseudo-r/Public-ESPN-API>.
15. Маккинни, У. Python и анализ данных / У. Маккинни. – 3-е изд. – Sebastopol: O'Reilly Media, 2022. – 579 p.
16. Харрис, Ч. Р. Программирование массивов с NumPy / Ч. Р. Харрис, К. Дж. Миллман, С. ван дер Валт, Р. Гоммерс, П. Виртанен и др. // Nature. – 2020. – Vol. 585. – P. 357–362. – DOI: 10.1038/s41586-020-2649-2.
17. Пашке, А. PyTorch: императивная высокопроизводительная библиотека глубокого обучения / А. Пашке, С. Гросс, Ф. Масса, А. Лерер, Дж. Брэдли, Г. Чанан и др. // Advances in Neural Information Processing Systems. – 2019. – Vol. 32. – P. 8024–8035.
18. Гудфеллоу, И. Глубокое обучение / И. Гудфеллоу, Й. Бенджио, А. Курвилль. – Cambridge: MIT Press, 2016. – 800 p.
19. Педрегоса, Ф. Scikit-learn: машинное обучение в Python / Ф. Педрегоса, Г. Вароко, А. Грамфор, В. Мишель, Б. Тирион, О. Гризель, М. Блондель и др. // Journal of Machine Learning Research. – 2011. – Vol. 12. – P. 2825–2830.
20. Хантер, Дж. Д. Matplotlib: среда двумерной графики / Дж. Д. Хантер // Computing in Science & Engineering. – 2007. – Vol. 9, № 3. – P. 90–95. – DOI: 10.1109/MCSE.2007.55.

ПРИЛОЖЕНИЕ А

(обязательное)

Интеллектуальная система оценки эффективности спортсмена в

игровых видах спорта

Листинг программы

Файл « feature_engineering.py»

```
from __future__ import annotations

import re
from typing import Any, Iterable

from server.services.league_average_service import league_averages_for_season


FEATURE_COLUMNS = [
    "points",
    "assists",
    "rebounds",
    "steals",
    "blocks",
    "turnovers",
    "fg_pct",
    "three_pct",
    "ft_pct",
    "shot_distribution",
    "made_shots",
    "missed_shots",
    "points_trend",
    "assists_trend",
    "rebounds_trend",
    "turnovers_trend",
    "fg_pct_trend",
    "three_pct_trend",
    "points_vs_league",
    "assists_vs_league",
    "rebounds_vs_league",
    "turnovers_vs_league",
    "fg_pct_vs_league",
    "three_pct_vs_league",
```

```

    "ft_pct_vs_league",
    "ast_tov_ratio",
    "shot_accuracy",
    "defensive_activity",
    "scoring_load",
    "efficiency_proxy",
    "position",
]

def feature_row_from_stats(
    stats: Any,
    shots: Iterable[Any],
    position: str,
    history: Iterable[Any] | None = None,
) -> dict[str, Any]:
    summary = shot_summary(shots)
    history_rows = _sorted_history(list(history or [stats]))
    averages = league_averages_for_season(getattr(stats,
"season", None))

    row = {
        "points": float(getattr(stats, "points", 0.0) or 0.0),
        "assists": float(getattr(stats, "assists", 0.0) or 0.0),
        "rebounds": float(getattr(stats, "rebounds", 0.0) or 0.0),
        "steals": float(getattr(stats, "steals", 0.0) or 0.0),
        "blocks": float(getattr(stats, "blocks", 0.0) or 0.0),
        "turnovers": float(getattr(stats, "turnovers", 0.0) or
0.0),
        "fg_pct": float(getattr(stats, "fg_pct", 0.0) or 0.0),
        "three_pct": float(getattr(stats, "three_pct", 0.0) or
0.0),
        "ft_pct": float(getattr(stats, "ft_pct", 0.0) or 0.0),
        "shot_distribution": summary["three_rate"],
        "made_shots": summary["made_shots"],

```

```

"missed_shots": summary["missed_shots"],
"points_trend": _trend(history_rows, "points"),
"assists_trend": _trend(history_rows, "assists"),
"rebounds_trend": _trend(history_rows, "rebounds"),
"turnovers_trend": _trend(history_rows, "turnovers"),
"fg_pct_trend": _trend(history_rows, "fg_pct"),
"three_pct_trend": _trend(history_rows, "three_pct"),
"points_vs_league": _relative_delta(
    float(getattr(stats, "points", 0.0) or 0.0),
    averages.get("points"),
),
"assists_vs_league": _relative_delta(
    float(getattr(stats, "assists", 0.0) or 0.0),
    averages.get("assists"),
),
"rebounds_vs_league": _relative_delta(
    float(getattr(stats, "rebounds", 0.0) or 0.0),
    averages.get("rebounds"),
),
"turnovers_vs_league": _relative_delta(
    float(getattr(stats, "turnovers", 0.0) or 0.0),
    averages.get("turnovers"),
),
"fg_pct_vs_league": _relative_delta(
    float(getattr(stats, "fg_pct", 0.0) or 0.0),
    averages.get("fg_pct"),
),
"three_pct_vs_league": _relative_delta(
    float(getattr(stats, "three_pct", 0.0) or 0.0),
    averages.get("three_pct"),
),
"ft_pct_vs_league": _relative_delta(
    float(getattr(stats, "ft_pct", 0.0) or 0.0),
    averages.get("ft_pct"),
),

```

```

        "position": str(position or "SF").upper(),
    }

    return complete_derived_features(row)

def complete_derived_features(row: dict[str, Any]) -> dict[str, Any]:
    completed = dict(row)

    raw_made = float(completed.get("made_shots", 0.0) or 0.0)
    raw_missed = float(completed.get("missed_shots", 0.0) or 0.0)
    raw_attempts = raw_made + raw_missed

    if raw_attempts > 140:
        scale = 140 / raw_attempts
        completed["made_shots"] = round(raw_made * scale, 2)
        completed["missed_shots"] = round(raw_missed * scale, 2)
    attempts = (
        float(completed.get("made_shots", 0.0) or 0.0)
        + float(completed.get("missed_shots", 0.0) or 0.0)
    )
    assists = float(completed.get("assists", 0.0) or 0.0)
    turnovers = float(completed.get("turnovers", 0.0) or 0.0)
    points = float(completed.get("points", 0.0) or 0.0)
    rebounds = float(completed.get("rebounds", 0.0) or 0.0)
    steals = float(completed.get("steals", 0.0) or 0.0)
    blocks = float(completed.get("blocks", 0.0) or 0.0)
    fg_pct = float(completed.get("fg_pct", 0.0) or 0.0)
    three_pct = float(completed.get("three_pct", 0.0) or 0.0)
    ft_pct = float(completed.get("ft_pct", 0.0) or 0.0)

    completed["ast_tov_ratio"] = round(assists / max(turnovers,
0.15), 4)
    completed["shot_accuracy"] = (

```

```

        round(float(completed.get("made_shots", 0.0) or 0.0) /
attempts, 4)
        if attempts else fg_pct
    )
    completed["defensive_activity"] = round(
        steals + blocks + rebounds * 0.18,
        4,
    )
    completed["scoring_load"] = round(
        points + attempts * 0.08 + turnovers * 0.65,
        4,
    )
    completed["efficiency_proxy"] = round(
        points
        + rebounds * 0.7
        + assists * 0.9
        + steals * 1.2
        + blocks * 1.2
        - turnovers * 1.35
        + fg_pct * 10
        + three_pct * 4
        + ft_pct * 2,
        4,
    )
    completed["position"] = str(completed.get("position") or
"SF").upper()

    return {
        column: completed.get(column, 0.0 if column != "position"
else "SF")
        for column in FEATURE_COLUMNS
    }
def shot_summary(shots: Iterable[Any]) -> dict[str, Any]:
    shot_list = list(shots)
    attempts = len(shot_list)

```

```

made = sum(
    1 for shot in shot_list
    if str(getattr(shot, "result", "")).lower() == "made"
)
three_attempts = sum(
    1 for shot in shot_list
    if str(getattr(shot, "shot_type", "")).upper() == "3PT"
)
return {
    "attempts": attempts,
    "made_shots": made,
    "missed_shots": attempts - made,
    "three_rate": round(three_attempts / attempts, 3) if
attempts else 0.0,
}
def _sorted_history(history: list[Any]) -> list[Any]:
    return sorted(
        history,
        key=lambda item: (
            _season_year(getattr(item, "season", "")),
            getattr(item, "id", 0) or 0,
        ),
    )
def _trend(history: list[Any], key: str) -> float:
    if len(history) < 2:
        return 0.0
    first = float(getattr(history[0], key, 0.0) or 0.0)
    last = float(getattr(history[-1], key, 0.0) or 0.0)

    return round((last - first) / max(1, len(history) - 1), 4)
def _season_year(season: str) -> int:
    match = re.search(r"(20\d{2})", str(season or ""))
    return int(match.group(1)) if match else
def _relative_delta(value: float, baseline: float | None) ->
float:

```

```

if not baseline:
    return 0.0
return round((float(value or 0.0) - baseline) / baseline, 4)

```

Файл « model.py»

```

from __future__ import annotations

from pathlib import Path
from typing import Any
from datetime import datetime, timezone

import joblib
import numpy as np
import pandas as pd

from sklearn.compose import ColumnTransformer
from sklearn.metrics import f1_score, hamming_loss
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler

from ml.dataset import FEATURE_COLUMNS, RECOMMENDATION_LABELS

DEFAULT_MODEL_PATH = (
    Path(__file__).resolve().parent
    / "saved_models"
    / "recommender.joblib"
)

DEFAULT_THRESHOLD = 0.42

class BasketballRecommendationModel:
    def __init__(self) -> None:
        numeric_features = [
            column for column in FEATURE_COLUMNS
            if column != "position"
        ]

        preprocessor = ColumnTransformer(
            transformers=[
                ("num", StandardScaler(), numeric_features),
                (
                    "cat",
                    OneHotEncoder(
                        handle_unknown="ignore",
                        sparse_output=False,
                    ),
                ),
            ],

```

```

        ["position"],
    ),
]
)

classifier = MLPClassifier(
    hidden_layer_sizes=(128, 64, 32),
    activation="relu",
    alpha=0.0008,
    learning_rate_init=0.0015,
    max_iter=650,
    early_stopping=True,
    validation_fraction=0.14,
    n_iter_no_change=24,
    random_state=42,
)

self.pipeline = Pipeline(
    steps=[
        ("preprocessor", preprocessor),
        ("classifier", classifier),
    ]
)

self.threshold = DEFAULT_THRESHOLD
self.thresholds = np.full(
    len(RECOMMENDATION_LABELS),
    self.threshold,
    dtype=float,
)

self.metadata: dict[str, Any] = {
    "model_type": "sklearn_mlp_multilabel_advanced",
    "hidden_layers": [128, 64, 32],
    "threshold": self.threshold,
    "thresholds": self.thresholds.tolist(),
    "feature_columns": FEATURE_COLUMNS,
    "labels": RECOMMENDATION_LABELS,
}

def fit(
    self,
    x: pd.DataFrame,
    y: pd.DataFrame,
) -> dict[str, float | int | str]:
    x_train, x_test, y_train, y_test = train_test_split(
        x,
        y,
        test_size=0.22,
        random_state=42,
    )

```

```

        return self.fit_on_split(
            x_train,
            y_train,
            x_test,
            y_test,
            total_samples=len(x),
        )

    def fit_on_split(
        self,
        x_train: pd.DataFrame,
        y_train: pd.DataFrame,
        x_test: pd.DataFrame,
        y_test: pd.DataFrame,
        total_samples: int | None = None,
    ) -> dict[str, float | int | str]:
        self.pipeline.fit(x_train, y_train)

        probabilities = self._predict_probability_matrix(x_test)

        try:
            from ml.training_report import optimize_thresholds

            self.thresholds = optimize_thresholds(
                y_test.to_numpy(dtype=int),
                probabilities,
            )
        except Exception:
            self.thresholds = np.full(
                len(RECOMMENDATION_LABELS),
                self.threshold,
                dtype=float,
            )

        predicted = (probabilities >=
self.thresholds).astype(int)
        y_true = y_test.to_numpy(dtype=int)

        exact_match = float(
            np.mean(np.all(predicted == y_test.to_numpy(),
axis=1))
        )
        loss = float(hamming_loss(y_true, predicted))

        metrics: dict[str, float | int | str] = {
            "samples": int(total_samples or len(x_train) +
len(x_test)),
            "exact_match_accuracy": round(exact_match, 3),
            "hamming_loss": round(loss, 3),
            "micro_f1": round(
                float(
                    f1_score(

```

```

        y_true,
        predicted,
        average="micro",
        zero_division=0,
    )
    ),
    3,
),
"macro_f1": round(
    float(
        f1_score(
            y_true,
            predicted,
            average="macro",
            zero_division=0,
        )
    ),
    3,
),
"threshold": self.threshold,
"mean_threshold":
round(float(np.mean(self.thresholds)), 3),
"feature_count": len(FEATURE_COLUMNS),
"label_count": len(RECOMMENDATION_LABELS),
"iterations": int(
    getattr(
        self.pipeline.named_steps["classifier"],
        "n_iter_",
        0,
    )
    or 0
),
}

self.metadata = {
    **self.metadata,
    **metrics,
    "thresholds": self.thresholds.tolist(),
    "loss_curve": [
        round(float(item), 6)
        for item in getattr(
            self.pipeline.named_steps["classifier"],
            "loss_curve_",
            [],
        )
    ],
    "trained_at": datetime.now(timezone.utc).isoformat(
        timespec="seconds"
    ),
}

return metrics

```

```

def predict_scores(
    self,
    row: dict[str, Any],
    threshold: float | None = None,
) -> list[dict[str, Any]]:
    thresholds = (
        self.thresholds
        if threshold is None
        else np.full(len(RECOMMENDATION_LABELS), threshold,
dtype=float)
    )

    frame = pd.DataFrame(
        [_normalize_row(row)],
        columns=FEATURE_COLUMNS,
    )

    probabilities = self._predict_probabilities(frame)

    return [
        {
            "label": label,
            "probability": round(float(probability), 4),
            "selected": bool(
                float(probability) >= float(threshold_value)
            ),
        }
        for label, probability, threshold_value
        in zip(
            RECOMMENDATION_LABELS,
            probabilities,
            thresholds,
            strict=True,
        )
    ]

def _predict_probabilities(self, frame: pd.DataFrame) ->
np.ndarray:
    matrix = self._predict_probability_matrix(frame)

    return matrix[0]

def _predict_probability_matrix(self, frame: pd.DataFrame) -
> np.ndarray:
    classifier = self.pipeline.named_steps["classifier"]
    transformed =
self.pipeline.named_steps["preprocessor"].transform(frame)
    raw = classifier.predict_proba(transformed)

    if isinstance(raw, list):
        return np.vstack([item[:, 1] for item in raw]).T

```

```

matrix = np.asarray(raw, dtype=float)

if matrix.ndim == 1:
    matrix = matrix.reshape(1, -1)

return matrix

def save(self, path: Path = DEFAULT_MODEL_PATH) -> None:
    path.parent.mkdir(parents=True, exist_ok=True)

    joblib.dump(
        {
            "pipeline": self.pipeline,
            "metadata": self.metadata,
        },
        path,
    )

@classmethod
def load(
    cls,
    path: Path = DEFAULT_MODEL_PATH,
) -> "BasketballRecommendationModel":
    model = cls.__new__(cls)
    payload = joblib.load(path)

    if isinstance(payload, dict) and "pipeline" in payload:
        model.pipeline = payload["pipeline"]
        model.metadata = dict(payload.get("metadata") or {})
        model.threshold = float(
            model.metadata.get("threshold") or
DEFAULT_THRESHOLD
        )

        raw_thresholds = model.metadata.get("thresholds")
        model.thresholds = (
            np.asarray(raw_thresholds, dtype=float)
            if raw_thresholds
            else np.full(
                len(RECOMMENDATION_LABELS),
                model.threshold,
            )
        )
    else:
        model.pipeline = payload
        model.threshold = DEFAULT_THRESHOLD
        model.thresholds = np.full(
            len(RECOMMENDATION_LABELS),
            model.threshold,
        )
        model.metadata = {

```

```

        "model_type": "sklearn_mlp_multilabel_legacy",
        "threshold": model.threshold,
        "thresholds": model.thresholds.tolist(),
        "feature_columns": [],
        "labels": RECOMMENDATION_LABELS,
        "legacy_model": True,
    }

    return model

    def is_compatible(self) -> bool:
        return list(self.metadata.get("feature_columns") or [])
    == FEATURE_COLUMNS

def _normalize_row(row: dict[str, Any]) -> dict[str, Any]:
    normalized = {
        column: row.get(column, 0.0)
        for column in FEATURE_COLUMNS
    }

    normalized["position"] = str(
        normalized.get("position") or "Не указана"
    ).upper()

    return normalized

```

Файл « recommender.py»

```

from __future__ import annotations

from typing import Any, Iterable

from ml.dataset import RECOMMENDATION_LABELS
from ml.feature_engineering import (
    feature_row_from_stats as build_feature_row_from_stats,
)
from ml.feature_engineering import shot_summary as
build_shot_summary
from ml.model import DEFAULT_MODEL_PATH,
BasketballRecommendationModel
from ml.torch_model import (
    DEFAULT_TORCH_MODEL_PATH,
    TorchRecommendationModel,
    is_torch_available,
)
from server.db.models import Player, PlayerStats, Shot
from server.schemas.player import (
    MLPredictRequest,
    RecommendationResponse,
    ShotCreate,
    StatsCreate,
)

```

```

LABEL_TITLES = {
    "improve_three_point": "Трехочковый бросок",
    "reduce_turnovers": "Контроль мяча",
    "improve_free_throws": "Штрафные броски",
    "attack_paint_more": "Атака краски",
    "develop_defense": "Защитная работа",
    "improve_decision_making": "Принятие решений",
    "improve_conditioning": "Физическая готовность",
    "develop_playmaking": "Пас и видение площадки",
}

LABEL_TEXTS = {
    "improve_three_point": (
        "Уделить отдельный блок тренировок трехочковому броску:
        "
        "работа с ногами, быстрый выпуск и серии из углов."
    ),
    "reduce_turnovers": (
        "Снизить количество потерь: добавить упражнения на
        контроль мяча "
        "под давлением и чтение ловушек."
    ),
    "improve_free_throws": (
        "Улучшить штрафные: зафиксировать рутину перед броском "
        "и вести учет серий после нагрузки."
    ),
    "attack_paint_more": (
        "Чаще атаковать краску и ближнюю дистанцию, если дальний
        бросок "
        "или общий процент нестабилен."
    ),
    "develop_defense": (
        "Развивать игру в защите: стойка, работа ног, закрывание
        линий "
        "передачи и подбор после контакта."
    ),
    "improve_decision_making": (
        "Улучшить принятие решений: разбор владений, чтение пик-
        н-ролла "
        "и выбор между пасом и броском."
    ),
    "improve_conditioning": (
        "Работать над физической готовностью, чтобы процент
        бросков "
        "меньше проседал при высокой нагрузке."
    ),
    "develop_playmaking": (
        "Развивать пас и видение площадки: упражнения на weak-
        side передачи "

```

```

        "и быстрый перевод мяча."
    ),
}

class RuleBasedRecommender:
    def recommend(
        self,
        stats: StatsCreate | PlayerStats | None,
        shots: Iterable[Shot | ShotCreate],
        position: str,
    ) -> tuple[list[str], list[str]]:
        labels: list[str] = []
        shot_summary = _shot_summary(shots)

        if stats is None:
            return (
                ["add_statistics"],
                [
                    "Добавьте статистику игрока, чтобы система
смогла "
                    "построить персональные рекомендации."
                ],
            )

        if stats.three_pct < 0.33 and position.upper() != "C":
            labels.append("improve_three_point")

        if stats.turnovers > 3:
            labels.extend(["reduce_turnovers",
"improve_decision_making"])

        if stats.ft_pct < 0.70:
            labels.append("improve_free_throws")

        if stats.fg_pct < 0.44 and shot_summary["three_rate"] >
0.42:
            labels.append("attack_paint_more")

        if stats.steals < 0.8 and stats.blocks < 0.6:
            labels.append("develop_defense")

        if shot_summary["missed_shots"] > max(
            8,
            shot_summary["made_shots"] * 1.3,
        ):
            labels.append("improve_conditioning")

        if position.upper() in {"PG", "SG"} and stats.assists <
4.5:
            labels.append("develop_playmaking")

```

```

labels = _deduplicate(labels)

recommendations = [
    LABEL_TEXTS[label]
    for label in labels
    if label in LABEL_TEXTS
]

if not recommendations:
    recommendations.append(
        "Текущий профиль выглядит сбалансированным. "
        "Поддерживайте нагрузку и фокусируйтесь на
стабильности."
    )

return labels, recommendations


class HybridRecommender:
    def __init__(self) -> None:
        self.rule_based = RuleBasedRecommender()
        self.model = self._load_model()

    def recommend_for_player(
        self,
        player: Player,
        stats: PlayerStats | None,
        shots: Iterable[Shot],
    ) -> dict[str, Any]:
        labels, recommendations, scores, model_info, ml_profile
= self._recommend(
            stats,
            shots,
            player.position,
            history=player.stats,
        )

        return RecommendationResponse(
            player_id=player.id,
            player_name=player.full_name,
            source="hybrid_ml_rules" if self.model else
"rule_based",
            labels=labels,
            recommendations=recommendations,
            scores=scores,
            model_info=model_info,
            ml_profile=ml_profile,
        ).model_dump()

    def recommend_from_payload(
        self,
        payload: MLPredictRequest,

```

```

        ) -> dict[str, Any]:
            labels, recommendations, scores, model_info, ml_profile
= self._recommend(
                payload.stats,
                payload.shots,
                payload.position,
            )

            return RecommendationResponse(
                source="hybrid_ml_rules" if self.model else
"rule_based",
                labels=labels,
                recommendations=recommendations,
                scores=scores,
                model_info=model_info,
                ml_profile=ml_profile,
            ).model_dump()

def _recommend(
    self,
    stats: StatsCreate | PlayerStats | None,
    shots: Iterable[Shot | ShotCreate],
    position: str,
    history: Iterable[PlayerStats | StatsCreate] | None =
None,
) -> tuple[
    list[str],
    list[str],
    list[dict[str, Any]],
    dict[str, Any],
    dict[str, Any],
]:
    shot_list = list(shots)
    rule_labels, _rule_texts = self.rule_based.recommend(
        stats,
        shot_list,
        position,
    )

    model_scores: list[dict[str, Any]] = []
    model_labels: list[str] = []
    row: dict[str, Any] = {}

    if stats is not None:
        row = build_feature_row_from_stats(
            stats,
            shot_list,
            position,
            history=history,
        )

    if self.model and stats is not None:

```

```

        model_scores = self.model.predict_scores(row)
        model_labels = [
            str(score["label"])
            for score in model_scores
            if score.get("selected")
            and _label_is_contextually_supported(
                str(score.get("label")),
                row,
            )
        ]

    labels = _deduplicate([*rule_labels, *model_labels])

    recommendations = [
        LABEL_TEXTS[label]
        for label in labels
        if label in LABEL_TEXTS
    ]

    if not recommendations:
        recommendations = [
            "Недостаточно выраженных проблемных зон. "
            "Модель рекомендует поддерживать текущий режим "
            "и отслеживать динамику по сезонам."
        ]

    scores = self._build_scores(
        labels,
        rule_labels,
        model_scores,
        row,
    )
    ml_profile = self._ml_profile(row)

    return labels, recommendations, scores,
self._model_info(), ml_profile

def _build_scores(
    self,
    selected_labels: list[str],
    rule_labels: list[str],
    model_scores: list[dict[str, Any]],
    row: dict[str, Any],
) -> list[dict[str, Any]]:
    by_label = {
        str(score.get("label")): dict(score)
        for score in model_scores
    }

    result: list[dict[str, Any]] = []

    for label in RECOMMENDATION_LABELS:

```

```

0.0
        model_probability = float(
            by_label.get(label, {}).get("probability") or
        )
        is_rule = label in rule_labels
        is_selected = label in selected_labels

        rule_score = (
            _rule_strength(label, row)
            if is_rule and row
            else (0.88 if is_rule else 0.0)
        )

        probability = (
            model_probability * 0.68 + rule_score * 0.32
            if self.model
            else rule_score
        )

        if is_rule and model_probability > 0:
            probability = max(probability, 0.62)

        if not self.model and is_rule:
            probability = max(probability, 0.86)

        if row and not is_rule and not
_label_is_contextually_supported(
            label,
            row,
        ):
            probability = min(probability, 0.24)

        result.append(
            {
                "label": label,
                "title": LABEL_TITLES.get(label, label),
                "probability": round(probability, 4),
                "source": "hybrid" if is_rule and
model_probability > 0 else "ml",
                "selected": is_selected,
            }
        )

    return sorted(
        result,
        key=lambda item: (
            not item["selected"],
            -float(item["probability"]),
        ),
    )

```

```

def _ml_profile(self, row: dict[str, Any]) -> dict[str, Any]:
    if not row:
        return {}

    profile = _heuristic_profile(row)

    return {
        **profile,
        "summary": _profile_summary(profile),
    }

def _model_info(self) -> dict[str, Any]:
    if not self.model:
        return {
            "available": False,
            "model_type": "rule_based_fallback",
            "message": "Обученная MLP-модель не найдена. "
                        "Используются экспертные правила.",
        }

    metadata = getattr(self.model, "metadata", {}) or {}

    return {
        "available": True,
        "model_type": metadata.get(
            "model_type",
            "sklearn_mlp_multilabel",
        ),
        "trained_at": metadata.get("trained_at"),
        "samples": metadata.get("samples"),
        "micro_f1": metadata.get("micro_f1"),
        "macro_f1": metadata.get("macro_f1"),
        "hamming_loss": metadata.get("hamming_loss"),
        "threshold": metadata.get("threshold"),
        "feature_count": metadata.get("feature_count"),
        "label_count": metadata.get("label_count"),
    }

    @staticmethod
    def _load_model() -> Any | None:
        if is_torch_available() and
        DEFAULT_TORCH_MODEL_PATH.exists():
            try:
                model =
                TorchRecommendationModel.load(DEFAULT_TORCH_MODEL_PATH)

                if model.is_compatible():
                    return model
            except Exception:
                pass

```

```

        if not DEFAULT_MODEL_PATH.exists():
            return None

        try:
            model =
BasketballRecommendationModel.load(DEFAULT_MODEL_PATH)

            if model.is_compatible():
                return model
        except Exception:
            pass

        return None

def _label_is_contextually_supported(
    label: str,
    row: dict[str, Any],
) -> bool:
    if not row:
        return True

    if label == "improve_three_point":
        return (
            float(row.get("three_pct") or 0.0) < 0.35
            or float(row.get("three_pct_vs_league") or 0.0) < -
0.06
        )

    if label == "reduce_turnovers":
        return (
            float(row.get("turnovers") or 0.0) >= 3.0
            or float(row.get("ast_tov_ratio") or 0.0) < 2.0
        )

    if label == "improve_free_throws":
        return float(row.get("ft_pct") or 0.0) < 0.78

    if label == "develop_defense":
        return (
            float(row.get("defensive_activity") or 0.0) < 2.25
            or (
                float(row.get("steals") or 0.0) < 0.8
                and float(row.get("blocks") or 0.0) < 0.5
            )
        )

    if label == "improve_decision_making":
        return (
            float(row.get("turnovers") or 0.0) > 3.1
            and float(row.get("ast_tov_ratio") or 0.0) < 2.0
        )

```

```

return True

def _heuristic_profile(row: dict[str, Any]) -> dict[str, float]:
    offense = min(
        100.0,
        max(
            0.0,
            float(row.get("points") or 0.0) / 34 * 40
            + float(row.get("assists") or 0.0) / 11 * 22
            + float(row.get("fg_pct") or 0.0) / 0.62 * 20
            + float(row.get("three_pct") or 0.0) / 0.48 * 10
            + float(row.get("ast_tov_ratio") or 0.0) / 4.5 * 8,
        ),
    )

    defense = min(
        100.0,
        max(
            0.0,
            float(row.get("rebounds") or 0.0) / 15 * 40
            + float(row.get("steals") or 0.0) / 2.8 * 25
            + float(row.get("blocks") or 0.0) / 3.2 * 20
            + float(row.get("defensive_activity") or 0.0) / 6 *
15,
        ),
    )

    return {
        "overall_rating": round(offense * 0.58 + defense * 0.42,
1),
        "offense_profile": round(offense, 1),
        "defense_profile": round(defense, 1),
    }

def _rule_strength(label: str, row: dict[str, Any]) -> float:
    if label == "improve_three_point":
        return _severity(
            0.36 - float(row.get("three_pct") or 0.0),
            0.12,
            floor=0.45,
        )

    if label == "reduce_turnovers":
        return _severity(
            float(row.get("turnovers") or 0.0) - 2.4,
            2.2,
            floor=0.50,
        )

```

```

    if label == "improve_free_throws":
        return _severity(
            0.78 - float(row.get("ft_pct") or 0.0),
            0.18,
            floor=0.45,
        )

    if label == "develop_defense":
        activity_gap = 2.2 - float(row.get("defensive_activity")
or 0.0)

        return _severity(activity_gap, 1.8, floor=0.42)

    if label == "improve_decision_making":
        ratio_gap = 1.75 - float(row.get("ast_tov_ratio") or
0.0)

        return _severity(ratio_gap, 1.35, floor=0.46)

    return 0.0

def _severity(value: float, scale: float, floor: float = 0.35) -
> float:
    return round(
        max(
            0.0,
            min(
                0.95,
                floor
                + max(0.0, value)
                / max(scale, 0.01)
                * (0.95 - floor),
            ),
        ),
        4,
    )

def _shot_summary(shots: Iterable[Shot | ShotCreate]) ->
dict[str, Any]:
    return build_shot_summary(shots)

def _profile_summary(profile: dict[str, Any]) -> str:
    offense = float(profile.get("offense_profile") or 0.0)
    defense = float(profile.get("defense_profile") or 0.0)
    overall = float(profile.get("overall_rating") or 0.0)

    if overall >= 78 and abs(offense - defense) <= 12:
        return "Сбалансированный профиль высокого уровня."

```

```

    if offense - defense >= 14:
        return "Профиль смещен в атаку."

    if defense - offense >= 14:
        return "Профиль смещен в защиту."

    return "Профиль близок к сбалансированному."

def _deduplicate(items: Iterable[str]) -> list[str]:
    seen: set[str] = set()
    result: list[str] = []

    for item in items:
        if item not in seen:
            seen.add(item)
            result.append(item)

    return result

```

Файл « train_model.py»

```

from __future__ import annotations

import json
from pathlib import Path
from datetime import datetime, timezone

from sklearn.model_selection import train_test_split

from ml.advanced_dataset import
generate_advanced_synthetic_dataset
from ml.model import DEFAULT_MODEL_PATH,
BasketballRecommendationModel
from ml.torch_model import (
    DEFAULT_TORCH_MODEL_PATH,
    TorchRecommendationModel,
    is_torch_available,
)
from ml.training_report import ARTIFACTS_DIR,
build_research_report

def train_and_save_model(
    samples: int = 6000,
    path: Path = DEFAULT_MODEL_PATH,
    create_report: bool = True,
) -> dict:
    x, y, meta =
generate_advanced_synthetic_dataset(samples=samples)

    x_train, x_val, y_train, y_val, meta_train, meta_val =
train_test_split(

```

```

        x,
        y,
        meta,
        test_size=0.22,
        random_state=42,
        stratify=x["position"],
    )

    model = BasketballRecommendationModel()
    metrics = model.fit_on_split(
        x_train,
        y_train,
        x_val,
        y_val,
        total_samples=len(x),
    )
    model.save(path)

    torch_result: dict = {
        "available": is_torch_available(),
        "trained": False,
    }

    torch_history = None
    torch_probabilities = None
    torch_auxiliary = None
    torch_model: TorchRecommendationModel | None = None

    if torch_result["available"]:
        try:
            torch_model = TorchRecommendationModel()
            torch_history = torch_model.fit(
                x_train,
                y_train,
                x_val,
                y_val,
            )
            torch_probabilities =
torch_model.predict_probabilities(x_val)
            torch_auxiliary =
torch_model.predict_auxiliary_frame(x_val)
            torch_model.save(DEFAULT_TORCH_MODEL_PATH)

            torch_result = {
                "available": True,
                "trained": True,
                "model_path": str(DEFAULT_TORCH_MODEL_PATH),
                "epochs": len(torch_history.get("train_loss",
[[]])),
                "model_type":
torch_model.metadata.get("model_type"),

```

```

        "task_heads":
torch_model.metadata.get("task_heads", []),
        "rating_mae":
torch_model.metadata.get("rating_mae"),
        "offense_mae":
torch_model.metadata.get("offense_mae"),
        "defense_mae":
torch_model.metadata.get("defense_mae"),
    }
    except Exception as exc:
        torch_result = {
            "available": True,
            "trained": False,
            "error": str(exc),
        }

report = {}

if create_report:
    report = build_research_report(
        x_train,
        y_train,
        x_val,
        y_val,
        meta=meta_train,
        mlp_pipeline=model.pipeline,
        torch_history=torch_history,
        torch_probabilities=torch_probabilities,
        torch_auxiliary=torch_auxiliary,
        output_dir=ARTIFACTS_DIR,
    )

    model.metadata["research_report"] = {
        "summary_path": report.get("summary_path"),
        "plots": report.get("plots", {}),
        "best_model": report.get("best_model"),
        "deployment_report_model":
report.get("deployment_report_model"),
    }

    model.save(path)

return {
    "message": (
        "Продвинутая ML-модель рекомендаций обучена; "
        "исследовательские графики сохранены."
    ),
    "model_path": str(path),
    "dataset_type": "advanced_synthetic_weak_supervision",
    "artifacts_dir": str(ARTIFACTS_DIR),
    "deployment_model": (
        torch_result.get("model_type")

```

```

        if torch_result.get("trained")
        else model.metadata.get("model_type")
    ),
    "torch": torch_result,
    "report": {
        "summary_path": report.get("summary_path"),
        "plots": report.get("plots", {}),
        "best_model": report.get("best_model"),
        "deployment_report_model":
report.get("deployment_report_model"),
        "comparison": report.get("comparison", []),
    },
    **metrics,
}

def ensure_model_exists(
    samples: int = 900,
    path: Path = DEFAULT_MODEL_PATH,
) -> None:
    if not path.exists():
        train_and_save_model(
            samples=samples,
            path=path,
            create_report=False,
        )
    return

try:
    model = BasketballRecommendationModel.load(path)
except Exception:
    train_and_save_model(
        samples=samples,
        path=path,
        create_report=False,
    )
    return

if not model.is_compatible():
    train_and_save_model(
        samples=samples,
        path=path,
        create_report=False,
    )

if __name__ == "__main__":
    print(
        json.dumps(
            train_and_save_model(),
            ensure_ascii=False,
            indent=2,

```

```
)  
)
```

Файл « routes_ml.py»

```
from __future__ import annotations  
  
import json  
  
from fastapi import APIRouter  
  
from ml.recommender import HybridRecommender  
from ml.training_report import ARTIFACTS_DIR  
from ml.train_model import train_and_save_model  
from server.schemas.player import MLPredictRequest,  
RecommendationResponse  
  
router = APIRouter(prefix="/ml", tags=["ml"])  
  
PRESENTATION_PLOTS = {  
    "label_distribution",  
    "feature_importance",  
    "model_comparison",  
    "per_label_metrics",  
    "confusion_matrices",  
    "roc_curves",  
    "precision_recall_curves",  
    "sklearn_mlp_loss",  
    "torch_training_curves",  
    "torch_auxiliary_heads",  
}  
  
@router.post("/train")  
def train_model() -> dict:  
    return train_and_save_model(samples=6000)  
  
@router.get("/report")  
def training_report() -> dict:  
    summary_path = ARTIFACTS_DIR / "training_summary.json"  
  
    if not summary_path.exists():  
        return {  
            "available": False,  
            "message": (  
                "Отчет обучения пока не создан. "  
                "Запустите переобучение ML-модели."  
            ),  
            "summary_path": str(summary_path),  
            "plots": {},  
        }
```

```

    }

    summary = json.loads(
        summary_path.read_text(encoding="utf-8")
    )

    plots = {
        str(key): str(value)
        for key, value in (summary.get("plots") or {}).items()
        if key in PRESENTATION_PLOTS
    }

    return {
        "available": True,
        "summary_path": str(summary_path),
        "best_model": summary.get("best_model"),
        "deployment_report_model":
summary.get("deployment_report_model"),
        "dataset": summary.get("dataset", {}),
        "comparison": summary.get("comparison", []),
        "plots": plots,
    }

@router.post("/predict", response_model=RecommendationResponse)
def predict(payload: MLPredictRequest) -> dict:
    recommender = HybridRecommender()

    return recommender.recommend_from_payload(payload)

```

ПРИЛОЖЕНИЕ Б

(обязательное)

Интеллектуальная система оценки эффективности спортсмена в
игровых видах спорта

Презентация

Интеллектуальная
система оценки
эффективности
спортсмена в
игровых видах
спорта



«Цель состоит в том, чтобы
превратить данные в информацию,
а информацию — в понимание»

—Carly Fiorina, ex-CEO HP

Введение

В условиях активного развития цифровых технологий и методов искусственного интеллекта всё большее распространение получают интеллектуальные системы анализа данных.

Одним из перспективных направлений их использования является спортивная аналитика

Особую актуальность интеллектуальные системы приобретают в игровых видах спорта, где эффективность спортсмена определяется не одним отдельным показателем, а совокупностью различных статистических характеристик.

Использование машинного обучения в спортивной аналитике

В спортивной аналитике машинное обучение используют там, где обычного просмотра статистики уже недостаточно.

В зависимости от задачи могут применяться разные подходы.



Анализ аналогов



Second Spectrum(NBA)

Система спортивной аналитики, применяющая компьютерное зрение и методы искусственного интеллекта для анализа игровых событий



Stats Perform

Платформа спортивной статистики, используемая клубами, спортивными организациями и медиа



Catapult Sports

Система, использующая носимые датчики для отслеживания физической нагрузки спортсмена

Постановка задачи



Целью данной работы является разработка интеллектуальной программной системы RateMyPlayer, обеспечивающей сбор, хранение, обработку и анализ баскетбольных статистических данных, расчет эффективности игрока, визуализацию результатов и формирование рекомендаций с использованием методов машинного обучения.



Источники данных и технологии получения доступа к ним



ESPN API

Основной источник. Позволяет получать сведения о спортивных лигах, командах, игроках и их статистике.



CSV-файлы

Возможность загрузки заранее подготовленных наборов данных.



Стек технологий



Python
Язык разработки



PyCharm
IDE



Pytorch
ML-модуль



FastAPI
Серверная часть



Uvicorn
Запуск сервера

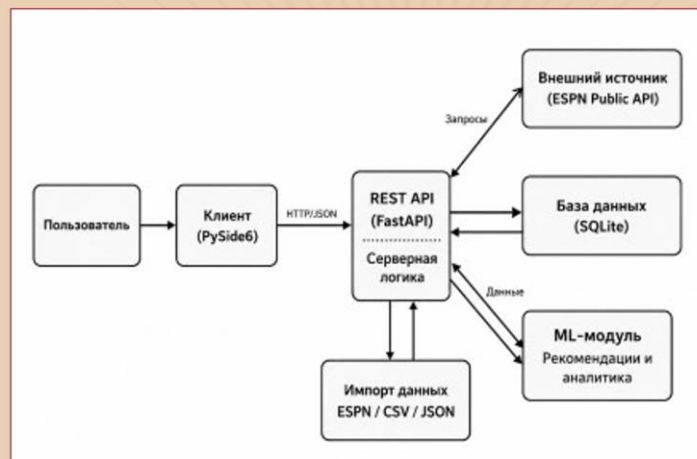


Pyside
Клиентский интерфейс

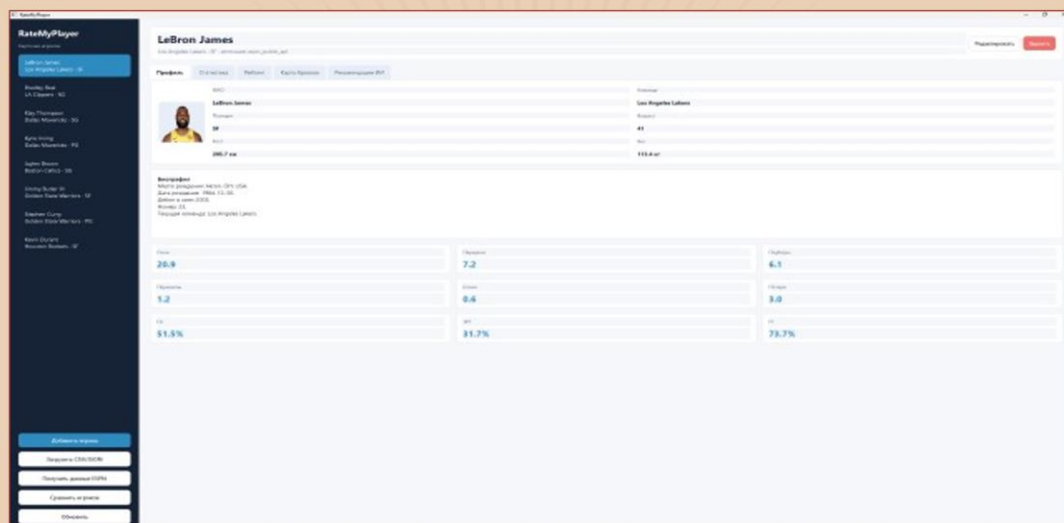


SQLite
Система хранения данных

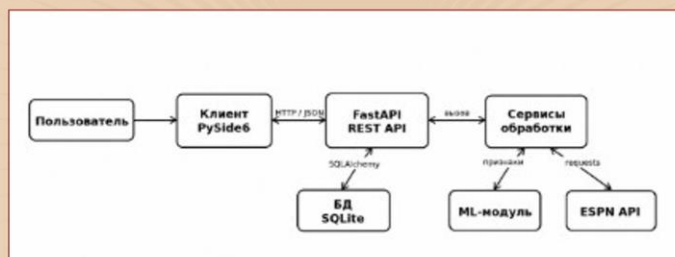
Общая архитектура системы



Разработка UI



Разработка клиент-серверной части



Разработка базы данных



Разработка ML-модуля



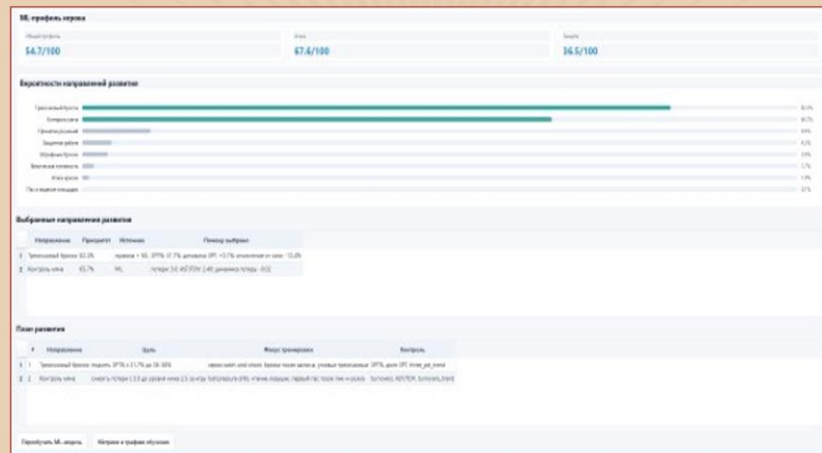
Разработка модуля формирования рекомендаций





144

Проверка работы ML-модуля и формирования рекомендаций

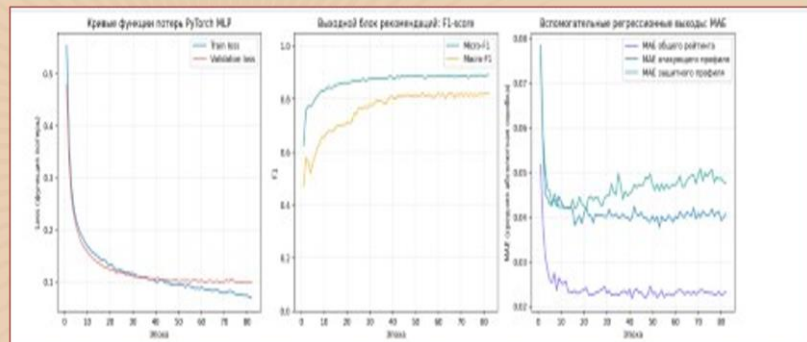


Проверка работы ML-модуля и формирования рекомендаций

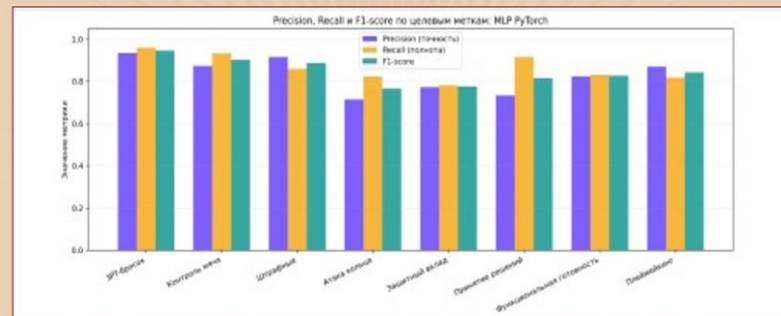
Метрики и графики обучения ML

Отчет обучения модели

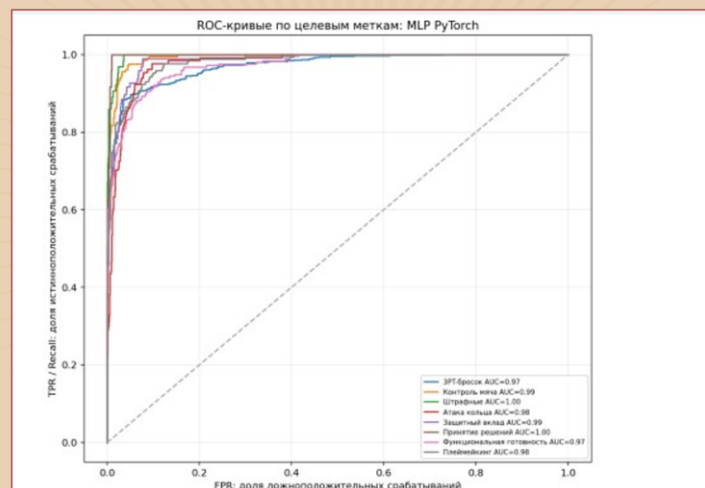
Лучшая исследовательская модель: Gradient Boosting
 Модель для диагностических графиков: PyTorch MLP
 Обучающих примеров: 4680
 Валидационных примеров: 1320
 Признаки: 31, метки: 8



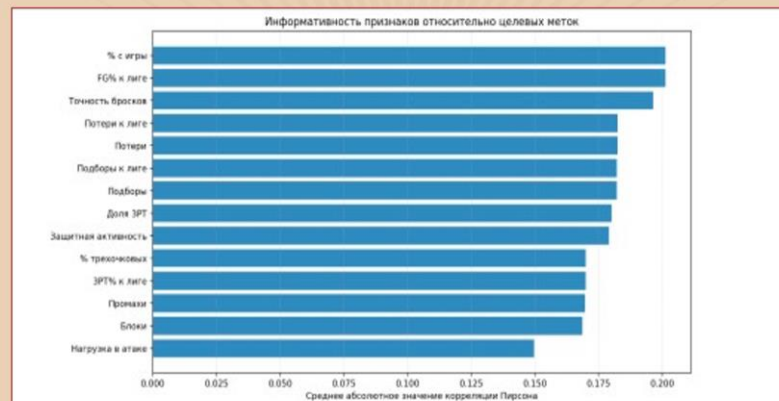
Проверка работы ML-модуля и формирования рекомендаций



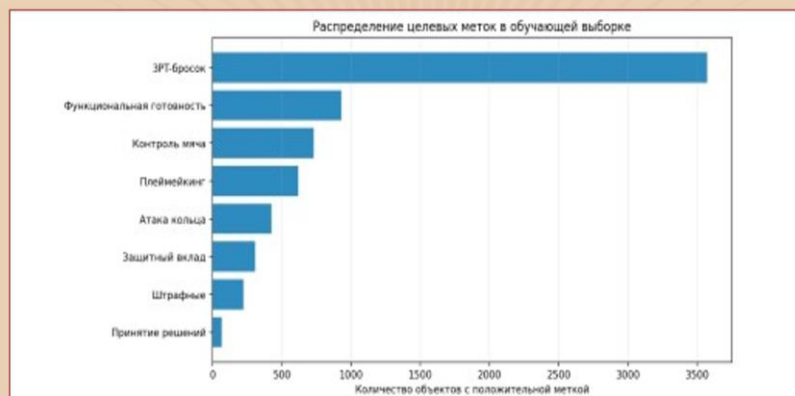
Проверка работы ML-модуля и формирования рекомендаций



Проверка работы ML-модуля и формирования рекомендаций



Проверка работы ML-модуля и формирования рекомендаций



Анализ результатов работы системы

Результаты тестирования подтвердили корректную работу основных аналитических функций.

В целом разработанная система выполняет задачи, поставленные в работе: хранит баскетбольные данные, обрабатывает их на серверной стороне, отображает результаты анализа и формирует рекомендации по игровому профилю баскетболиста.

Заключение

Практическая значимость RateMyPlayer заключается в том, что система помогает анализировать индивидуальные показатели баскетболистов, находить сильные стороны и зоны роста, сравнивать игроков и выбирать направления для дальнейшей тренировки

Дальнейшее развитие системы может быть связано с расширением набора спортивных данных, увеличением обучающей выборки, улучшением модели рекомендаций, добавлением других игровых видов спорта и доработкой интерфейса.





Спасибо за внимание!

Готов ответить на ваши
вопросы!

