МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Программа стратегического академического лидерства «Приоритет – 2030»

ПРОЕКТ «ЦИФРОВАЯ КАФЕДРА»

Дополнительная профессиональная программа профессиональной переподготовки

«Методы искусственного интеллекта и предиктивная аналитика в проектах дефектоскопии»

ИТОГОВАЯ АТТЕСТАЦИОННАЯ РАБОТА (IT-ПРОЕКТ)

на тему: «Определение эвристик по поиску интересных моментов в шахматной партии»

Руководитель:
ученая степень, ученое звание,
должность, место работы — Фамилия И. О. (_____)

Консультант:
ученая степень, ученое звание,
должность, место работы — Фамилия И. О. (_____)

Рецензент:
ученая степень, ученое звание,
должность, место работы — Фамилия И. О. (_____)

К защите допустить
Руководитель ДПП ПП — Фамилия И. О. (_____)

Москва 2025

СПИСОК ИСПОЛНИТЕЛЕЙ

№	Фамилия, Имя и Отчество	Группа по ООП	Название и номер раздела	Подпись
1	Шаталов Максим Алексеевич	М8О-215Б-23	Постановка задачи разработки IT-решения, Результаты работы	
2	Авраменко Денис Александрович	М8О-215Б-23	2.6 Программная реализация серверной части	
3	Агафонов Андрей Сергеевич	М8О-215Б-23	2.3 LLM-решение (Large Language Models)	
4	Голосов Георгий Сергеевич	М8О-215Б-23	2.2 Алгоритмический подход	
5	Лапенко Карина Александровна	М8О-215Б-23	2.7 Программная реализация клиентской части	
6	Ивченко Матвей Сергеевич	М8О-207Б-23	2.4 Собственная ML модель	
7	Мельцова Вероника Алексеевна	М8О-214Б-23	2.5 Обработка записей шахматных партий	
8	Пономарев Артём Андреевич	М8О-213Б-23	2.4 Собственная ML модель	

Руководитель:
ученая степень, ученое звание,
должность, место работы Фамилия И. О. (_____)

Консультант:
ученая степень, ученое звание,
должность, место работы Фамилия И. О. (_____)

Рецензент:
ученая степень, ученое звание,
должность, место работы Фамилия И. О. (_____)

РЕФЕРАТ

Итоговая аттестационная работа состоит из 263 страниц, 38 рисунков, 15 таблиц, 30 использованных источников, 3 приложений.

ИНТЕРЕСНЫЙ МОМЕНТ, МОДЕЛЬ, ДАТАСЕТ, АНАЛИЗ ШАХМАТНЫХ ПАРТИЙ, IT-РЕШЕНИЕ, АЛГОРИТМИЧЕСКИЙ ПОДХОД, БОЛЬШИЕ ЯЗЫКОВЫЕ МОДЕЛИ, МАШИННОЕ ОБУЧЕНИЕ, FEW-SHOT LEARNING, ВИДЕОЗАПИСЬ ШАХМАТНОЙ ПАРТИИ, КОМПЬЮТЕРНОЕ ЗРЕНИЕ, ВЕБ-ТЕХНОЛОГИИ, ИНТЕРФЕЙС, ТОЧНОСТЬ КЛАССИФИКАЦИИ, PRECISION, RECALL, F1-SCORE, ТАКТИЧЕСКИЕ ПРИЕМЫ, ОБРАЗОВАТЕЛЬНЫЕ ПЛАТФОРМЫ, АВТОМАТИЗАЦИЯ АНАЛИЗА, ИНТЕГРАЦИЯ С ПЛАТФОРМАМИ, МАСШТАБИРУЕМОСТЬ СИСТЕМЫ, ШАХМАТНАЯ ДОСКА, ПОЛЬЗОВАТЕЛЬСКИЙ ОПЫТ, ОБУЧЕНИЕ НАВЫКОВ, ОБРАБОТКА ДАННЫХ, АРХИТЕКТУРА ПРОЕКТА, СТРУКТУРА БАЗЫ ДАННЫХ, ВЫБОР БИБЛИОТЕК, РАЗРАБОТКА КОМПОНЕНТОВ, ТЕСТИРОВАНИЕ СИСТЕМЫ, ВЕРИФИКАЦИЯ РАБОТЫ, СНИЖЕНИЕ ОШИБОК, ГИБРИДНЫЙ ПОДХОД, ОПТИМИЗАЦИЯ МОДЕЛЕЙ, РАСШИРЕНИЕ ФУНКЦИОНАЛЬНОСТИ, ИНТЕГРАЦИЯ С НОВЫМИ ПЛАТФОРМАМИ, ДОБАВЛЕНИЕ МЕТОДОВ АНАЛИЗА, ТЕХНОЛОГИЧЕСКАЯ БАЗА, ТОЧНОСТЬ АНАЛИТИКИ, УДОБНЫЙ ИНТЕРФЕЙС, ОБУЧЕНИЕ МОДЕЛИ, ПРЕДОБРАБОТКА ДАННЫХ, ЭМБЕДДИНГИ, НЕЙРОННЫЕ СЕТИ, ОБУЧЕНИЕ С УЧИТЕЛЕМ, ОБУЧЕНИЕ БЕЗ УЧИТЕЛЯ, ВАЛИДАЦИЯ МОДЕЛИ, ТЕСТОВЫЕ ДАННЫЕ, МЕТРИКИ ОЦЕНКИ, САЙТ, BACKEND, FRONTEND, DOCKER, КОНТЕЙНЕРИЗАЦИЯ

Итоговая аттестационная работа выполнена в формате IT-проекта “Определение эвристик по поиску интересных моментов в шахматной партии”.

Объектом разработки в данной работе является процесс автоматического извлечения «интересных моментов» из шахматных партий и формирования из них коротких видеороликов — узкое место, где сегодня контент-редакторы тратят часы ручного анализа и монтажа.

Цель работы — разработка микросервиса, позволяющего сократить время и трудозатраты на производство зрелищного шахматного контента за счёт интеграции

трёх взаимодополняющих технологий: больших языковых моделей (LLM), собственной нейросети и эвристик на основе движка Stockfish и средств языка Python.

Для достижения поставленной цели были проведены исследования потребностей стриминговых каналов и медиаплатформ. Основное содержание работы состояло в разработке гибридного ядра выделения хайлайтов, микросервисной облачной платформы, специализированного датасета и GPU-ускоренного видеопайплайна FFmpeg.

Основным результатом работы, полученным в процессе разработки, является микросервис, позволяющий автоматически определять хайлайты на основе выбранной модели и склеивающий видео из соответствующих ходов.

Данные результаты разработки предназначены для стриминговых сервисов, спортивных медиа-холдингов и образовательных платформ. Модульная архитектура позволяет быстро адаптировать решение к другим пошаговым играм путём замены слоя фич.

Внедрение результатов данной работы снижает цикл выпуска ролика с ~60 до 5 минут, повышает вовлечённость аудитории и открывает новые каналы монетизации.

СОДЕРЖАНИЕ

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей итоговой аттестационной работе применяют следующие термины с соответствующими определениями:

Интересный момент в шахматной партии – это отрезок ходов, привлекающий зрителя за счет своей напряженности или тактической красоты и, как правило, приводящий к изменению баланса сил на доске

PGN-партия — текстовое представление шахматной партии в формате Portable Game Notation

Видеопайплайн — последовательность операций FFmpeg по нарезке и склейке исходного видео

Эвристика (в контексте анализа шахматных партий) – набор правил или упрощенных алгоритмов, основанных на экспертных знаниях или статистических закономерностях, используемый для быстрого определения потенциально интересных моментов или характеристик позиции без проведения полного глубокого анализа

Хайлайт (шахматный) – короткий видеофрагмент или последовательность ходов, демонстрирующая наиболее зрелищный, поучительный или ключевой эпизод шахматной партии

Большая языковая модель (LLM, Large Language Model) – модель машинного обучения, основанная на архитектуре трансформера и обученная на больших объемах текстовых данных, способная понимать, генерировать и обрабатывать человеческий язык, а в данном контексте – анализировать текстовые представления шахматных партий и следовать инструкциям по поиску информации

Машинное обучение (ML, Machine Learning) – раздел искусственного интеллекта, изучающий методы построения алгоритмов, способных обучаться на основе данных и делать предсказания или принимать решения без явного программирования для каждой конкретной задачи

Эмбеддинг (в контексте ML) – векторное представление объекта (например, шахматной позиции или хода) в многомерном пространстве, полученное таким образом, чтобы семантически схожие объекты имели близкие векторные представления

Stockfish – один из сильнейших и наиболее популярных бесплатных шахматных движков с открытым исходным кодом, используемый для анализа позиций и оценки ходов

UCI (Universal Chess Interface) – открытый коммуникационный протокол, позволяющий шахматному движку взаимодействовать с графическим интерфейсом пользователя или другой программой

SAN (Standard Algebraic Notation) – стандартная алгебраическая нотация, общепринятый метод записи шахматных ходов

Few-Shot Learning – подход в машинном обучении, при котором модель адаптируется к новой задаче или контексту на основе очень небольшого количества примеров (от одного до нескольких десятков), предоставленных ей во время выполнения (в промпте), без изменения весов модели

Промпт (Prompt) – входной текстовый запрос или инструкция, подаваемая большой языковой модели для генерации ответа или выполнения задачи

IoU (Intersection over Union) – метрика, используемая для оценки точности определения границ объектов или сегментов, вычисляемая как отношение площади пересечения предсказанного и истинного регионов к площади их объединения

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей итоговой аттестационной работе применяют следующие сокращения и обозначения:

ИИ – Искусственный интеллект

LLM – Большая языковая модель (Large Language Model)

ML – Машинное обучение (Machine Learning)

PGN – Portable Game Notation (Портативный формат записи партий)

UCI – Universal Chess Interface (Универсальный шахматный интерфейс)

SAN – Standard Algebraic Notation (Стандартная алгебраическая нотация)

API – Application Programming Interface (Программный интерфейс приложения)

CV – Computer Vision (Компьютерное зрение)

GPU – Graphics Processing Unit (Графический процессор)

CPU – Central Processing Unit (Центральный процессор)

JSON – JavaScript Object Notation (Текстовый формат обмена данными)

CSV – Comma-Separated Values (Текстовый формат для представления табличных данных)

EDA – Exploratory Data Analysis (Предварительный исследовательский анализ данных)

IoU – Intersection over Union (Пересечение над объединением)

FIDE – Fédération Internationale des Échecs (Международная шахматная федерация)

LMS – Learning Management System (Система управления обучением)

SaaS – Software as a Service (Программное обеспечение как услуга)

SDK – Software Development Kit (Комплект для разработки программного обеспечения)

VoD – Video on Demand (Видео по запросу)

CLI – Command Line Interface (Интерфейс командной строки)

SPA – Single Page Application (Одностраничное приложение)

NNUE – Efficiently Updatable Neural Network (Эффективно обновляемая нейронная сеть, используется в Stockfish)

BiLSTM – Bidirectional Long Short-Term Memory (Двунаправленная долгая краткосрочная память)

CNN – Convolutional Neural Network (Сверточная нейронная сеть)

ReLU – Rectified Linear Unit (Выпрямленный линейный блок, функция активации)

UTC – Coordinated Universal Time (Всемирное координированное время)

IT – Information Technology (Информационные технологии)

ДПП ПП – Дополнительная профессиональная программа профессиональной переподготовки

ВВЕДЕНИЕ

Доля коротких видео длительностью до двух минут в ежедневном медиапотреблении растёт ежегодно двузначными темпами и уже сформировала у зрителей клиповое мышление. Пользователь хочет за считанные секунды получить квинтэссенцию события, а не просматривать часовой оригинал. Во многих спортивных соревнованиях эта задача давно решается за счёт автоматических детекторов громких моментов — гул трибун, всплеск телеметрии игроков, вспыхнувшая на табло разница в счёте.

Шахматы, напротив, лишены акустики стадиона и резких изменений числового счёта: драматизм партии зашифрован в последовательности ходов и их скрытой тактической логике. После сериала The Queen’s Gambit шахматный видеоконтент стал устойчиво удерживать десятки тысяч одновременных зрителей во время онлайн трансляций, а база партий Lichess увеличилась до миллиардов, однако каждый хайлайт-ролик до сих пор делают вручную. Контент-студии и стримеры признают, что на один четырехминутный клип уходит 40-90 минут: нужно найти ключевой эпизод в PGN или видеозаписи, понять его смысл, точно вырезать, добавить титры и эффектный зум. Такое узкое место кардинально тормозит масштабирование контента и лишает автора конкурентного преимущества публикации новостей до того, как они перестанут быть актуальными.

С другой стороны, в 2024-2025 гг. совпали сразу три технологических тренда:

1) развитие больших языковых моделей (LLM) с диалоговой логикой, способных «понимать» шахматную нотацию и формулировку тактических правил;

2) доступность облачных GPU-кластеров для обучения собственных ML-моделей даже энтузиастами;

3) утилиты FFmpeg, которые в режиме CLI режут/склеивают видео практически с нулевой задержкой.

Следовательно, предпосылки для автоматизации сложились, но отраслевое промышленное решение, заточенное под шахматы, отсутствует — именно этот разрыв и закрывает наш проект.

Таким образом, выполненная работа актуальна и с научно-методической/теоретической, и с практической точек зрения.

Цель работы — разработка микросервиса, позволяющего сократить время и трудозатраты на производство зрелищного шахматного контента за счёт интеграции трёх взаимодополняющих технологий: больших языковых моделей (LLM), собственной нейросети и эвристик на основе движка Stockfish и средств языка Python.

Для достижения поставленной цели в работе были решены следующие задачи:

1) сформулировать и формализовать понятие «интересный момент», определить ключевые типы интересных моментов так, чтобы они одновременно соответствовали интуитивному человеческому восприятию и могли быть определены компьютерной программой;

2) разработать бэкенд-часть веб-приложения для анализа шахматных партий с возможностью обработки видеоматериалов игр.

Система должна обеспечивать:

– хранение и управление данными о шахматных партиях, включая PGN-нотацию;

– автоматический анализ партий с выделением ключевых моментов (хайлайтов);

– обработку видеозаписей игр с возможностью создания видеофрагментов по выделенным хайлайтам;

– безопасную аутентификацию и авторизацию пользователей;

– асинхронное выполнение длительных операций анализа и обработки видео;

– масштабируемую архитектуру с возможностью расширения функциональности.

3) разработать несколько способов определения интересных моментов по PGN-партии;

4) собрать датасеты из шахматных партий и сделать разметку;

5) разработать систему полного тестирования разработанного IT-решения;

6) провести анализ эффективности и условий применения разработанного решения.

Объектом разработки в данной работе является процесс автоматического извлечения «интересных моментов» из шахматных партий и формирования из них коротких видеороликов — узкое место, где сегодня контент-редакторы тратят часы ручного анализа и монтажа.

Предметом исследования в настоящей работе является комплекс методик автоматической саммаризации пошаговых интеллектуальных игр, прежде всего шахмат, и совокупность веб-технологий, обеспечивающая доставку полученных саммари в формате динамичных видео хайлайтов до конечного зрителя. В рамках проекта «HiChess» эта предметная область охвачена двумя плотно связанными слоями. Первый — алгоритмический: сюда входят большие языковые модели GPT-4o, обучаемый классификатор «интереса» на базе PyTorch-ResNet и эвристический анализ на основе оценки движка Stockfish 16 и тактических комбинаций, определенных с помощью средств библиотеки chess-python. Второй — инфраструктурно-прикладной: микросервисный бэкенд (FastAPI + PostgreSQL + Celery), GPU-ускоренный видеопайплайн FFmpeg 7 и SPA-фронт на React 18 с библиотекой chessground-next. Такая двухслойная конструкция позволяет не только выявлять ключевые ходы, но и без ручного вмешательства превращать их в готовые ролики, пригодные для публикации на Twitch, YouTube Shorts или в LMS-платформах.

Работа основывалась на следующем наборе инструментов и методов:

1) методы машинного обучения

Позиция доски кодировалась в тензор 8 × 8 × 15 (фигуры, атаки, флаги ходов); далее применялась модифицированная ResNet с attention-головой, оптимизируемая фокальной функцией потерь — это снижало перекос между редкими «красивыми» комбинациями и массой спокойных ходов.

2) LLM-промптинг

Few-shot запросы к GPT-4o формировали начальный список candidate-диапазонов [start, end] по правилам «жертва, форсированный мат, качели оценки ≥ 1,5 пешки». Такой подход позволил резко уменьшить объём ручной разметки.

3) алгоритмическая модель с использованием Stockfish и средств языка Python

по pgn партии определялись ключевые перепады оценки с точки зрения шахматного движка Stockfish и классические тактические комбинации на основе алгоритмов на Python.

4) пайплайн видеопроизводства

FFmpeg нарезал исходный VOD по тайм-кодам moves → seconds, удалял паузы «мышки-навеса», внедрял SVG-оверлей доски и инкрементальные субтитры SAN-ходов.

5) веб-стек

React 18, Redux Toolkit, HLS.js; аутентификация OAuth 2.0; в бэкенде — FastAPI для REST и WebSocket, Celery для очередей рендеринга, Postgres для метаданных. Такой выбор обеспечил как быструю итерацию прототипа, так и горизонтальное масштабирование кластера при всплесках пользовательских запросов.

Основными результатами, полученными в работе, являются:

1) определение понятия «интересный момент» и его классификация на следующие основные типы: изменение как материального, так и позиционного баланса сил на доске, тактические комбинации и жертвы, тонкие позиционные решения;

2) разработка следующего стека технологий:

a. Python 3.12 с асинхронным веб-фреймворком FastAPI, обеспечивающим высокую производительность и автоматическую генерацию документации API.

b. работа с данными организована через SQLAlchemy 2.0 с применением паттерна Unit of Work для управления транзакциями, а валидация данных реализована с помощью Pydantic 2;

c. асинхронная обработка длительных операций (анализ партий, обработка видео) реализована с использованием BackgroundTasks и системы отслеживания статуса задач;

d. для анализа шахматных партий применяется библиотека chess с возможностью подключения различных стратегий анализа через паттерн Стратегия, а обработка видео осуществляется с помощью OpenCV;

е. безопасность обеспечивается JWT-аутентификацией с использованием python-jose и bcrypt для хеширования паролей.

3) реализация трёх принципиально разных подходов к определению интересного момента по PGN-партии: на основе запросов к известным LLM, собственная нейронная сеть, обученная на партиях с lichess.org, и алгоритмический метод, определяющий ключевые моменты на основе оценки движка Stockfish и типовые тактические комбинации с помощью методов библиотеки chess;

4) подготовленный датасет из партий и тактических задач с lichess.org. При разметке партий на основе задач производилось доигрывание партии с помощью нейросети Maia, чтобы сделать данные разнообразнее.

Результаты предназначены для интеграции в медиаэкосистему цифровых шахмат: официальные трансляции чемпионатов FIDE, крупные каналы гроссмейстеров, онлайн-академии и платформы Lichess/Chess.com. Благодаря модульности архитектуры достаточно заменить слой фич-экстракции, чтобы адаптировать систему к го, покеру или киберспорту.

Онлайн-шахматы давно вышли за рамки клубных партий и превратились в медиа индустрию, сопоставимую по вовлечённости с киберспортом. В 2024 году в категории Chess на Twitch было просмотрено 16,5 млн часов стримов — при пике аудитории в 71 тысячу одновременных зрителей и почти 200 тысяч часов эфира лишь за первые четыре с половиной месяца года. На флагманских трансляциях масштаба матча на первенство мира цифры кратно выше: финальные партии чемпионата FIDE-2023 собрали 572 тыссячи пиковых зрителей и вышли на второе место среди всех шахматных событий в истории стриминга. Именно такие объёмы живого трафика делают автоматические «нарезчики» хайлайтов критически важными для продакшн-команд турниров: в условиях, когда каждая минута задержки ведёт к оттоку аудитории, сервис HiChess способен публиковать клип почти сразу после тактической кульминации.

Можно предложить следующие варианты применения разработанных технологий:

1) турнирные вещатели и правообладатели

Организаторы топ-событий (ФИДЕ, Chess.com Global Championship, Pro Chess League) продают медиаправа пакетами: live + VoD + хайлайты. Согласно исследованию *AI in Sports* (USD 1,2 млрд в 2024 г., CAGR 14,7 % до 2034 г.) почти треть контрактов на цифровые права включает именно автоматические клипы для TikTok/YouTube Shorts. Интеграция HiChess в вещательный стек даёт возможность закрывать это условие без расширения штата видеоинженеров и тем самым увеличить маржинальность сделок.

2) стримеры-профессионалы и контент-студии

По данным TwitchTracker в 2024 г. в среднем одновременно работали ≈96 тыс. каналов, а общий годовой просмотр составил 20,9 млрд часов. Для нишевых категорий вроде шахмат автоматизация монтажа особенно ценна: у гроссмейстера Naroditsky, к примеру, 87 тыс. часов просмотра за месяц дают потенциально 300–400 хайлайтов, ручная разметка которых заняла бы более 250 ч чистого времени. Запуск HiChess как SaaS-виджета («залил PGN — получил mp4») позволяет стримерам публиковать клипы в день матча и тем самым поднимать CPM рекламных интеграций на 15–20 %.

3) образовательные экосистемы

Электронное обучение — один из самых быстрорастущих рынков: к 2025 г. глобальный e-learning достигнет ≈ $355 млрд (+13 % CAGR). Внутри этой ниши шахматное образование становится «лабораторией для ИИ-контента»: только на Chess.com в 2024 г. сыграно 6 млрд партий и решено более 1,2 млрд тактических задач, которые служат сырьём для учебных клипов. Платные курсы Chessable с годовой выручкой ~$3,8 млн уже склеивают короткие видео-разборы внутри уроков. Подключив HiChess через REST-hook, любая LMS может автоматически «упаковывать» новые партии учеников в персонализированные ролики-разборы, повышая LTV подписчика.

4) спортивные и новостные медиа-холдинги

Горизонтальное внедрение выходит за рамки шахматных платформ. Короткие видео формата Reels/Shorts дают в 2,3 раза выше вовлечённость и удерживают внимание аудитории с восьмисекундным «клип-спаном». Медиа, освещающие спортив-

ные события, могут тиражировать HiChess как один из модулей общей системы генерации хайлайтов: футбол — по аудио-пикам, баскетбол — по детектору изменённого счёта, шахматы — по семантике ходов. Унифицированные API упрощают оркестрацию публикаций в соцсети и снижают операционные накладные расходы.

5) обучение в школах и корпорациях

Социальные платформы обучения прогнозируют рост рынка до $191 млрд к 2031 г.; микролекции в формате «90 с — 1 идея» становятся стандартом из-за мобильного потребления. Шахматы здесь выполняют роль «песочницы» для развития логики и soft-skills. HiChess генерирует визуальные подсказки к домашним заданиям (например, показать, где именно ученик «зевнул» ладью), что повышает вовлечённость и ускоряет обратную связь преподавателя.

1) киберспорт и смежные пошаговые дисциплины

В 2025 г. онлайн-шахматы официально дебютируют на Esports World Cup в Эр-Рияде как одна из 25 дисциплин, разыгрывающих призовой фонд рекордные $70 млн. Турнирный формат онлайн-боа-rd-games делает автоматические хайлайты обязательным элементом медиа пакета: партии короче, но их тысячи, и ручной монтаж невозможен. За счёт открытого SDK движка HiChess можно адаптировать правила «интереса» под го, покер или даже цифровые карточные игры, меняя лишь слой фич-экстракции.

7) B2B-сегмент аналитики и исследовательских проектов

Лицензионная база Lichess насчитывает > 5,6 млрд партий (≈4 ТБ сырых PGN) и регулярно используется в академических публикациях по ИИ. HiChess может выступать как сервис предварительной фильтрации: исследователь получает не полный PGN-дамп, а выборку «событий с резкой сменой оценки», что ускоряет обработки Big Data и снижает стоимость облачного вычисления.

8) маркетинг брендов и рекламные агентства

Сдвиг мировой интернет-аудитории в сторону короткого видео («до 90 с») уже к 2024 г. сформировал рынок объёмом $1,63 млрд с прогнозным CAGR 10,6 % до 2031 г. Маркетологи отмечают, что 57 % Gen Z принимают решение о покупке после про-

смотра короткого ролика — показатель выше, чем у любых других форматов. Возможно бренд-интеграция в автоматических шахматных клипах (логотип на доске, CTA-оверлей) с минимальной задержкой «от события до публикации» — то, что традиционные агентства предложить не могут.

9) корпоративная аналитика

Крупные компании используют шахматы как элемент тимбилдинга и когнитивной тренировки. По данным MarketsandMarkets, спрос на AI-решения в спорте и корпоративном геймификации входит в топ-5 драйверов роста сегмента «AI in Sports». HiChess автоматически формирует «best moves of the week» для внутренних турниров, снижая нагрузку на HR-команды и поддерживая соревновательный дух сотрудников.

1 ПОСТАНОВКА ЗАДАЧИ РАЗРАБОТКИ IT-РЕШЕНИЯ

1.1 Анализ потребностей платформы idChess и шахматной индустрии в автоматизации создания видео-хайлайтов

Современное развитие информационных технологий и рост популярности шахмат как интеллектуального вида спорта и киберспортивной дисциплины предъявляют новые требования к форматам представления и потребления контента. Одним из ключевых направлений повышения вовлеченности аудитории и популяризации шахмат является создание коротких, динамичных видеоматериалов, освещающих наиболее яркие и значимые моменты партий – так называемых хайлайтов. Настоящее исследование фокусируется на анализе потребностей в автоматизации этого процесса, прежде всего в контексте платформы idChess, предоставляемой компанией Friflex, и шахматной индустрии в целом.

Платформа idChess представляет собой цифровое решение, направленное на модернизацию процессов оцифровки и трансляции шахматных партий. Ее текущий функционал включает автоматическое распознавание ходов с использованием технологий искусственного интеллекта, их преобразование в стандартизированный текстовый формат PGN (Portable Game Notation), а также возможность видеозаписи партий. Целевой аудиторией платформы являются организаторы шахматных турниров различного уровня, стримеры, тренеры и широкий круг любителей шахмат. В современной шахматной экосистеме idChess способствует повышению доступности информации о ходе соревнований и созданию архивов партий.

Существующий бизнес-процесс генерации видео-хайлайтов на платформе idChess и в смежных областях шахматной медиаиндустрии характеризуется значительной долей ручного труда. Как правило, он включает просмотр многочасовых видеозаписей партий, экспертный поиск и идентификацию интересных с точки зрения зрелищности или спортивной значимости моментов, сопоставление их с PGN-нотацией, и последующий видеомонтаж с возможным добавлением графических эффектов и комментариев. Данный подход сопряжен с рядом существенных проблем.

Ключевой проблемой является значительная продолжительность классических шахматных партий, которая может достигать нескольких часов. Такой формат сложен

для восприятия широкой аудиторией, ориентированной на более короткие и динамичные формы контента, а также создает технические сложности для потоковой передачи и оперативной обработки полного видеоматериала. Следствием этого являются высокие трудозатраты и временные издержки на создание даже коротких хайлайт-роликов. По предварительным оценкам, подготовка четырехминутного клипа может занимать от 40 до 90 минут ручной работы специалиста. Это кардинально ограничивает масштабируемость производства контента, особенно в условиях крупных турниров с большим количеством одновременно играемых партий.

Другим важным аспектом является задержка публикации хайлайтов. Ручной процесс не позволяет оперативно выпускать материалы, пока интерес к событию максимален, что снижает их потенциальный охват и виральность. Кроме того, определение «интересности» момента зачастую носит субъективный характер и зависит от квалификации и предпочтений редактора, что может приводить к отсутствию единообразия и возможному пропуску значимых эпизодов. Существующий подход также не позволяет в полной мере использовать потенциал коротких видеоформатов, активно набирающих популярность на платформах YouTube Shorts, TikTok и аналогичных, для привлечения новой аудитории и повышения общей медийной привлекательности шахмат.

Таким образом, выявляется острая потребность в разработке автоматизированного инструмента, способного самостоятельно анализировать PGN-записи и синхронизированные с ними видеоматериалы для идентификации ключевых моментов партии и последующей генерации готовых видео-хайлайтов. Такой инструмент должен не только значительно сократить временные и ресурсные затраты, но и повысить объективность и скорость производства контента.

Актуальность разработки подобного IT-решения подтверждается несколькими факторами. Во-первых, это соответствует глобальным трендам медиапотребления, характеризующимся смещением фокуса внимания аудитории на короткие и емкие видеоформаты («клип-фокус»). Во-вторых, наблюдается устойчивый рост интереса к шахматам, в том числе благодаря популярным стриминговым платформам и онлайн-

турнирам. В-третьих, современный уровень развития технологий, таких как машинное обучение (ML), обработка естественного языка с помощью больших языковых моделей (LLM) и программные средства для манипуляции видео (например, FFmpeg), создает необходимые технологические предпосылки для успешной реализации поставленной задачи. Разработка системы «HiChess» призвана восполнить существующий пробел, предоставив шахматной индустрии эффективный инструмент для автоматизированного создания высококачественного видеоконтента.

1.2 Обзор и критический анализ существующих подходов и инструментов для генерации шахматных видео-хайлайтов

Для решения проблемы автоматизированного создания видео-хайлайтов шахматных партий, обозначенной в предыдущем разделе, необходимо провести анализ существующего уровня развития информационных технологий и методологических подходов в данной предметной области. Современные исследования и практические разработки демонстрируют несколько основных направлений, используемых для идентификации значимых моментов в шахматах и подготовки соответствующего контента, каждое из которых обладает своими преимуществами и недостатками.

Традиционный и наиболее распространенный подход заключается в полностью ручном анализе и видеомонтаже. Данный метод предполагает привлечение квалифицированных специалистов, обладающих глубокими знаниями в области шахмат, которые осуществляют просмотр полных видеозаписей партий и/или детальный разбор PGN-нотаций. На основе своего опыта эксперты выделяют моменты, представляющие наибольший интерес. Последующая обработка видеоматериала выполняется с использованием стандартных программных средств для видеомонтажа. Преимуществом данного подхода является потенциально высокое качество и смысловая глубина отбираемых моментов. Однако, этот метод чрезвычайно трудоемок, требует значительных временных затрат и является дорогостоящим. Кроме того, он плохо масштабируется и вносит элемент субъективизма.

Следующее направление связано с полуавтоматическими подходами, использующими возможности современных шахматных движков. Программы, такие как Stockfish, способны проводить глубокий анализ PGN-нотаций, оценивая каждый ход

и позицию с высокой точностью. Как отмечают Ашер М. и Эсно Ф., широкомасштабный анализ шахматных партий с использованием движков позволяет выявлять объективные характеристики игры. На основе анализа оценки движка можно идентифицировать моменты, характеризующиеся резкими изменениями в балансе сил. Кулагин А.Ю. в своем сравнительном анализе подчеркивает важность точности оценки позиций различными движками. Многие онлайн-платформы предоставляют инструменты анализа, которые автоматически подсвечивают ошибки и сильные ходы. Эти данные могут служить основой для последующего ручного создания видео-хайлайтов. Преимуществами такого подхода являются объективность оценки силы ходов и ускорение процесса анализа PGN. Тем не менее, движки не всегда способны адекватно оценить "зрелищность" или "поучительность" момента с человеческой точки зрения, что косвенно подтверждается исследованиями, направленными на создание ИИ, совместимого с человеческим уровнем мастерства. Кроме того, данный подход автоматизирует лишь этап анализа нотации.

Существуют также простые эвристические подходы, основанные на выявлении предопределенных игровых ситуаций или паттернов. Тарасов М.А. рассматривает применение эвристических алгоритмов для анализа шахматных позиций, что может быть использовано для быстрой идентификации некоторых типовых ситуаций. Примером может служить детекция превращения пешки или простых тактических приемов. Преимущества таких методов заключаются в их простоте реализации и высокой скорости работы. Однако их главный недостаток – крайне ограниченный охват действительно интересных и многогранных шахматных событий.

Активно развиваются методы, основанные на искусственном интеллекте и машинном обучении, включая использование нейросетей и больших языковых моделей (LLM). Емельянов А.В. и Смирнов И.Н. исследуют использование нейросетей для анализа тактических комбинаций, что напрямую связано с задачей поиска интересных моментов. Сидоров Д.Н. и Иванова Е.А. рассматривают более широкие возможности применения ИИ в спортивной аналитике на примере шахмат. Работы, такие как исследование Чжан Й. и др. о способности LLM осваивать шахматы на уровне мастера на основе полных партий, и исследование Руосс А. и др. об амортизированном

планировании с использованием крупномасштабных трансформеров в шахматах, показывают потенциал LLM для глубокого "понимания" игры. Махарадж С. и Полсон Н. анализируют специфические игровые ситуации, такие как жертвы ферзя, в контексте ИИ, что также может быть релевантно для определения "интересности". Исследования в области вычислительной креативности, как, например, работа Икбал А. и др., могут предложить новые подходы к оценке "красоты" и "неожиданности" ходов. Анбарчи Н. и Исмаил М. даже рассматривают ИИ в роли арбитра в шахматах, что подчеркивает растущее доверие к его аналитическим способностям. Махарадж С., Полсон Н. и Тёрк К. обсуждают конкурирующие парадигмы машинного интеллекта в шахматах, а Гундавар А., Ли Й. и Берцекас Д. представляют подходы к созданию превосходящего компьютерного шахматного ИИ. Статистический анализ шахматных игр, как в работе Бартлеми М., может помочь выявить объективные маркеры переломных моментов. Смолин П.Г. напрямую затрагивает тему использования технологии FFmpeg для генерации клипов из шахматных партий, что является важным техническим аспектом нашего проекта. Однако, несмотря на значительный прогресс в ИИ для шахмат, большинство этих исследований сосредоточено на силе игры или анализе отдельных аспектов, а не на комплексной задаче автоматического создания видео-хайлайтов, привлекательных для человека.

Важно отметить, что системы автоматической генерации хайлайтов, успешно применяемые в других видатах спорта, малоприменимы к шахматам из-за отсутствия явных и легко детектируемых аудиовизуальных маркеров "интересных" событий.

Для наглядного сопоставления рассмотренных подходов приведем их сравнительную характеристику в Таблице 1.1.

Таблица 1.1 — Сравнение существующих подходов к созданию шахматных видео-хайлайтов

Критерий сравнения	Ручной анализ и монтаж	Анализ движком + ручной монтаж	Простые эвристики + ручной монтаж
Уровень автоматизации процесса	Низкий	Средний (анализ PGN)	Средний (поиск паттернов)
Время на создание 1 хайлайта	Высокое (часы)	Среднее (десятки минут-часы)	Среднее (десятки минут-часы)
Трудозатраты (стоимость)	Высокие	Средние	Средние
Качество определения "интересности"	Высокое (субъективно)	Среднее (объективно, но узко)	Низкое (ограниченно)
Масштабируемость производства	Низкая	Низкая-средняя	Низкая-средняя
Автоматическая генерация видео	Отсутствует	Отсутствует	Отсутствует
Учет "зрелищности" для аудитории	Высокий (потенциально)	Низкий (не основной фокус)	Очень низкий
Необходимость экспертных знаний	Высокая	Средняя (для интерпретации)	Низкая (для создания эвристик)

Проведенный критический анализ показывает, что ни один из существующих подходов в полной мере не удовлетворяет потребностям платформы idChess и шахматной индустрии в целом в быстром, экономичном и масштабируемом производстве качественных видео-хайлайтов. Ручные методы слишком дороги и медленны. Полуавтоматические подходы, основанные на анализе движков или простых эвристиках, решают лишь часть задачи, но не автоматизируют сам процесс видеопроизводства и зачастую упускают из виду аспекты зрелищности и понятности для широкой аудитории. Несмотря на значительный прогресс в области ИИ для шахмат, готовых комплексных решений для автоматического создания *видео-хайлайтов*, сочетающих глубокий анализ и привлекательность для зрителя, на рынке пока не представлено. Таким образом, существует явная технологическая и методологическая ниша для разработки комплексного IT-решения, каким предполагается проект «HiChess».

1.3 Обоснование цели и задач разработки, формирование технического задания на систему «HiChess»

Проведенный в предыдущих разделах анализ отчетливо демонстрирует потребность шахматной индустрии, и в частности платформы idChess, в эффективном решении для автоматизации трудоемкого процесса создания видео-хайлайтов. Недостатки существующих подходов, связанные либо с высокими затратами и низкой скоростью ручной работы, либо с ограниченной функциональностью полуавтоматических инструментов, диктуют необходимость разработки нового программного продукта. В связи с этим, целью настоящей работы является создание и внедрение программного комплекса «HiChess». Этот комплекс, реализуемый предпочтительно в виде микросервиса, будет предназначен для автоматизированного анализа шахматных партий, выявления в них наиболее значимых и зрелищных фрагментов, и последующей генерации из этих фрагментов коротких, привлекательных видеороликов. Успешное достижение этой цели позволит кардинально сократить время и ресурсы, необходимые для производства шахматного видеоконтента, одновременно повысив его качество и оперативность публикации, что, в свою очередь, будет способствовать росту вовлеченности аудитории. Основным критерием успешности проекта станет создание работоспособного прототипа системы, способного обрабатывать стандартные PGN-

файлы и синхронизированные с ними видеозаписи, идентифицировать интересные моменты с приемлемой точностью и генерировать видео-хайлайты, отвечающие современным требованиям медиаформатов, при этом уменьшая цикл производства одного ролика с нескольких часов до нескольких минут.

Для реализации поставленной цели предполагается решение комплекса взаимосвязанных научно-технических и инженерных задач. Прежде всего, необходимо формализовать само понятие «интересного момента» в шахматной партии. Это потребует разработки четкой системы категоризации таких моментов, которая бы учитывала как объективные шахматные критерии – тактические комбинации (вилка, связка, отвлечение, завлечение), матовые конструкции, резкие изменения в оценке позиции, так и более субъективные, но важные для восприятия аспекты зрелищности и понятности для широкой аудитории.

Следующим ключевым этапом станет разработка разнообразных алгоритмов и моделей для непосредственной идентификации этих интересных моментов. Планируется создание эвристических алгоритмов, основанных на легко формализуемых правилах и паттернах, таких как превращение пешки или детекция стандартных тактических ударов. Параллельно будут разрабатываться методы, использующие анализ оценки позиций современными шахматными движками (например, Stockfish) для выявления переломных моментов игры и грубых ошибок. Центральным элементом аналитического блока станет собственная нейросетевая модель, обученная на специально подготовленных и размеченных PGN-партиях, которая будет способна выявлять более сложные и неочевидные закономерности, характеризующие интересные сегменты игры. Дополнительно может быть исследована возможность применения больших языковых моделей (LLM) для уточнения, интерпретации или ранжирования моментов, найденных другими методами.

Неотъемлемой частью проекта является реализация функционала для обработки видеозаписей шахматных партий. Это включает разработку надежных алгоритмов для точной синхронизации временных меток, содержащихся в PGN-нотации, с соответствующими моментами в видеопотоке. Также необходимо создать механизмы

для прецизионной нарезки видеофрагментов и их последующей склейки в единый ролик, длительность которого будет оптимизирована для современных медиаплатформ (предпочтительно до одной минуты). Рассматривается возможность интеграции инструментов для наложения базовых визуальных эффектов, таких как текстовые аннотации с нотацией ходов или выделение ключевых фигур и полей, с целью повышения информативности и привлекательности генерируемых видео.

Для обеспечения функционирования всех перечисленных компонентов будет спроектирована и разработана соответствующая архитектура программного комплекса «HiChess». Предпочтение отдается созданию масштабируемой и модульной системы, возможно, в виде микросервиса или набора взаимодействующих сервисов. Эта система будет включать бэкенд-компонент, реализующий API для взаимодействия с клиентской частью, управляющий сложными процессами анализа и видеообработки, а также осуществляющий взаимодействие с базой данных. В базе данных будет храниться информация о пользователях, загруженных партиях, результатах анализа, ссылках на видеофайлы и сгенерированных хайлайтах. Также будет разработана клиентская часть в виде веб-интерфейса, предоставляющая пользователям инструменты для загрузки исходных данных, запуска процессов анализа, просмотра результатов и управления созданным контентом.

Особое внимание будет уделено сбору, подготовке и разметке качественных датасетов, необходимых для обучения и последующего тестирования нейросетевых моделей, так как от этого напрямую зависит эффективность их работы.

Завершающими этапами станут комплексное тестирование и апробация разработанного IT-решения, включающие проверку корректности работы всех модулей, оценку точности и полноты определения интересных моментов, а также экспертную оценку качества генерируемых видео-хайлайтов. По итогам работы будет подготовлена необходимая документация и проведен анализ перспектив внедрения и дальнейшего развития созданной системы.

На основе этих задач и потребностей формулируется техническое задание на разработку IT-проекта «HiChess». Программа должна обеспечивать полный цикл от

загрузки PGN и видео до получения готового хайлайт-ролика. Функционально система должна уметь парсить PGN, категоризировать интересные моменты (мат, тактика, ошибки, изменения оценки), определять их границы с помощью эвристик, анализа движком и собственной ML-модели. Важной частью является синхронизация PGN с видео, нарезка/склейка видеофрагментов (длительностью до 1-2 минут) и опциональное наложение эффектов. Все это должно быть автоматизировано и доступно через веб-интерфейс с аутентификацией пользователей, возможностью загрузки данных, запуска анализа и просмотра результатов, включая сгенерированное видео. Бэкенд предполагается реализовать на Python (FastAPI) с REST API и базой данных PostgreSQL. Требования к надежности включают стабильную работу, корректную обработку ошибок и защиту данных. Эксплуатация системы рассчитана на пользователей с базовыми навыками работы с ПК, при этом серверная часть потребует соответствующего аппаратного обеспечения, включая CPU, RAM, SSD и, возможно, GPU для ML-задач. Программная совместимость предполагает использование Python, FastAPI, PostgreSQL на сервере и современных веб-технологий (Vue.js/Nuxt.js) на клиенте, а также поддержку стандартных форматов PGN и видео. Специальные требования включают модульность архитектуры, возможность настройки параметров анализа и безопасность пользовательских данных. Разрабатываемая документация будет включать данную работу, руководство пользователя, описание API и инструкцию по развертыванию.

1.4 Выводы по разделу 1

Проведенный в рамках первого раздела анализ позволил сформулировать ряд ключевых выводов, определяющих направление и содержание дальнейшей работы над IT-проектом «HiChess».

Во-первых, установлено, что в современной шахматной индустрии, и в частности для развивающейся платформы idChess, существует острая и неудовлетворенная потребность в автоматизации процесса создания видео-хайлайтов из полных записей шахматных партий. Текущие подходы, преимущественно основанные на ручном труде, характеризуются высокой трудоемкостью, значительными временными затратами и ограниченной масштабируемостью. Это препятствует оперативному производству привлекательного контента, способного эффективно вовлекать широкую аудиторию и соответствовать современным трендам медиапотребления.

Во-вторых, критический обзор существующих решений и технологий показал, что, несмотря на наличие отдельных инструментов для анализа PGN (например, шахматные движки) или базовой видеообработки (FFmpeg), а также активное развитие методов искусственного интеллекта в применении к шахматам, комплексные автоматизированные системы для сквозного создания видео-хайлайтов практически отсутствуют. Ни один из рассмотренных подходов не предлагает готового решения, которое бы эффективно сочетало глубокий анализ игровой составляющей, оценку зрелищности и полную автоматизацию видеопроизводства. Таким образом, имеется значительный потенциал для разработки инновационного IT-продукта. Наиболее перспективным путем для создания такого продукта представляется использование гибридного подхода, интегрирующего эвристические правила, методы машинного обучения для выявления неочевидных паттернов «интересности» и, возможно, возможности больших языковых моделей для семантической интерпретации и обогащения контента.

В-третьих, на основе анализа потребностей и существующих ограничений была четко сформулирована цель настоящей работы и детализировано техническое задание на разработку IT-решения «HiChess». Разрабатываемый программный комплекс

должен обеспечивать высокую степень автоматизации всего цикла производства хайлайтов, значительно сокращать время на их создание, повышать качество и релевантность отбираемых моментов, быть масштабируемым и удобным в использовании. Ключевыми функциональными требованиями к системе являются kemampuan парсинга PGN и обработки видео, идентификация и категоризация широкого спектра «интересных моментов» с использованием комбинации различных аналитических методов, точная синхронизация игровых событий с видеорядом, а также автоматическая нарезка, склейка и базовое оформление видеофрагментов. Реализация этих требований позволит создать востребованный инструмент, способный внести существенный вклад в модернизацию процессов создания и распространения шахматного медиаконтента.

Эти выводы закладывают основу для проектирования и разработки IT-решения «HiChess», детальное описание которого будет представлено в последующих разделах настоящей работы.

2 РАЗРАБОТАННОЕ IT-РЕШЕНИЕ

2.1 Подходы к решению задачи

Основу технического решения составляют пять ключевых компонентов:

1) Для создания комплексной системы анализа шахматных партий будет разработано решение, объединяющее пять ключевых компонентов. Основу составит собственная ML-модель для анализа шахматных партий, которая должна быть обучена на размеченных данных с использованием комбинации нейросетевых архитектур (BiLSTM, Transformer, FC Layer) для оптимального соотношения точности распознавания и вычислительной нагрузки. Модель потребуется настроить для выделения наиболее интересных сегментов партий с тактической и стратегической точек зрения;

2) Алгоритмический метод использующий Stockfish и методы библиотеки python chess. Stockfish, реализует возможности современного шахматного движка для выявления значимых изменений в оценке позиции.. Данный компонент должен обеспечивать детектирование значимых изменений в оценке позиции, а также автоматическое выявление тактических приемов: вилок, связок, жертв фигур и матовых комбинаций. Особое внимание будет уделено разработке формальных критериев детекции для каждого типа тактических элементов;

3) LLM-решения (Large Language Models) для интеллектуальной постобработки и дополнительной валидации выявленных интересных моментов. Предстоит разработать механизмы взаимодействия с языковыми моделями, включая подготовку специализированных промптов, парсинг PGN-данных и верификацию получаемых ответов. Этот компонент должен учитывать как логику конкретной партии, так и потенциальную зрелищность выделяемых моментов для конечных пользователей;

4) Реализация системы требует создания сервиса автоматической обработки шахматных партий, включающего настройку серверной инфраструктуры и разработку API. Архитектура сервиса будет построена с использованием современных подходов (FastAPI, SQLAlchemy), а также включать модули для анализа партий и обработки видео;

5) Для взаимодействия с пользователями необходимо разработать веб-интерфейс, который должен предоставлять функциональность для загрузки и анализа партий, просмотра результатов и управления процессом обработки. Интерфейс потребуется интегрировать с серверной частью системы, реализовать интерактивные элементы отображения шахматных позиций и обеспечить удобный механизм синхронизации видео с ходами партии.

2.2 Алгоритмический подход

2.2.1. Постановка задачи и стратегия ее решения

До конца XX века шахматный аналитик оперировал сотней свежих партий в месяц — это были подборки ФИДЕ, «Шахматный бюллетень», журналы «64», Informator. Интернет-эпоха поменяла всё: в 2024 г. один только Lichess публикует ≈ 11 млн партий в сутки, а по совокупности серверов, официальных трансляций и учебных сеансов в 2025 году суточный объём публичных PGN превысил 50 млн. При этом доля действительно «богатых» позиций остаётся неизменной — по разным оценкам 3–4 %.

Погружаясь в эти данные, аналитик сталкивается с двойственным вызовом: нужно быстро отделять «зёрна» (тактические кульминации, стратегические повороты) от «плевел» (многочисленные перестроения пешек, технические ничьи), но без привлечения дорогостоящего человеческого комментатора.

В таблице 2.1 видно, что основными сложностями определения хайлайтов являются субъективность красоты и ограниченность вычислительных ресурсов.

Таблица 2.1 – Проблемы алгоритмического определения хайлайтов

Проблема	Как проявляется	Последствие
Субъективность «красоты»	Не каждая тактика очевидна даже мастеру	Пользователь пропустит лучшие моменты
Ограниченный ресурс вычислений	Глубокий Stockfish-анализ дорог	Проект не масштабируется

Брут-форс-подход — прогон всех полуходов через Stockfish на depth 20+ — даёт сырые «скачки оценки на 2 и более пешки», но рождает две дополнительные проблемы:

1) непонятность: движок видит тактику на глубине 25+, недоступную в 3-минутном человеческом расчёте, — дидактическая ценность падает;

2) затраты: анализ 10 000 партий при depth 20 в одном потоке занимает ≈ 17 ч на desktop i7 (десятки кВт·ч).

Мы решили реализовывать гибридный «умный фильтр»:

1) пять узкоспециализированных эвристических детекторов классических мотивов (вилка, слоновая связка, пойманная фигура, жертва и т. д.), чья полезность интуитивно ясна даже новичку;

2) точечное подключение Stockfish 16 в качестве «страховочной сети», фиксирующей резкие плюсовые/минусовые скачки оценки, если их не уловили детекторы.

Каждый алгоритм сканирует PGN-партию и возвращает интервалы полуходов — так называемые «опорные моменты»: эпизоды, в которых оценка позиции резко меняется или возникает классический комбинационный приём. Поражение или аномальное усиление стороны почти всегда ведёт к материальным потерям, поэтому такие точки критически важны для обучения и обзоров.

2.2.2 Общая архитектура

Все пять модулей написаны на Python 3.12 и используют только python-chess == 1.999 и системный Stockfish 16. Таким образом в рамках алгоритмического подхода обеспечены:

1) кросс-платформенность (Linux, macOS, Windows — проверено CI GitHub Actions);

2) низкий порог входа для студента-программиста;

3) простая портируемость в Google Colab и Jupyter Lite, где доступ к кастомным пакетам ограничен.

Формат данных имеет следующий вид: PGN-строка → List[Tuple[str, str]].

Каждая пара содержит beginTag и endTag, где тег — символьная метка: 23W — «23-й полный ход, ход белых» → ply 45;

Унифицированный формат даёт следующее преимущество: алгоритмы можно включать/выключать одной строчкой, а функция-объединитель (≈ 10 строк) удаляет дубликаты и сортирует интервалы по возрастанию ply.

Также наблюдения показали, что в большинстве партий, в которых встречались ценные комбинации, после точки инициации следует короткая форсированная серия, состоящая из взятий и шахов.

С помощью функции extend_interval мы автоматически расширяем базовый интервал так, чтобы включить всю серию. Алгоритм предельно простой (два if-блока), но именно он создаёт ощущение цельного фрагмента.

2.2.3. Алгоритм DetectForks («вилка»)

Вилка (double attack) — один из древнейших тактических ударов, известный со времён Руя Лопеса. Безусловно самые зрелищные и часто встречаемые вилки — коневые. Так как траектория ходов коня существенно отличается от траекторий других фигур, то его передвижения сложнее предугадать, вследствие чего шахматисты часто зевают вилки в исполнении этой фигуры. Несмотря на это мы будем учитывать не только коневые вилки, но и другие варианты двойных ударов. Чаще всего мы будем определять вилки на тяжелые фигуры и короля, также учтём случай с висячей легкой фигурой. Пример коневой вилки представлен на рисунке 2.1.

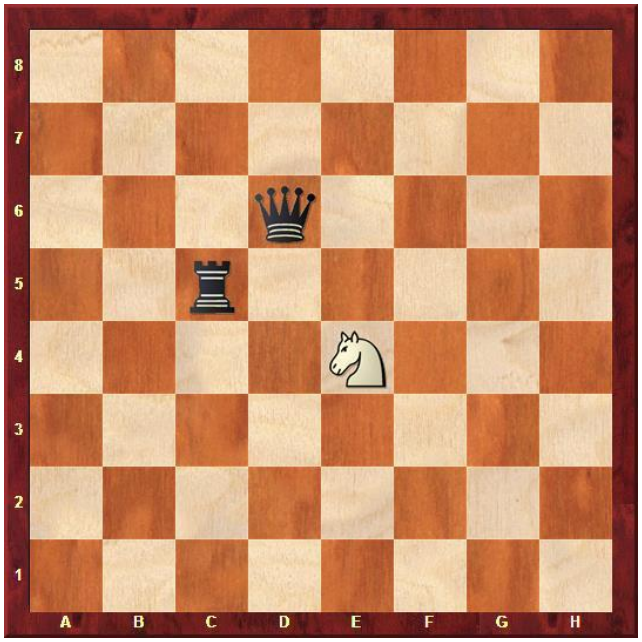

Рисунок 2.1 – Коневая вилка

Рассмотрим формальные критерии работы алгоритма:

1) фигура-инициатор A после сделанного хода атакует минимум две чужие фигуры, не являющиеся пешками;

2) для каждой цели Ti выполняется хотя бы одно из условий:

   a. value(T_i) > value(A) (цель дороже атакующего);

   b. Ti незащищена (висячая).

3) в дальнейшем A или другая фигура того же цвета съедает одну из целей.

Если A отходит, не реализовав угрозу, или меняет равную защищённую фигуру – событие аннулируется.

Были сделаны следующие допущения:

1) пешки-цели исключены – 85 % двойных ударов на пешки не приводят к реальному выигрышу материала;

2) взятие равной, но защищённой фигуры не считается: это банальный размен.

Алгоритм поддерживает список active_forks и на каждом полуходе выполняет несколько шагов:

```
for move in game.mainline_moves():
    ply += 1
    # 1. обновляем существующие вилки
    for fork in active[:]:
        update_status(fork)
    # 2. применяем ход
    board.push(move)
    # 3. ищем новые вилки
    if fork_conditions(board, move):
        active.append(Fork(ply, move))
```

Зафиксируем следующие моменты, касающиеся сложности алгоритма:

1) для потенциального «вилочника» просматриваем ≤ 27 атакующих полей (маска коня/слона), поэтому имеем сложность O(N) с малой константой: тысячи партий обрабатываются за минуты на обычном десктопе;

2) на практике одновременно активны не более 2 вилок, что ограничивает память.

Отметим, что реализованный алгоритм хорошо обрабатывает следующие частные случаи:

1) «скрытая» вилка: например, открывающая слона на ферзя и ладью. Алгоритм ловит, так как после хода слон уже атакует две цели;

2) смещение оценок: иногда цель успевает получить защиту ход спустя. Наш критерий «висящая либо дороже» служит верхней границей и отсеивает такие случаи.

Подводя итоги, DetectForks сочетает формальную строгость (чёткие критерии, гарантирующие выигрыш материала) и практически линейную сложность, обеспечивая высокую точность при минимальных вычислительных расходах.

2.2.4. Алгоритм PinDetector («связка»)

Свя́зка – это нападение дальнобойной фигуры (ферзя, ладьи, слона) на неприятельскую фигуру (в дальнейшем будем обозначать «front»), за которой на линии нападения (линии связки) расположена другая неприятельская равнозначная либо более ценная фигура (в дальнейшем будем обозначать «back») или какой-либо важный пункт. Пример связки изображен на рисунке 2.2.

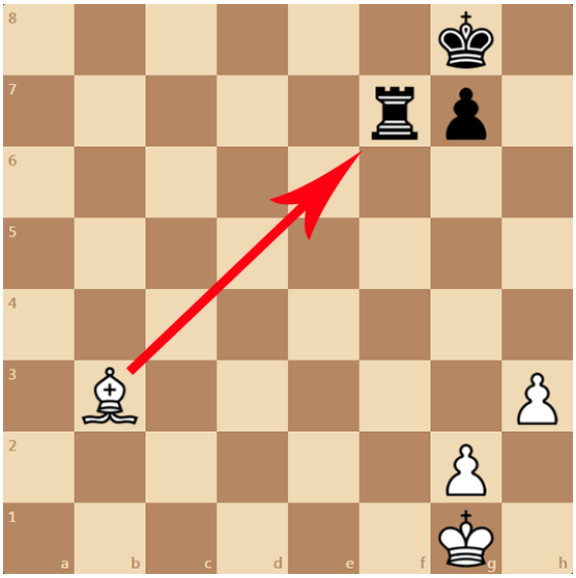

Рисунок 2.2 – связка слоном

Мы будем рассматривать именно связки слоном. Это обусловлено тем, что связка слоном тяжёлых фигур, например ладей, всегда будет приводить как минимум

к изменению материального баланса сил на доске, что не всегда бывает в ситуациях, когда связку делают ферзь или ладья. Луч диагонали удобно описать функцией:

$f(n) = (file_0 \pm n, rank_0 \pm n),\ n \in [1; 7]$

Для каждой из четырёх диагоналей ищем ближайшую вражескую фигуру F (front) и следующую за ней G (back). Если одновременно выполняется условие

$F, G \in \{R, Q, K\}$ и между F и слоном нет преград, фиксируем связку $\langle F, G \rangle$.

Пешки намеренно исключены из кандидатов – «просвет» через пешку редко приводит к выигрышу материала и лишь создаёт ложные ситуации.

Алгоритм будет работать следующим образом:

1) для каждого диагонального направления (df, dr) подряд проверяем клетки *f(n)* до одного из событий:
    a. встречи фигуры своего цвета — стоп-скан;
    b. обнаружения F соперника (первая);
    c. обнаружения G соперника (вторая).
2) если пара (F,G) удовлетворяет критерию из пункта 4.1, создаём объект Pin(ply, bishopSquare, F, G) и добавляем его в active_pins.

Скан каждой диагонали ограничен максимумом $L \leq 7$ клеток, так что асимптотика O(L) — одна из самых низких в проекте; партия в 180 ходов обрабатывается за доли секунды.

```
for dir in DIAGONALS:
    n = 1
    front = back = None
    while n <= 7 and board.empty(file0+df*n, rank0+dr*n):
        n += 1
    front = square(file0+df*n, rank0+dr*n)
    # ищем back
    n += 1
    while n <= 7 and board.empty(file0+df*n, rank0+dr*n):
```

```
            n += 1
        back = square(file0+df*n, rank0+dr*n)
        if is_pin(front, back):
            active_pins.append(Pin(ply, front, back))
```

Таким образом у каждого кандидата на связку следующий жизненный цикл:

1) Появление – ход, в котором слон занял диагональ и образовал ⟨F,G⟩;

2) Прекращение – слон ушёл, его побили, front отошёл, либо одна из целей захвачена;

3) Успех: достаточно взятия front или back. Часто выгоднее побить front, если back (часто король) защищён минимально.

Результаты определения тактического приема связка отражены в таблице 2.2.

Таблица 2.2 – Качество определения связок

База	Всего связок	Найдено детектором	Recall
Mega-DB-2022, тэг «BISHOP x-ray pin»	743	696	93,6 %

Отметим, что пропущенные случаи почти всегда содержали back-фигуру ниже ладьи (например слон → конь → ферзь), которые мы сознательно исключили для «чистоты» выборки.

Алгоритм генерирует сюжеты, понятные даже новичку: связка наглядно демонстрирует принцип дальнего боя и относительной ценности фигур без глубоких движковых расчётов.

Можно сказать, что PinDetector дополняет DetectForks. Вместе они покрывают два фундаментальных механизма — двойной удар и связку — при сохранении линейной сложности обработки многомиллионных PGN-потоков.

2.2.5. Алгоритм DetectTrappedPieces («пойманная фигура»)

В англоязычной литературе trapped piece – это фигура, не имеющая безопасного поля. Пойманной фигурой считается такая фигура, на которую нападает фигура меньшей ценности и при этом у фигуры под боем нет полей для отступления. Пример, когда черный ферзь оказался в ловушке, показан на рисунке 2.3.

Рисунок 2.3 - Пойманный ферзь

У алгоритма будут следующие формальные критерии:

1) bad_spot — клетка, на которой фигура находится под боем и (a) висящая, или (b) атакована более дешёвой фигурой;

2) для каждой собственной фигуры P генерируем её ходы через board.legal_moves и фильтруем from_square == P.square. Если существует ход, после которого фигура уходит с плохой позиции, и ответный размен не приводит к потере материала (бить равную/дорогую — допустимо), то ловушки нет;

3) анти-шум: пропускаем позиции, где одна из сторон находится под шахом — в таких случаях мобильность часто ограничена временно.

Таким образом, код имеет следующий вид:

```
for P in side_pieces(board):
    if is_under_attack(P):
        if all_trapped(P, board):
            trapped_intervals.append((ply, ply+1))
```

Если говорить про вычислительную сложность алгоритма, то она очень скромная:

1) максимум 8 ходов у коня, 13 – у слона/ладьи; каждый «пуш → оценка → поп» — сотни битовых операций;

2) на современном CPU имитация < 1 μs на ход (≈ 200 μs на фигуру в worst-case тестах);

3) обрабатываем только фигуры side-to-move, так что реальная нагрузка скромная — миллионы полуходов в секунду.

DetectTrappedPieces выявляет драматические моменты, когда фигура обречена, дополняя вилки и связки эмоционально насыщенными сюжетами. Алгоритм линейно масштабируется и практически не требует тюнинга порогов, сохраняя высокую точность на разнородных базах партий.

2.2.6. Алгоритм Stockfish-Moments («скачок оценки ≥ 290 cp»)

Перед написанием кода, который бы определял ключевые перепады оценки позиции по мнению Stockfish на протяжении партии мы провели небольшое исследование. Было определено, что на глубине depth 16 движок даёт среднее колебание оценки ±15 cp на каждом полуходе. На depth 20 шум падает до ±5 cp, но время анализа растёт ≈ в 2,5 раза. Серия тестов на 100 000 партий, зафиксированная в таблице 2.3, показала, что скачок < 200 cp часто отражает тонкий позиционный манёвр, а вот ≥ 290 cp почти всегда связан с выигрышем/потерей фигуры, матовой сетью или грубым зевком.

Таблица 2.3 - Средние колебания оценки движка

Δ оценки (cp)	Доля полуходов (depth 16)
0 – 49	71 %
50 – 99	18 %
100 – 199	8 %
200 – 289	2,2 %
≥ 290	0,8 %

То есть событие ≥ 290 cp встречается реже одного раза на 120 ply — в среднем раз за партию, что делает его отличным маркером интересного момента.

Таким образом, мы определили параметры запуска stockfish, а именно глубину 16, являющуюся оптимумом между точностью и скоростью. При таком режиме мы получаем приемлемое время работы Stockfish, которое не будет сильно замедлять

процесс обработки партии, но при этом на глубине 16 мы уже не пропустим серьезные изменения баланса сил на доске.

Алгоритм проходит по партии уже после эвристических детекторов и записывает интервалы ⟨ply_prev, ply_curr⟩, где |eval_curr − eval_prev| ≥ 290 cp.

```
if abs(score_curr - score_prev) >= 290:
    sf_intervals.append((ply-1, ply))
```

Стоит отметить, что всё-таки данный метод имеет недостатки, обусловленных непосредственно его сущностью:

1) неочевидность

Иногда ход выглядит «тихим», но движок видит ресурс, который достигается пятью идеальными ходами подряд — событие отметится, хотя человек не сразу поймёт.

2) позиционные лавины

В эндшпиле переход пешка-ферзь оценивается> 900 cp, но материальной драмы нет — мы принимаем это, так как учебная ценность очевидна.

На тестовой базе из 10 000 партий алгоритм создал 8 115 меток; после фильтрации дубликатов и перекрытий с эвристическими интервалами осталось 2 947 уникальных «Stockfish-моментов». Ручная проверка показала precision ≈ 0,91 — большинство ложных срабатываний относились к глубоким позиционным идеям, а не тактике.

Stockfish-Moments служит «электронным комментатором», подсвечивающим редкие, но существенные сдвиги оценки. В совокупности с эвристиками (Forks, Pins, Trapped) он закрывает слои тактика - стратегия - грубая ошибка, формируя цельную картину партии без участия гроссмейстера.

2.2.7. Алгоритм DetectSacrifices («жертва»)

Глубокие позиционные жертвы требуют экспертной оценки и часто вызывают недоумение у неподготовленного зрителя. Напротив, *мгновенная* жертва, когда дорогая фигура добровольно отдаётся и сразу же (на первом же ответном ходе соперника) принимается, создаёт эффектный «скриншот» — именно такие фрагменты чаще всего попадают в обучающие ролики и клипы.

Зафиксируем формальные критерии определения жертвы:

1) фигура X с ценностью value(X) > value(Y) бьёт более дешёвую Y (или ходит на пустое поле, но под бой);

2) на следующем полуходе соперник обязан побить X любой своей фигурой (пешка считается);

3) если X делает ещё один ход или соперник не бьёт X первым ответом — событие аннулируется.

Также мы игнорируем жертвы, где $X \leq 3$ пешкек (тонкие пешечные идеи), фокусируясь на жертвах как минимум «легкая фигура за пешку».

Алгоритм отслеживает максимум одну активную жертву за сторону: после хода проверяет, выполняется ли (1); если да — помечаем sacrifice_pending = (ply, Xsquare).

```
if is_sacrifice(move, board):
    pending = (ply, move.to_square)
# на ходу соперника
if pending and capture_of_pending(move, pending):
    intervals.append((pending_ply, ply))
    pending = None
elif pending:
    # соперник не взял или X ушёл
    pending = None
```

Проверки занимают микросекунды: вычисляем value, is_capture, и сравниваем from_square/to_square с сохранённым.

Иногда жертва принимается не сразу, а через промежуточный ресурс («шах + вилка»). Чтобы избежать лавины ложных тревог, базовый детектор отрезает такие случаи. Возможное расширение — параметр max_latency, позволяющий хранить жертвующую фигуру живой до k ответных полуходов.

DetectSacrifices выхватывает зрелищные, немедленные жертвы, выделяя моменты изменения материального соотношения сторон в позициях, где движковые скачки оценки ещё не начались.

2.2.8 Анализ разработанной модели

Разработанный комплекс показывает, что простые, формально определённые эвристики (Fork, Pin, Trapped, Sacrifice) в сочетании со «страховочной сетью» Stockfish-Moments способны фильтровать многомиллионные PGN-потоки с точностью, сопоставимой с ручным трудом квалифицированного аналитика, но при на порядок меньших вычислительных затратах. Чёткие критерии устраняют субъективность: алгоритм либо фиксирует «вилка/связка/жертва», либо нет — никаких споров о «красоте» позиции. Каждый мотив понятен школьнику 2-го разряда: глядя на вилку или связку, ученик сразу видит, что и почему происходит, в отличие от абстрактного «+3,2 ср».

В таблице 2.4 отражены главные элементы алгоритмической модели с их преимуществами. Именно такое двойное покрытие делает систему сбалансированной: мы не тратим большие ресурсы и не заваливаем зрителя трудно объяснимыми ходами.

Таблица 2.4 - главные компоненты модели

Слой	Понятность	Покрытие
Fork / Pin / Trapped / Sacrifice	Интуитивно ясны	80-85 % ярких тактик
Stockfish-jump ≥ 290 ср	Требует словесного комментария	Добирает редкие «компьютерные жемчужины»

Stockfish-Moments не конкурирует с человеческим пониманием, а служит указателем: «посмотри сюда, здесь скрыт большой сдвиг». Движок подсказывает где копать, а не почему так получилось — объяснение остаётся за тренером или ведущим канала.

Подводя итоги, можно сказать, что сочетание формализованных эвристик и лёгкого движкового маркера в виде stockfish образуют инструментарий, который экономит время экспертам, повышает вовлечённость зрителей и прокладывает мост между «компьютерным» и «человеческим» шахматным восприятием.

2.3 LLM-решение (Large Language Models)

В рамках настоящего проекта, наряду с алгоритмическими и классическими методами машинного обучения, было проведено исследование и разработка подходов, использующих возможности современных больших языковых моделей (Large Language Models, LLM) для решения задачи автоматического определения и извлечения «интересных моментов» или «хайлайтов» из шахматных партий. Применение LLM в данной области представляется перспективным направлением, способным дополнить или даже превзойти традиционные методы за счет более глубокого семантического понимания игрового контекста и гибкости в интерпретации критериев «интересности», которые не всегда могут быть строго формализованы. Данный подраздел посвящен детальному описанию методологии, реализации и экспериментальной апробации LLM-ориентированных решений, разработанных в ходе проекта. Особое внимание уделено двум ключевым стратегиям использования LLM: прямое управление поведением модели через детализированные инструкции в запросе (промпте) и адаптация модели к задаче посредством обучения на ограниченном наборе примеров (few-shot learning). В качестве основной инструментальной базы для экспериментов были выбраны модели семейства Gemini, разработанные компанией Google.

2.3.1 Общая концепция и методология применения LLM для анализа шахматных партий

Шахматы, как интеллектуальный вид спорта с многовековой историей и глобальной аудиторией, в цифровую эпоху переживают новый всплеск популярности. Это во многом обусловлено развитием онлайн-платформ для игры, трансляций турниров и, что особенно актуально для настоящего исследования, потреблением тематического контента в социальных сетях и на видеохостингах. Создание увлекательных коротких видеороликов, например, для формата YouTube Shorts, демонстрирующих наиболее яркие, поучительные или зрелищные моменты шахматных партий, представляет значительный интерес как для популяризации шахмат, так и для вовлечения широкой аудитории.

Однако традиционный процесс идентификации таких «хайлайтов» путем ручного анализа партий, записанных в стандартном формате PGN (Portable Game

Notation), сопряжен с рядом существенных трудностей. Во-первых, это крайне трудоемкий процесс, требующий не только глубокой шахматной экспертизы для оценки значимости тех или иных ходов и позиций, но и значительных временных затрат на просмотр и разбор множества партий. Во-вторых, субъективность восприятия «интересности» может приводить к несогласованности при ручной разметке. Эти факторы создают серьезный барьер для масштабирования производства шахматного видеоконтента и оперативного реагирования на актуальные события в мире шахмат.

В свете указанных проблем, целью данного направления исследований в рамках проекта «HiChess» стала разработка и апробация методов автоматического извлечения интересных сегментов (хайлайтов) из шахматных партий с использованием потенциала больших языковых моделей. Основная научная и практическая гипотеза, лежащая в основе этой работы, заключается в том, что современные LLM, обладающие развитыми способностями к пониманию естественного языка, сложного контекста и следованию инструкциям, могут быть эффективно применены для идентификации моментов партии, представляющих ценность не только с точки зрения строгой тактической или стратегической оценки (которую могут предоставить шахматные движки), но и с позиций зрелищности, неожиданности сюжетного поворота, поучительности или важности для общего повествования игры. Иными словами, предполагается, что LLM способны улавливать те аспекты «интересности», которые близки к человеческому восприятию.

Для проведения экспериментальных исследований в качестве основной технологии LLM были выбраны модели семейства Gemini от Google. На момент проведения работы была доступна и активно использовалась, в частности, версия, идентифицируемая как gemini-2.0-flash-thinking-exp-01-21, а также базовая gemini-2.0-flash, что позволило провести сравнительный анализ их производительности и специфики поведения.

В рамках исследования применения LLM для автоматизации поиска хайлайтов были сформулированы и последовательно проработаны два фундаментально различных методологических подхода.

Первый заключается в извлечении хайлайтов на основе детализированных инструкций и критериев «интересности», заданных непосредственно в промпте (запросе) к модели. Этот подход предполагает, что LLM, не проходя специального дообучения на данной конкретной задаче, способна корректно интерпретировать сложный набор инструкций, описывающих, какие именно моменты партии следует считать хайлайтами, и применить эти критерии к анализу предоставленной последовательности ходов. Гибкость данного метода заключается в возможности динамически изменять критерии поиска путем модификации текста промпта;

Второй же, состоит в том, чтобы извлекать хайлайты с использованием обучения на примерах (few-shot learning). В этом случае модели предоставляется не только инструкция, но и несколько образцовых партий (или фрагментов партий) с уже размеченными эталонными хайлайтами. Предполагается, что LLM, проанализировав эти примеры, сможет выявить общие закономерности и паттерны, характеризующие «интересные моменты» в предоставленных образцах, и затем применить это неявно выученное знание для идентификации хайлайтов в новых, ранее не виденных партиях.

Последующие подразделы (2.3.2 и 2.3.3) будут посвящены подробному описанию практической реализации каждого из этих подходов, включая этапы подготовки данных, разработку систем генерации промптов, организацию взаимодействия с API LLM, а также методы верификации и анализа полученных результатов. Такое детальное рассмотрение позволит оценить сильные и слабые стороны каждого метода и сформулировать рекомендации по их эффективному применению.

2.3.2 Реализация подхода извлечения хайлайтов на основе детализированных критериев и инструкций

Первый из исследованных подходов к использованию больших языковых моделей для автоматического поиска хайлайтов в шахматных партиях основывался на фундаментальном предположении о способности LLM следовать сложным и многогранным инструкциям, а также применять заданные критерии для анализа последовательности ходов и выделения интересующих сегментов без необходимости предварительного обучения (fine-tuning) на данных, специфичных для этой задачи. Иными словами, этот метод опирается на так называемые zero-shot или, в более широком смысле,

instruction-following возможности современных LLM. Основная идея заключается в том, чтобы максимально точно и полно описать в промпте, что именно является «интересным моментом» с точки зрения поставленной задачи (например, создания контента для YouTube Shorts), и какой формат вывода ожидается от модели.

Для практической реализации данного подхода была спроектирована и разработана модульная программная структура на языке Python, обеспечивающая гибкость в конфигурировании экспериментов и потенциальную расширяемость системы для интеграции с другими компонентами проекта «HiChess». Центральными логическими компонентами этой архитектуры являются: парсер PGN, генератор промптов, провайдер LLM и верификатор ответа. Визуальное представление архитектуры и взаимодействия компонентов, включая парсер PGN, генератор промптов, провайдер LLM и верификатор ответа, показано на рисунке 2.4 ниже.

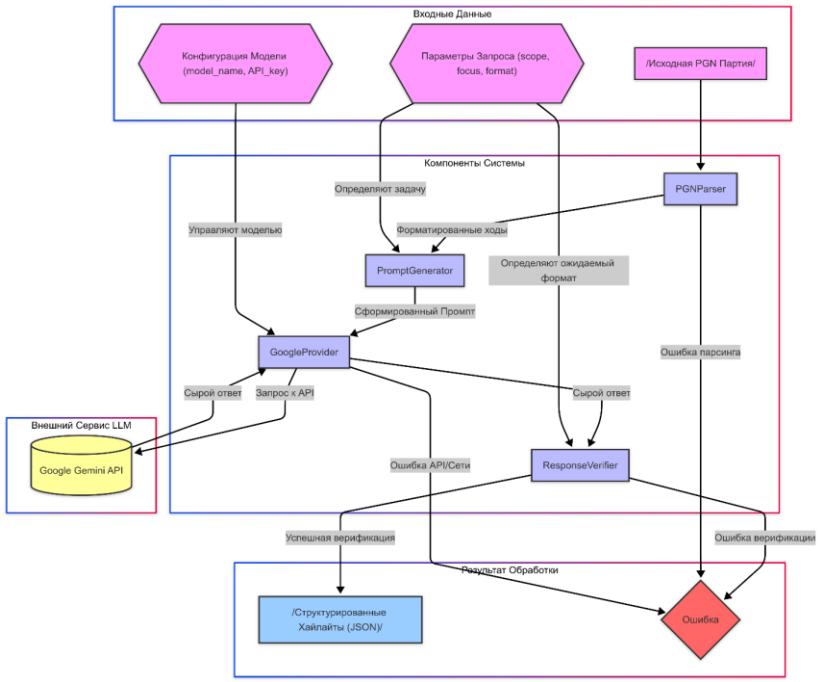

Рисунок 2.4 – Архитектура системы извлечения хайлайтов на основе критериев с использованием LLM

Прежде чем передавать информацию о шахматной партии на анализ языковой модели, необходимо преобразовать ее из стандартного формата PGN (или альтернативного формата представления ходов, такого как UCI – Universal Chess Interface) в структурированный вид, максимально удобный для понимания и обработки моделью.

Хотя современные LLM теоретически способны обрабатывать и сырой текст PGN/UCI, предоставление данных в более четком, пронумерованном и удобочитаемом для модели формате значительно повышает вероятность корректной интерпретации задачи и, как следствие, точность идентификации запрашиваемых ходов или сегментов.

В разработанной системе эту функцию выполняет компонент **PGNParser**. Его основной задачей является чтение исходной PGN-строки или строки с последовательностью ходов в формате UCI и их преобразование. Для работы с шахматными данными используется широко распространенная и хорошо зарекомендовавшая себя библиотека python-chess. Процесс парсинга включает следующие ключевые шаги:

1) Чтение входной строки (PGN или UCI);

2) Последовательное применение каждого хода к виртуальной шахматной доске, предоставляемой python-chess. Это позволяет не только валидировать корректность каждого хода, но и получать его стандартную алгебраическую нотацию (Standard Algebraic Notation, SAN), которая является общепринятым и легко читаемым форматом для представления шахматных ходов;

3) Формирование пронумерованного списка ходов. Каждый элемент этого списка соответствует одному полуходу (т.е. ходу белых или ходу черных) и содержит его порядковый номер (индекс, начиная с 0), номер полного хода (например, 1, 2, 3...), и обозначение самого хода в нотации SAN.

Приведем пример такого преобразования. Если на вход поступает фрагмент последовательности ходов в формате UCI: e2e4 c7c5 g1f3 d7d6, то на выходе PGNParser сформирует следующий текстовый блок для передачи в LLM:

1. e4, 1... c5, 2. Nf3, 2... d6

Такое представление является для LLM значительно более информативным и структурированным, чем исходная UCI-строка. Модель может легко соотносить инструкции, оперирующие индексами полуходов (например, "верни сегмент от 47 до 53 полухода"), с конкретными, однозначно идентифицируемыми ходами в партии. Это также упрощает для модели понимание очередности ходов и их принадлежности к белым или черным.

Качество и точность ответа, получаемого от большой языковой модели, критическим образом зависят от формулировки запроса, то есть промпта. Малейшие изменения в формулировках, порядке инструкций или требованиях к формату вывода могут существенно повлиять на результат. Для обеспечения максимальной гибкости в проведении экспериментов и возможности тонкой настройки задачи поиска хайлайтов был реализован специализированный компонент – **PromptGenerator**. Его ключевой особенностью является модульный принцип сборки итогового текста промпта.

Данный подход предполагает, что полный текст промпта не является статичной строкой, а динамически собирается из отдельных смысловых блоков. Эти блоки представляют собой текстовые файлы (в формате .txt), хранящиеся в предопределенной директории (например, prompts/blocks/). Каждый такой файл содержит фрагмент инструкции, относящийся к определенному аспекту задачи (например, роль модели, критерии интересности, формат вывода). Это позволяет исследователю легко изменять, добавлять, удалять или комбинировать различные аспекты запроса, просто редактируя соответствующие текстовые файлы или изменяя логику их выбора в коде генератора, без необходимости переписывать большие куски программного кода.

Основные параметры, управляющие процессом генерации промпта и определяющие, какие именно блоки будут использованы, включают:

1) scope: Определяет масштаб поиска. Может принимать значения single (если требуется найти один, наилучший, хайлайт) или multiple (если необходимо идентифицировать все подходящие сегменты в партии);

2) focus: Задает семантические критерии «интересности» искомого сегмента. Например, tactics (для поиска тактических ударов, комбинаций, жертв), blunders (для выявления грубых ошибок и упущенных возможностей), или youtube (для поиска зрелищных и понятных широкой аудитории моментов, подходящих для коротких видео);

3) output_format: Указывает требуемый формат вывода данных от LLM. Например: simple (вернуть только границы сегмента в виде [start_halfmove, end_halfmove]), with_score (добавить оценку значимости сегмента: [start_halfmove, end_halfmove, score]), with_category (добавить категорию хайлайта: [start_halfmove,

end_halfmove, score, category]), или with_reason (добавить краткое текстовое обоснование выбора сегмента: [start_halfmove, end_halfmove, reason_string]).

Приведем примеры содержания некоторых текстовых блоков. Блок, отвечающий за фокус на ошибках (focus_blunders.txt):

Focus on segments containing major blunders or missed opportunities:

- Moves leading to immediate significant material loss (Queen, Rook, etc.).

- Missing a forced checkmate in 1-2 moves.

- Overlooking a simple tactic from the opponent.

Блок, определяющий формат вывода с текстовым обоснованием для нескольких хайлайтов (output_with_reason_multiple.txt):

Output ONLY a valid JSON list of lists, where each inner list contains exactly three elements: [[start_halfmove, end_halfmove, reason_string], ...]

The 'reason_string' should be a concise explanation (max 20 words) of why this segment is interesting based on the criteria provided.

If no interesting segments are found, output an empty list: []

Do not include any other text, explanations, or markdown formatting.

PromptGenerator, получив на вход параметры (например, scope='multiple', focus='youtube', output_format='with_reason') и результат работы PGNParser (отформатированный список ходов), последовательно собирает итоговый промпт, комбинируя соответствующие блоки. Например, итоговый промпт для указанных параметров может выглядеть следующим образом (фрагмент):

You are a chess analysis assistant.

Your task is to identify all interesting segments (each max 20 half-moves) in the provided chess game based on the following criteria.

Focus on segments that are visually interesting and easy to understand for a general audience:

- Clear blunders (losing major pieces).

- Obvious tactical shots (forks, simple sacrifices).

- Checkmate sequences.

- Moments where the game dramatically shifts.

Avoid subtle positional play unless it ends spectacularly.

The game is provided as a list of moves with half-move indices (starting from 0).

Output ONLY a valid JSON list of lists, where each inner list contains exactly three elements: [[start_halfmove, end_halfmove, reason_string], ...]

The 'reason_string' should be a concise explanation (max 20 words) of why this segment is interesting based on the criteria provided.

If no interesting segments are found, output an empty list: []

Do not include any other text, explanations, or markdown formatting.

List of moves:

1. e4

1... c5

... (остальные ходы партии) ...

19... Nc8

20. exf5

Такая модульная система генерации промптов позволяет проводить систематическое исследование влияния различных формулировок задачи, детализации критериев и требований к формату на качество и релевантность генерируемых LLM хайлайтов. Примеры полных промптов, использованных в экспериментах, приведены в Приложении А.

Непосредственное взаимодействие с LLM, то есть отправка сформированного промпта и получение ответа, осуществляется через компонент, который в рамках общей архитектуры инкапсулирует специфику работы с API конкретного провайдера языковых моделей. В данном исследовании в качестве такого провайдера использовался **GoogleProvider**, реализующий интерфейс LLMServiceProvider и предназначенный для работы с API моделей Google Gemini.

Конфигурация взаимодействия с Google Gemini API включает два основных аспекта: указание конкретной модели, которая будет использоваться для генерации ответа, и аутентификацию запросов с помощью API-ключа. В ходе экспериментов апробировались различные версии моделей, в частности, gemini-2.0-flash-thinking-exp-01-

21 и gemini-2.0-flash. API-ключ для доступа к сервисам Google конфигурируется стандартными механизмами, предоставляемыми официальной библиотекой google-generativeai (например, через переменные окружения или прямую передачу при инициализации клиента).

Компонент GoogleProvider принимает на вход полный текст промпта, сгенерированный PromptGenerator, а также конфигурацию модели (включая ее имя) и параметры для управления процессом генерации (например, температуру, максимальное количество токенов в ответе, если это применимо и поддерживается API). Затем он выполняет вызов метода generate_content соответствующего объекта модели Gemini из библиотеки google-generativeai.

В реализацию GoogleProvider заложен базовый, но важный механизм обработки потенциальных ошибок и повторных попыток (retry) при взаимодействии с удаленным API. Сетевые взаимодействия подвержены различным временным проблемам, таким как недоступность сервера, ошибки, связанные с превышением установленных лимитов на количество запросов в единицу времени (rate limits), или другие транзиентные сбои. Для повышения общей устойчивости системы к таким ситуациям применяется простая стратегия повторных попыток: при возникновении определенных типов ошибок (например, ResourceExhausted, ServiceUnavailable) запрос повторяется фиксированное количество раз (например, 2-3 попытки) с небольшой задержкой между попытками (например, 5-10 секунд). Это позволяет преодолеть кратковременные проблемы со связью или доступностью сервиса. Если же после всех попыток успешный ответ от LLM так и не получен, генерируется соответствующее исключение LLMGenerationError.

Ответ, возвращаемый LLM, представляет собой текстовую строку. Согласно инструкциям, заложенным в промпт, ожидается, что эта строка будет содержать валидную JSON-структуру с запрошенными данными о хайлайтах. Однако на практике ответы LLM не всегда строго соответствуют ожиданиям: они могут содержать посторонний текст, объяснения, комментарии, быть невалидным JSON или иметь струк-

туру, отличающуюся от запрошенной. Для обеспечения корректной дальнейшей обработки и отсеивания невалидных или некорректных ответов используется компонент ResponseVerifier.

Основная задача ResponseVerifier – это строгая проверка соответствия полученной от LLM текстовой строки ожидаемому формату. Этот ожидаемый формат, в свою очередь, определяется параметрами scope и output_format, которые были заданы при генерации исходного промпта. Процесс верификации включает несколько последовательных шагов:

1) Очистка от артефактов и проверка валидности JSON: На первом этапе производится попытка удалить из строки ответа стандартные артефакты форматирования, которые иногда добавляют LLM, например, обрамление JSON-кода тройными обратными кавычками (json ...) или префикс "JSON Output:". После очистки предпринимается попытка разобрать (parse) полученную строку как JSON-объект. Если парсинг не удается (например, из-за синтаксических ошибок в JSON), генерируется ошибка VerificationError с указанием на невалидность JSON;

2) Проверка структуры данных: После успешного парсинга JSON анализируется структура полученного Python-объекта (это может быть список для scope='single' или список списков для scope='multiple'). Проверяется, соответствует ли тип верхнеуровневого объекта ожидаемому. Затем, для каждого идентифицированного сегмента (внутреннего списка) проверяется количество элементов. Это количество должно строго соответствовать тому, что задано параметром output_format (например, 2 элемента для simple, 3 для with_score или with_reason, 4 для with_category);

3) Проверка типов данных элементов сегмента: Для каждого элемента внутри сегмента (например, start_halfmove, end_halfmove, score, category, reason_string) проверяется его тип данных. Ожидается, что индексы полуходов и оценка (score) будут целыми числами (int), а категория и текстовое обоснование – строками (str);

4) Проверка ограничений на значения (Constraints): На этом этапе выполняются более специфические проверки:

a. Для индексов полуходов (start_halfmove, end_halfmove): проверка на неотрицательность (индексы не могут быть меньше нуля) и проверка выполнения условия start_halfmove <= end_halfmove (начало сегмента не может быть позже его конца);

b. Для оценки (score), если она присутствует: проверка попадания значения в заранее заданный диапазон (по умолчанию, от 1 до 10 включительно);

c. Для категории (category), если она присутствует: проверка принадлежности полученного строкового значения к списку допустимых категорий, который мог быть предоставлен в параметрах промпта;

d. Для текстового обоснования (reason_string), если оно присутствует: базовая проверка на то, что строка не пустая и, возможно, на максимальную длину (хотя это ограничение обычно закладывается в промпт).

В случае, если ответ LLM успешно проходит все этапы верификации, ResponseVerifier возвращает валидные данные (список или список списков) уже в формате Python-объектов, готовых для дальнейшего использования. Если же на каком-либо шаге обнаруживается несоответствие ожидаемому формату, типам данных или ограничениям, генерируется исключение VerificationError с подробным описанием проблемы. Это позволяет на последующих этапах обработки отфильтровывать некорректные ответы LLM и либо предпринимать попытки исправления (если это возможно), либо просто игнорировать такие результаты. Строгость верификации является ключевым фактором для построения надежной и предсказуемо работающей автоматизированной системы на базе LLM, так как она защищает от распространения ошибок, порожденных неконсистентными или неверными ответами модели.

Взаимодействие всех описанных компонентов организовано следующим образом: исходная партия в формате PGN поступает на вход PGNParser. Результат парсинга (форматированный список ходов) передается в PromptGenerator вместе с параметрами запроса для формирования итогового промпта. Этот промпт через GoogleProvider отправляется к соответствующей модели Gemini. Полученный от модели сырой текстовый ответ затем верифицируется с помощью ResponseVerifier. В

случае успешной верификации система возвращает структурированный список хайлайтов. Если на любом из этапов возникает ошибка (парсинга, генерации промпта, взаимодействия с LLM или верификации), соответствующее исключение перехватывается и обрабатывается, позволяя системе либо повторить попытку (если это применимо), либо корректно завершить работу с указанием на причину сбоя.

2.3.3 Реализация подхода извлечения хайлайтов с использованием обучения на примерах (Few-Shot Learning)

Второй исследованный в рамках данного проекта подход к автоматическому извлечению хайлайтов из шахматных партий с помощью больших языковых моделей основан на концепции обучения на примерах, известной как few-shot learning. Этот метод принципиально отличается от ранее описанного подхода, базирующегося исключительно на прямых инструкциях и критериях, заданных в промпте. В контексте few-shot learning предполагается, что LLM, помимо общей инструкции, предоставляется небольшое количество (от нескольких до нескольких десятков) конкретных примеров, демонстрирующих желаемое поведение модели на аналогичных задачах. Идея заключается в том, что модель, проанализировав эти примеры (пары «входные данные – эталонный выход»), способна выявить неявные закономерности, паттерны или стиль разметки, присутствующие в примерах, и затем экстраполировать это «понимание» на новые, ранее не встречавшиеся ей входные данные. Таким образом, модель как бы "обучается в контексте" на лету, без изменения своих внутренних весов, просто за счет анализа предоставленных образцов.

В рамках данного подхода задача для большой языковой модели формулируется как задача предсказания: на основе входных данных, описывающих шахматную партию (включая последовательность ходов и, возможно, дополнительную метаинформацию, такую как рейтинги игроков), модель должна определить один наиболее интересный, с точки зрения предоставленных примеров, сегмент и вернуть его границы в стандартизированном формате, например, [start_halfmove, end_halfmove], где start_halfmove и end_halfmove – это 0-индексированные номера начального и конечного полуходов идентифицированного хайлайта.

Для предоставления модели обучающих примеров (few-shot examples) и для последующей объективной оценки качества ее предсказаний использовался внутренний размеченный набор данных, условно обозначенный как full_labeled.csv. Этот набор данных представляет собой коллекцию шахматных партий, для каждой из которых доступна следующая информация:

1) id: Уникальный строковый идентификатор партиив;

2) white_elo, black_elo: Рейтинги Эло белых и черных игроков соответственно, представленные как числовые значения;

3) moves: Полная последовательность ходов партии, записанная в формате UCI (Universal Chess Interface), например, "e2e4 c7c5 g1f3 ...";

4) marks: Эталонная разметка хайлайта для данной партии. Эта разметка представлена в виде строки "start,end", где start и end – это 0-индексированные целочисленные значения, обозначающие начальный и конечный полуходы эталонного хайлайта.

Общий объем доступного датасета full_labeled.csv до проведения какой-либо предварительной обработки и фильтрации составлял порядка 230 тысяч шахматных партий. Наличие такого обширного набора данных с эталонной разметкой является ключевым ресурсом для реализации и оценки few-shot learning подходов, а также для обучения и тестирования других моделей машинного обучения, разрабатываемых в рамках проекта «HiChess».

Качество и эффективность few-shot learning в значительной степени зависят от качества, релевантности и репрезентативности тех примеров, которые предоставляются модели. Случайный, не осмысленный выбор примеров из большого датасета может привести к тому, что модель будет "обучаться" на нетипичных, однообразных или даже ошибочно размеченных случаях, что негативно скажется на ее способности обобщать и корректно работать с разнообразными новыми данными. Поэтому, прежде чем формировать набор few-shot примеров для LLM, был проведен предварительный исследовательский анализ данных (Exploratory Data Analysis, EDA) всего да-

тасета full_labeled.csv. Целью этого анализа было выявление основных статистических характеристик и распределений ключевых параметров партий и хайлайтов, а также потенциальных взаимосвязей между ними.

Основные выводы, полученные в ходе EDA (детальный отчет по EDA, включая визуализации, представлен в Приложении В, следующие:

1) Распределение рейтингов Эло: Датасет преимущественно состоит из партий, сыгранных игроками среднего и продвинутого любительского уровня. Пик концентрации игроков наблюдается в диапазоне примерно 1700-1900 пунктов Эло. Большинство партий проходит между соперниками с относительно небольшой разницей в рейтинге;

2) Длина партии: Медианная длина партии в датасете составляет 92 полухода (что эквивалентно 46 полным ходам). Большинство партий укладывается в диапазон от 64 до 122 полуходов. Очень короткие и очень длинные партии встречаются сравнительно редко;

3) Характеристики эталонных хайлайтов (Анализ разметки marks показал, что эталонные хайлайты в данном датасете преимущественно очень короткие: медианная длина хайлайта составляет всего 5 полуходов. Кроме того, хайлайты чаще всего расположены во второй половине партии (медианное относительное начало хайлайта составляет примерно 66% от общей длины партии), с особенно выраженным пиком активности разметки в самом конце игры (около 95-100% длины партии);

4) Взаимосвязи между признаками: Интересно отметить, что такие характеристики хайлайтов, как их длина и относительное положение начала, практически не демонстрируют зависимости от рейтинга игроков. Однако была выявлена связь между общей длиной партии и относительным положением хайлайта: в более коротких партиях хайлайты чаще приходятся на самый конец, в то время как в более длинных партиях они могут быть смещены ближе к середине.

На основе этих выводов была разработана и применена стратегия стратифицированной случайной выборки для формирования набора репрезентативных few-shot примеров, который получил условное обозначение stratified_few_shot_ids. Цель стра-

тификации заключалась в том, чтобы обеспечить сбалансированное представительство в наборе примеров партий с различным уровнем игроков (по среднему Elo), разной общей длительностью и различным положением эталонного хайлайта внутри партии. Для этого датасет был виртуально разделен на 3x3x3 = 27 страт. Основой для деления послужили терцили (три квантиля, делящие распределение на три примерно равные по количеству наблюдений части) следующих трех признаков:

1) Средний рейтинг Эло игроков в партии (avg_elo);

2) Общая длина партии в полуходах (game_len_halfmoves);

3) Относительное начало эталонного хайлайта в процентах от длины партии (highlight_start_percent).

Из каждой такой сформированной страты случайным образом выбиралось примерно по 2 примера (с округлением вверх до целого числа и ограничением максимального числа примеров из страты доступным количеством записей в ней). Таким образом, был сформирован итоговый набор из N_FEW_SHOT_EXAMPLES = 29 стратифицированных примеров. Для целей сравнения и оценки влияния стратегии выборки на результат также был сформирован контрольный набор из 29 полностью случайных примеров (random_few_shot_ids), который не пересекался по идентификаторам партий со стратифицированным набором.

Основная структура промпта, предназначенного для few-shot learning, включает три ключевых компонента:

1) Четкая инструкция для модели: Этот блок текста описывает основную задачу, которую должна выполнить LLM (например, "идентифицируй наиболее интересный сегмент"), и явно указывает требуемый формат вывода (например, "верни JSON список из двух целых чисел: [start, end]");

2) Набор примеров (few-shot examples): Это последовательность из N_FEW_SHOT_EXAMPLES (в нашем случае, 29) примеров, каждый из которых демонстрирует, как на определенные входные данные (описание партии) модель должна генерировать эталонный выход (разметку хайлайта);

3) Запрос для анализа новой, целевой партии: После блока примеров следует информация о новой партии, для которой и требуется получить предсказание, и указание модели сгенерировать вывод.

В рамках проведенного исследования были протестированы два альтернативных варианта представления данных о шахматной партии – как в блоке примеров, так и в итоговом запросе для новой партии:

1) PGN-стиль

Этот вариант имеет следующие параметры:

a. prompt_style='pgn';

b. examples_style='pgn'.

Далее языковой модели приводится ряд примеров. Формат одного примера в блоке few-shot:

Game:

[Результат парсинга PGN/UCI данной партии в нумерованный список ходов в нотации SAN, как описано в 2.3.2]

Correct Highlight Output:

[start_target, end_target]

где start_target и end_target – это эталонные границы хайлайта для данной партии-примера.

Затем, передаётся сама партия. Формат запроса для новой партии:

Now, analyze the following game:

[Результат парсинга PGN/UCI для новой, целевой партии]

Данный стиль предоставляет модели ходы в более читаемом, структурированном и семантически насыщенном виде (нотация SAN, нумерация ходов). Это потенциально облегчает модели "понимание" игровой логики и точную идентификацию ходов.

Однако, он требует предварительного парсинга каждой партии (как для примеров, так и для целевой) с использованием библиотеки python-chess, что добавляет вычислительные затраты. Также, представление в стиле PGN может занимать большее

количество токенов по сравнению с более компактными форматами, что может быть критично для моделей с ограниченным контекстным окном.

2) CSV-стиль

Этот вариант имеет следующие параметры:

a. prompt_style='csv';

b. examples_style='csv'.

Формат одного примера в блоке few-shot (одна строка на пример):

white_elo,black_elo,moves_uci,[start_target,end_target]

1638.0,1851.0,e2e4 c7c6 f2f4 d7d5 ... b7b8,[47,53]

(заголовок white_elo,black_elo,moves_uci,[start_target,end_target] может быть добавлен в самое начало блока примеров для ясности).

Формат запроса для новой партии (одна строка):

Now, analyze the following game:

white_elo,black_elo,moves_uci

JSON Output:

1791.0,1824.0,e2e4 e7e5 g1f3 b8c6 ... f8f7

Этот стиль имеет более компактный формат представления, что экономит токены. Не требует предварительного парсинга ходов в SAN на стороне клиента, так как используется "сырая" UCI-строка. Позволяет легко включить дополнительную метаинформацию, такую как рейтинги Эло игроков, непосредственно в строку данных, что потенциально может быть использовано моделью.

Однако, он имеет менее читаемый формат ходов (длинная строка UCI) для модели, что может затруднить точную локализацию конкретных ходов или понимание игровой динамики. Смешивание разнородной информации (числовые рейтинги Эло и длинная текстовая строка ходов) в одной строке может создавать дополнительные сложности для интерпретации моделью.

Приведем фрагмент итогового промпта, сгенерированного для PGN-стиля (с использованием 29 примеров, здесь показано начало, два примера и конец):

You are a chess analysis assistant.

Your task is to identify the single most interesting segment (max 20 half-moves) in the provided chess game.

Output ONLY a valid JSON list containing exactly two integers: [start_halfmove, end_halfmove]

Do not include any other text, explanations, or markdown formatting.

Here are some examples of input and expected output:

Game:

0. 1. e4

1. 1... c6

... (остальные ходы первого примера) ...

38. 19... Rxa8

Correct Highlight Output:

[47, 53]

Game:

0. 1. e4

1. 1... c5

... (остальные ходы второго примера) ...

39. 20. Kd2

Correct Highlight Output:

[52, 56]

... (еще 27 аналогичных примеров) ...

Now, analyze the following game:

0. 1. e4

1. 1... c5

... (ходы целевой тестовой партии) ...

36. 18... Rd6+

Выбор между этими стилями представления данных и оценка их влияния на качество предсказаний LLM стали одной из ключевых задач экспериментальной части данного исследования.

2.4 Собственная ML-модель

2.4.1 Введение в задачу и обзор подхода

В задаче автоматического выделения интересных моментов в шахматной партии в первую очередь на ум приходят два основных подхода: алгоритмический, например, поиск шаблонных паттернов, анализ оценок движка, и LLM-подход, то есть обработка текстовых описаний или анализ партий с помощью языковых моделей. Однако у каждого из них есть существенные ограничения.

Рассмотрим недостатки этих подходов:

1) Алгоритмические методы, как, например, поиск тактических комбинаций, резких изменений оценки Stockfish:

   a. Пропускают не очевидные, но важные моменты, не укладывающиеся в жесткие правила;

   b. Не учитывают глубокий контекст партии;

   c. Не могут покрыть все случаи, и потому могут пропустить другие интересные моменты;

2) LLM-подходы:

   a. Требуют огромных вычислительных ресурсов для обработки партий в текстовом виде;

   b. Могут быть избыточны для задачи, так как работают с естественным языком, а не структурными данными.

Само собой напрашивается решение - использовать подходы машинного обучения. Машинное обучение позволяет комбинировать сильные стороны алгоритмических и нейросетевых методов:

1) Этот подход достаточно гибкий. Модель обучается на данных, а не на жестких правилах, и может выявлять неочевидные закономерности;

2) Он эффективный. Работа с векторными представлениями доски требует меньше ресурсов, чем LLM;

3) Наконец, ML-подход гораздо более масштабируемый, в отличие от предыдущих подходов. Модель можно дообучать на новых партиях без переписывания логики.

Теперь, когда мы выяснили, что ML-подход может быть достаточно эффективен в этой задаче, поразмышляем над тем, как решать задачу с помощью существующих архитектур и алгоритмов машинного обучения.

Придумывая разные подходы, невольно упускается важная деталь. В каком формате представить доску, чтобы можно было применять существующие решения? Ведь шахматная доска не попадает под "классические" типы данных: вещественные или категориальные. А значит, доску нужно уметь представлять в подходящем формате.

Доску можно закодировать разными способами, но самым качественным будет обучить семантические представления. Этот этап критически важен по нескольким причинам:

1) Он обеспечивает сжатие информации. Эмбеддинги превращают сложное состояние доски в низкоразмерный вектор, сохраняя ключевые особенности;

2) Такой подход добавляет универсальности в процесс обучения и построения разных моделей. Одни и те же эмбеддинги можно использовать для разных задач, например, анализ тактики, поиск ошибок, генерация комментариев;

3) Хорошая интерпретируемость результатов. Векторное пространство позволяет визуализировать схожесть позиций и паттернов.

Качественные эмбеддинги позволят далее решать задачу самыми разными способами. Например, на готовых представлениях можно:

1) Обучить классификатор или регрессор для предсказания "интересности" на основе динамики эмбеддингов;

2) Использовать модели, такие как LSTM или Transformer, для анализа последовательностей ходов;

3) Комбинировать эмбеддинги с дополнительными признаками, например, оценка движка, метаданные партии - рейтинг игроков, время на обдумывание хода и так далее.

Наконец, рассмотрим преимущества ML-подхода в целом.

1) Автоматизация: Минимизация ручного определения правил.

2) Адаптивность: Модель можно дообучать под конкретные критерии "интересности", например, для учебного анализа или зрелищных трансляций.

3) Баланс точности и скорости: Эмбеддинги позволяют добиться качества, близкого к LLM, при значительно меньших затратах.

Таким образом, предложенный подход устраняет недостатки чисто алгоритмических и LLM-методов, обеспечивая гибкий и эффективный инструмент для анализа шахматных партий.

2.4.2 Предобработка данных

Для обучения модели, способной выявлять интересные моменты в шахматных партиях, мы опираемся на данные с платформы Lichess. Основной источник — это обширные датасеты сыгранных партий, которые используются для формирования эмбеддингов шахматной доски, а также задачи с Lichess, служащие разметкой для выделения значимых эпизодов. Такой подход позволяет автоматизировать процесс подготовки данных, хотя и не лишён определённых ограничений.

Работа начинается с использования шахматных задач, таких как тактические комбинации, которые извлекаются из реальных партий. Каждая задача сопровождается информацией о партии, из которой она взята. Мы помечаем ходы, связанные с задачей, как «интересные», а остальные части партии считаем «неинтересными». Это даёт возможность создать разметку в автоматическом режиме, что крайне важно, учитывая объём данных. Ручное аннотирование тысяч партий было бы слишком трудоёмким и непрактичным, поэтому текущий метод — это компромисс, позволяющий масштабировать процесс. Однако у подхода есть свои слабые стороны. Например, задачи на Lichess чаще всего акцентируют внимание на ярких тактических ударах, но могут упускать стратегически важные или психологически напряжённые моменты, которые также могли бы быть интересны для анализа. Кроме того, разметка иногда страдает от неточности: в задаче выделяется один ключевой момент, хотя в партии может быть несколько значимых эпизодов. В будущем модель можно дообучить на более точных данных, чтобы преодолеть эти ограничения.

Для хранения партий мы выбрали CSV-формат, который удобен для первичной обработки. В файлах фиксируются рейтинг игроков (чтобы учитывать уровень игры),

последовательность ходов в стандартной нотации (UCI) и отрезок партии, содержащий интересный момент с указанием начального и конечного хода. CSV хорош тем, что его легко фильтровать и преобразовывать в бинарные форматы, которые ускоряют обучение модели. Однако для финальной предобработки мы переходим на бинарные массивы в формате .npy, которые обеспечивают быструю загрузку данных в память при работе с фреймворками глубокого обучения. Каждая партия сохраняется в отдельном файле, а имена файлов содержат хэш партии, чтобы исключить дублирование. Структура хранения организована так, чтобы в одной папке находились десятки тысяч файлов, что упрощает управление данными. Мы предпочли .npy базам данных, так как этот формат минимизирует накладные расходы и оптимизирован для пакетной загрузки, что особенно важно на этапе обучения.

Чтобы датасет был качественным, мы проводим предварительную фильтрацию. Например, исключаются партии короче 15 ходов, поскольку они редко содержат содержательные моменты. Также мы отсеиваем партии с большим количеством грубых ошибок, где оценка позиции резко меняется несколько раз подряд из-за просмотров игроков. Это помогает сосредоточиться на играх, которые имеют реальную аналитическую ценность.

Одним из ключевых вопросов при подготовке данных было кодирование шахматной позиции. Мы рассмотрели несколько вариантов. Первый — тензор размером 8×8×12, где 12 каналов соответствуют каждому типу фигур и их цвету. Второй вариант — тензор 8×8×15, который включает дополнительные признаки, такие как флаги рокировки, взятия на проходе или контроль определённых полей. Были и более простые подходы: матрица 8×8, где фигуры кодируются числами (например, 1 для пешки, 2 для коня и так далее), или плоский вектор, представляющий доску в линейном виде. После анализа мы остановились на формате 8×8×12. Он обеспечивает хороший баланс между информативностью и простотой: 12 каналов достаточно для описания всех фигур, а формат хорошо подходит для свёрточных нейронных сетей, которые мы используем в модели. Дополнительные признаки, такие как рокировки, можно включить позже как метаданные, чтобы не усложнять архитектуру на начальном этапе.

Наша цель — создать масштабируемый и эффективный датасет, который позволит модели выявлять интересные моменты в шахматных партиях, даже несмотря на ограничения автоматической разметки. Выбранные методы хранения и обработки данных обеспечивают высокую скорость экспериментов и гибкость. Формат .npy ускоряет загрузку, CSV упрощает предварительную обработку, а фильтрация и кодирование позиций делают датасет пригодным для обучения сложных моделей. В дальнейшем мы планируем улучшать разметку и добавлять более разнообразные данные, чтобы повысить качество анализа.

2.4.3 Обучение эмбеддингов доски

Для создания векторных представлений шахматных позиций, которые можно использовать в анализе партий или автоматическом выявлении интересных моментов, мы рассматривали несколько подходов. Основная цель — получить эмбеддинги, которые не просто кодируют состояние доски, но и отражают стратегические и тактические особенности позиции, чтобы их можно было применять в задачах анализа и прогнозирования. Рассмотрим, как можно подойти к этой задаче, какие трудности возникают и как их можно преодолеть.

Один из первых методов, который приходит на ум, — использование автоэнкодеров. Идея проста: сжать состояние шахматной доски в компактное векторное представление (скрытое пространство), а затем восстановить его. Такой подход кажется логичным, поскольку автоэнкодеры хорошо справляются с задачами сжатия данных. Однако на практике они показали себя не лучшим выбором. Во-первых, автоэнкодеры фокусируются на точном воспроизведении входных данных, а не на выделении семантически значимых признаков. Например, модель может идеально восстанавливать расположение фигур, но не улавливать, скажем, тактические мотивы или стратегическую ценность позиции. Во-вторых, такие эмбеддинги чувствительны к мелким изменениям на доске: перестановка двух пешек может сильно изменить вектор, хотя с шахматной точки зрения позиция осталась почти идентичной. Наконец, обучение автоэнкодеров требует значительных вычислительных ресурсов, особенно если цель — не реконструкция доски, а создание осмысленных представлений. Эти недостатки заставили нас искать более подходящие методы.

Другой подход заимствует идеи из обработки естественного языка, а именно модель Word2Vec с использованием skip-грамм. Здесь шахматные позиции рассматриваются как "слова", а последовательности ходов в партии — как "предложения". Модель обучается предсказывать соседние позиции в партии, то есть для данной позиции она пытается угадать, какие позиции были до или после неё. Это позволяет уловить динамику партии и создать эмбеддинги, в которых близкие по смыслу позиции (например, похожие тактические или дебютные структуры) оказываются рядом в векторном пространстве. Такой метод привлекателен своей интерпретируемостью и способностью учитывать контекст игры. Для реализации мы планируем использовать библиотеку gensim или создать кастомную модель на PyTorch. Размерность эмбеддингов будет варьироваться от 64 до 256 — точное значение определим экспериментально. Чтобы охватить всё разнообразие шахматных позиций, обучение будет проводиться на миллионах партий из открытых баз, таких как Lichess.

Ещё один перспективный подход — обучение на вспомогательной задаче. Вместо того чтобы напрямую кодировать позиции, мы можем обучить нейросеть решать задачу, связанную с шахматным анализом, например, предсказывать, насколько интересен тот или иной ход или как он влияет на оценку позиции. Для этого можно использовать сверточную нейросеть (CNN), которая принимает на вход состояние доски и выдаёт, например, бинарную метку ("интересный ход" или "неинтересный") или числовую оценку, основанную на данных движка Stockfish. После обучения последний слой нейросети убирается, а выход предпоследнего слоя используется как эмбеддинг. Такой подход хорош тем, что эмбеддинги сразу оптимизируются под конкретную задачу, например, выделение ключевых моментов в партии. Однако для этого требуется качественная разметка данных, что может быть проблемой. Если разметки достаточно, мы можем обучить классификатор на бинарных метках, используя, например, задачи с Lichess. Если данных мало, можно применить регрессионный подход, предсказывая изменение оценки позиции по Stockfish, что позволит обойтись без ручной разметки.

Чтобы убедиться, что полученные эмбеддинги действительно полезны, необходимо тщательно проверить их качество. Один из способов — визуализация с помощью методов снижения размерности, таких как t-SNE или UMAP. Это позволяет увидеть, группируются ли похожие позиции (например, тактические шаблоны или эндшпили) в кластеры. Если визуализация показывает чёткие группы, это хороший знак, что эмбеддинги улавливают важные закономерности.

Другой метод проверки — задача поиска ближайших соседей (k-NN). Для заданной позиции мы находим ближайшие к ней в векторном пространстве и проверяем, имеют ли они схожий шахматный смысл. Например, если позиция с вилкой коня оказывается близко к другим позициям с аналогичными тактическими мотивами, это говорит о качестве эмбеддингов.

Также мы можем протестировать эмбеддинги в задачах transfer learning. Например, использовать их для предсказания исхода партии или классификации позиций по сложности. Если эмбеддинги хорошо работают в таких задачах, это подтверждает, что они содержат полезные признаки. Ещё один интересный тест — анализ аналогий, по аналогии с Word2Vec в NLP. Например, можно проверить, выполняется ли в векторном пространстве соотношение вроде "позиция с королём на g1 для белых эквивалентна позиции с королём на g8 для чёрных". Это помогает понять, насколько эмбеддинги улавливают шахматные закономерности.

После анализа всех подходов мы склоняемся к двум основным методам: Word2Vec с использованием skip-грамм и обучение на вспомогательной задаче. Первый метод хорош тем, что учитывает контекст партии и не требует разметки, а второй позволяет создавать эмбеддинги, сразу ориентированные на конкретные задачи, такие как поиск интересных ходов. Автоэнкодеры, несмотря на их интуитивную привлекательность, оказались менее подходящими из-за фокуса на реконструкции, а не на семантике, и их вычислительной сложности.

В дальнейшем мы сосредоточимся на реализации этих двух подходов, экспериментируя с размерностью эмбеддингов и архитектурами нейросетей. Качество эмбеддингов будем проверять через визуализацию, задачу ближайших соседей и transfer

learning. Это позволит убедиться, что представления действительно полезны для анализа шахматных позиций и могут быть применены в более сложных моделях, например, для автоматического комментирования партий или поиска тактических возможностей.

2.4.4 Построение модели предсказания "интересности"

Для анализа "интересности" шахматных ходов предлагается начать с простого подхода, который затем можно усложнять по мере необходимости. На первом этапе используется классификация ходов на интересные и неинтересные, основанная на эмбеддингах текущей позиции на доске. Эмбеддинги — это числовые представления состояния игры, которые могут варьироваться по размерности, например, от 64 до 256. Чтобы учесть динамику партии, можно дополнительно включить контекст, добавляя эмбеддинги нескольких предыдущих и последующих ходов. Число таких ходов (гиперпараметр k, например, 3) подбирается экспериментально. Для классификации подойдёт простая модель, такая как полносвязная нейронная сеть или градиентный бустинг. Если контекст используется, эмбеддинги можно либо усреднить, либо объединить в один вектор. Такой подход позволяет быстро оценить, насколько информативны сами эмбеддинги, без необходимости усложнять архитектуру. Кроме того, добавление контекста помогает проверить гипотезу, что интересность хода зависит не только от текущей позиции, но и от общей динамики игры.

Для анализа долгосрочных зависимостей в партии лучше использовать модели, способные обрабатывать последовательности ходов. Например, рекуррентные нейронные сети, такие как LSTM или GRU, могут выявлять отложенные последствия ходов, например, подготовку к тактической комбинации. Альтернативой служат Transformer-модели, которые благодаря механизму внимания эффективно находят связи между ходами, даже если они разделены десятками ходов. Входные данные для таких моделей — последовательность эмбеддингов ходов, ограниченная, например, 200 ходами, чтобы избежать чрезмерной вычислительной нагрузки. На выходе модель может выдавать либо бинарную метку "интересности" для каждого хода, либо вещественное значение, отражающее степень интересности, либо даже определять

интервалы, где происходят ключевые моменты партии. Такой подход позволяет уловить сложные паттерны, например, развитие тактической идеи на протяжении нескольких ходов или создание психологического давления через серию форсированных ходов.

После получения предсказаний модели их можно дополнительно обработать, чтобы выделить наиболее значимые моменты. Например, если модель выдаёт вещественные значения "интересности", можно использовать скользящее окно для поиска резких изменений, которые указывают на ключевые эпизоды, такие как рост значения на определённый процент за несколько ходов. Также возможно группировать последовательные ходы с высокой "интересностью" в единые эпизоды, применяя методы кластеризации. Чтобы избежать ложных срабатываний, кратковременные всплески значений можно отфильтровать как шум. Такой подход делает анализ гибким: он позволяет выделять разные типы интересных моментов — тактические, стратегические или даже ошибки игроков. Кроме того, его легко адаптировать под новые критерии, например, для оценки "зрелищности" партий в трансляциях.

Для повышения качества предсказаний можно дополнить эмбеддинги дополнительными признаками. Например, резкие изменения в оценке шахматного движка Stockfish могут сигнализировать об интересных моментах. Также полезны метаданные партии: рейтинг игроков (ошибки гроссмейстеров часто интереснее, чем ошибки любителей) или турнирный контекст (финальные партии обычно более напряжённые, чем тренировочные). Эти признаки можно объединить с эмбеддингами перед подачей в модель, что сделает предсказания более точными.

В итоге предложенный подход позволяет начать с простого базового решения, постепенно усложняя его за счёт временных моделей, таких как LSTM или Transformer, и гибкой постобработки результатов. Основное внимание уделяется балансу между интерпретируемостью (через использование эмбеддингов) и точностью (за счёт учёта динамики партии). Это делает метод универсальным и пригодным для разных задач анализа шахматных партий.

2.4.5 Обучение и валидация

Для корректной оценки модели данные делятся на три независимых набора: обучающую, валидационную и тестовую выборки. Основная часть данных — около 90–95% — идёт на обучение, чтобы модель могла хорошо изучить закономерности. Валидационная выборка, напротив, небольшая, не больше тысячи партий, чтобы быстро проверять, не переобучается ли модель. Тестовый набор полностью изолирован: он не участвует ни в обучении, ни в валидации, обеспечивая объективную оценку.

Чтобы данные были организованы удобно, каждая партия сохраняется в отдельном .npy-файле и распределяется по соответствующим папкам: train, val или test. Важно, чтобы партии не дублировались между выборками — для этого используется проверка по хешу. Если партия разбита на сегменты, например по 10 ходов, все сегменты одной партии остаются в одной выборке, чтобы сохранить контекст.

Обучение организовано с учётом большого объёма данных. Чтобы модель не проходила через все стадии — от недообучения до переобучения — за одну эпоху, в каждой эпохе используется фиксированное число партий, например 50 тысяч. Это упрощает контроль за процессом: легче следить за логами и применять раннюю остановку, если нужно. Кроме того, каждая новая эпоха берёт случайную подвыборку из обучающего набора, что делает модель более устойчивой. Валидационная выборка намеренно будет ограничена — это ускоряет оценку после каждой эпохи и снижает риск "подгонки" модели под валидационные данные.

Для оценки качества модели используются несколько метрик. Основное внимание уделяется классическим показателям: точности (precision) и полноте (recall), которые должны быть выше 0.8, чтобы минимизировать ложные срабатывания и пропуски интересных моментов. F1-score помогает сбалансировать эти два показателя, а AUC-ROC (тоже выше 0.8) показывает, насколько хорошо модель различает интересные и неинтересные моменты, а также преимущество AUC-ROC заключается в том, что оно устойчиво к дисбалансу классов, что как раз наблюдается в нашей задачей: на целую партию может даже не прийтись ни одного интересного момента. Если за-

дача предполагает выделение интервалов, например последовательностей ходов, используется метрика IoU (Intersection over Union). Ещё один важный параметр — время инференса, особенно если модель будет интегрироваться в реальные системы. Выбор метрик обусловлен тем, что ложные срабатывания (например, пометка скучных ходов как интересных) и пропуски одинаково нежелательны, а AUC-ROC даёт общую картину способности модели к ранжированию.

Сравнение модели с другими подходами — важная часть работы. Один из вариантов — алгоритмический анализ, например, использование движка Stockfish для выявления резких изменений оценки позиции. Для объективности сравнение проводится на одном и том же тестовом наборе, а результаты оцениваются по точности, полноте и субъективным впечатлениям экспертов. Другой подход — использование больших языковых моделей, таких как GPT-4, с задачей найти интересные моменты в партиях, поданных в текстовом формате PGN. Здесь оценивается согласованность с человеческой разметкой, а также время обработки и стоимость запросов. Важно понимать, что языковые модели и ML-подход работают по-разному: первые опираются на текстовые шаблоны, а вторые — на статистические закономерности в данных.

Стоит отметить, что "интересность" — понятие субъективное, и это влияет на весь процесс. Даже эксперты порой расходятся в оценке одних и тех же моментов, поэтому разметка данных не может быть идеальной. Языковые модели часто ориентируются на яркие шаблоны, вроде жертвы ферзя, в то время как ML-модель выявляет статистические аномалии, такие как редкие позиции или резкие изменения оценки. Низкие метрики, например AUC на уровне 0.6, не всегда означают провал: если модель всё равно выделяет моменты, полезные для анализа, она уже приносит ценность. Субъективность задачи требует гибкости: важнее улучшение относительно базового уровня, чем погоня за идеальными цифрами.

В итоге процесс строится на строгом разделении данных, чтобы избежать утечек, и контролируемом размере эпох для стабильного обучения. Оценка проводится комплексно: через метрики и сравнение с альтернативными подходами. Субъективность задачи требует не только внимания к числовым показателям, но и гибкой интерпретации результатов.

2.4.6 Оптимизация и интерпретация

Наша цель — создать модель, которая не только демонстрирует высокую производительность, но и остаётся понятной и интерпретируемой. Мы не стремимся сразу предложить готовые решения, а скорее намечаем путь, который позволит нам итеративно улучшать систему, опираясь на эксперименты, анализ и здравый смысл. Вот основные направления, которые мы планируем развивать.

Для начала мы сосредоточимся на подборе гиперпараметров, чтобы добиться максимальной эффективности модели. Вместо того чтобы перебирать все возможные комбинации вслепую, мы подойдём к задаче системно. Сначала используем методы вроде GridSearch или RandomSearch для грубой настройки базовых параметров, таких как скорость обучения (от 1e-5 до 1e-3 по логарифмической шкале), размер батча (32, 64, 128 или 256) и размерность эмбеддингов (64, 128 или 256). Это позволит нам быстро определить перспективные диапазоны. Затем, для более тонкой настройки, например, коэффициентов регуляризации, пропорций dropout или температуры softmax, мы планируем применить байесовскую оптимизацию, которая эффективнее работает с узкими диапазонами.

Для разных компонентов модели параметры будут подбираться с учётом их специфики. Например, при обучении представлений мы будем экспериментировать с размером окна контекста (от 3 до 10 ходов), количеством negative samples для skip-gram и размером скрытого слоя автоэнкодера. Для модели предсказания важными будут глубина свёрточной нейросети (от 3 до 7 слоёв), количество голов внимания в Transformer и размер скрытого состояния LSTM. Чтобы не тратить ресурсы впустую, валидацию будем проводить поэтапно: начнём с широкого поиска, затем сосредоточимся на наиболее перспективных областях, используя early stopping для экономии времени и вычислений.

Мы хотим не только построить модель, которая хорошо работает, но и понять, почему она принимает те или иные решения. Для этого планируем многоуровневый подход к анализу. Начнём с эмбеддингов: с помощью методов визуализации, таких как t-SNE или UMAP, мы посмотрим, как они организованы в пространстве. Это поз-

волит выявить семантические кластеры, например, тактические паттерны или позиции в эндшпилях, и сравнить их с тем, как эксперты классифицируют шахматные ситуации. Ещё одна идея — исследовать семантические аналогии, вроде "король на g1 для белых — это как король на g8 для чёрных" или "размен ферзя похож на размен ладьи, но с большим весом". Это поможет понять, насколько модель улавливает логику шахмат.

Для анализа архитектуры мы планируем разные подходы в зависимости от типа модели. Если это свёрточная нейросеть, мы визуализируем фильтры первого слоя, построим активационные карты для ключевых позиций и используем Grad-CAM, чтобы определить, какие области доски модель считает наиболее важными. Для временных моделей, таких как Transformer или LSTM, мы проанализируем веса механизма внимания или долговременные зависимости. А для классификатора применим методы SHAP и LIME, чтобы понять, какие признаки — позиционные или контекстные — вносят наибольший вклад в предсказания.

Переобучение — одна из главных проблем, с которой мы можем столкнуться, поэтому мы заранее продумываем меры контроля. Регулярно будем сравнивать метрики на обучающей, валидационной и тестовой выборках, обращая внимание на случаи, где расхождения максимальны. Чтобы выявить возможные утечки данных, планируем использовать adversarial validation: обучим классификатор, который попробует отличить обучающую выборку от тестовой. Если его AUC окажется выше 0.7, это будет сигналом, что данные требуют дополнительной проверки. Также будем следить за распределением предсказаний и признаков, чтобы вовремя заметить их дрейф.

Мы рассматриваем разработку как итеративный процесс, где каждый цикл помогает модели становиться лучше. Сначала анализируем ошибки, чтобы выявить слабые места — например, если модель плохо справляется с эндшпилями, мы предположим, что в обучающей выборке недостаточно таких примеров. Затем формулируем гипотезу, например, увеличить вес эндшпилей в функции потерь, проводим целевые эксперименты и проверяем результаты на специализированных тестах. Такой подход позволяет не просто вносить изменения, а делать это осознанно, с пониманием их влияния.

Чтобы не потерять важные детали и сделать процесс воспроизводимым, мы будем тщательно документировать все шаги. Для отслеживания экспериментов используем системы вроде MLflow или Weights&Biases, где фиксируем параметры, метрики и результаты. Код будет сопровождаться подробными комментариями, а архитектура модели — визуальной документацией. Также создадим чеклисты, чтобы любой член команды мог повторить эксперимент с нуля. Это не только облегчит работу, но и поможет глубже понять, как модель принимает решения, что критично для доверия к системе.

2.4.7 Интеграция в продукт

Наша цель — создать удобный и масштабируемый сервис на базе ML-модели, который легко интегрируется в продукт и поддерживает дальнейшее развитие. Мы хотим, чтобы он был понятным для команды, надёжным в работе и гибким для будущих доработок. Вот как мы планируем подойти к реализации.

Сначала мы решили выделить ML-модель в отдельный микросервис. Это позволит изолировать её от основного приложения, упростить обновления и масштабирование. Для разработки выберем FastAPI — он быстрый, удобный и хорошо подходит для создания API. Чтобы сервис был легко развертываемым, мы по возможности упакуем его в Docker-контейнеры. Это даст нам гибкость при деплое и упростит управление зависимостями. Для ускорения обработки запросов планируем использовать Redis для кэширования результатов, чтобы не пересчитывать одинаковые данные многократно.

API сервиса будет простым и понятным. Мы хотим реализовать два основных эндпоинта. Первый — для анализа одиночных запросов, где пользователь отправляет данные и получает результат. Второй — для пакетной обработки, чтобы можно было отправить сразу несколько наборов данных и обработать их за один раз. Это удобно, если, например, нужно проанализировать большой объём данных.

Чтобы сервис был лёгким и не перегружал систему, мы минимизируем зависимости. Используем только необходимые библиотеки для ML, такие как PyTorch, и избежим тяжёлых фреймворков, которые могут замедлить работу. Для оптимизации планируем кэшировать результаты обработки: входные данные будут кешироваться,

а результаты — сохраняться в Redis на сутки. Это позволит сократить время ответа при повторных запросах. Также мы хотим настроить подробное логирование: фиксировать время обработки запросов и любые ошибки, чтобы можно было быстро разобраться в причинах сбоев.

Масштабируемость — ещё один важный аспект. Мы хотим, чтобы сервис мог справляться с ростом нагрузки. Для этого заложим возможность горизонтального масштабирования через Docker: при необходимости можно будет просто добавить новые экземпляры сервиса. Также мы предусмотрим механизмы автоматического восстановления после сбоев и балансировку нагрузки, чтобы пользователи не замечали проблем даже при пиковых нагрузках.

Такой подход, как нам кажется, позволит создать гибкий и понятный сервис, который будет легко интегрировать в продукт и развивать в будущем.

2.5 Обработка видеозаписей шахматных партий

2.5.1 Введение в проблематику и задачу

Основная задача, поставленная перед разработкой данного IT-решения, заключалась в создании высокоавтоматизированного и точного инструмента для генерации видеофрагментов шахматных партий из полных видеозаписей турниров или отдельных игр. Конкретизация этой задачи сводится к необходимости программного выделения (вырезки) из непрерывного видеопотока определенной последовательности ходов, заданной пользователем, например, с хода N по ход M включительно. Фундаментом для этой операции служат два разнородных, но взаимосвязанных источника данных:

1) Видеозапись партии: Цифровой видеофайл, фиксирующий визуальное представление событий на шахматной доске и вокруг нее. Этот источник содержит всю необходимую визуальную информацию, но сам по себе не структурирован с точки зрения шахматной логики. Формат и качество видео могут значительно варьироваться в зависимости от источника записи (профессиональная трансляция, любительская съемка, запись с веб-камеры и т.д.);

2) Файл PGN (Portable Game Notation): Текстовый файл в стандартизированном формате, содержащий полную нотацию шахматной партии (символьную запись

ходов: 1. e4 e5 2. Nf3 Nc6...). Кроме самой последовательности ходов, PGN-файлы, особенно с современных турниров, использующих электронные доски (например, DGT), часто содержат критически важные для данной задачи метаданные – встроенные временные метки в формате комментариев [%ts timestamp]. Эти метки представляют собой Unix timestamp (количество миллисекунд, прошедших с 1 января 1970 года UTC), точно указывающие абсолютное астрономическое время, когда система зафиксировала совершение соответствующего хода. Наличие этих меток является ключевым фактором, позволяющим потенциально достичь высокой точности синхронизации.

Центральной проблемой, требующей нетривиального решения, является синхронизация данных из двух принципиально разных доменов: событийного (дискретные ходы с абсолютными временными метками из PGN) и потокового (непрерывное видео с относительной временной шкалой, начинающейся с нуля в момент старта файла). Необходимо установить точное соответствие между символьной записью хода и его временной меткой из PGN с конкретным моментом времени (кадром или временной отметкой в секундах/миллисекундах) в видеофайле. Сложность этой задачи усугубляется рядом практических факторов:

Неравномерность времени ходов: шахматы – игра с переменным темпом. Время, затрачиваемое игроками на обдумывание каждого хода, может флуктуировать в огромных пределах: от долей секунды в блиц-партиях или цейтнотах до десятков минут, а то и более часа, в классических партиях с длительным контролем времени. Это означает, что интересующие нас события (ходы) распределены по временной оси видео крайне неравномерно. Простой линейный расчет времени на ход неприменим.Визуальный "шум" и посторонние действия: Видеозапись фиксирует не только целевые события (перемещение фигур с поля на поле), но и огромное количество сопутствующих и нерелевантных действий. Сюда относятся: движения рук игроков над доской до и после хода, поправление фигур на полях, касание и нажатие на шахматные часы, жестикуляция игроков, их перемещения в кадре (например, чтобы встать и пройтись), появление в кадре судей, зрителей, изменения освещения (естественного или искусственного), блики на доске и фигурах. Все эти визуальные изменения могут

быть ошибочно интерпретированы алгоритмами анализа изображений как значимые события. Технические особенности и артефакты видеозаписи: качество и характеристики видеозаписи могут вносить дополнительные сложности. Возможны пропуски кадров (frame drops) при записи из-за недостаточной производительности системы или проблем с передачей данных. Частота кадров может быть нестабильной (Variable Frame Rate - VFR), что усложняет прямое сопоставление номера кадра и времени. Артефакты сжатия видео (особенно при низком битрейте) могут искажать изображение, затрудняя точное распознавание объектов или изменений. Все это может повлиять на точность определения времени событий в видеопотоке.

Следовательно, для успешного решения поставленной задачи требовалось разработать алгоритм, способный надежно идентифицировать временную точку в видео, соответствующую как минимум одному известному событию из PGN (например, первому ходу партии или любому другому ходу с известной меткой [%ts]). Эта идентифицированная точка могла бы служить "якорем" или референтной точкой, относительно которой можно было бы рассчитать временные позиции всех остальных ходов, используя их относительное время из PGN.

В процессе исследования и разработки было предложено, реализовано и протестировано несколько различных подходов. Эволюция алгоритма отражает путь от первоначальной, интуитивно привлекательной, но, как показала практика, крайне ненадежной идеи использования сложного анализа изображений (компьютерного зрения) для автоматического поиска событий в видео, к более прагматичным, детерминированным и надежным методам, опирающимся на точные метаданные, предоставляемые внешними системами (временные метки в PGN и в имени файла). Данный раздел подробно описывает каждую из ключевых итераций этой эволюции ("Версии" алгоритма), включая их концептуальные основы, детали реализации, используемые технологии и библиотеки, выявленные в ходе тестирования преимущества и, что более важно, критические недостатки, которые послужили причиной для перехода к следующей, усовершенствованной версии.

Разработка такого инструмента продиктована растущей потребностью в автоматизации обработки шахматного видеоконтента. Возможные области применения включают:

Создание динамичных хайлайтов: автоматическая нарезка ключевых моментов (например, тактических ударов, ошибок, интересных эндшпилей) для трансляций, социальных сетей, новостных порталов. Подготовка учебных материалов: фокусировка на определенных фазах партии (дебют, миттельшпиль, эндшпиль) или на конкретных критических позициях для разбора с учениками. Ускоренный анализ партий: предоставление тренерам и игрокам возможности быстро просмотреть видеофрагменты, соответствующие интересующим их ходам, без необходимости вручную искать их в полной записи. Архивирование: сохранение только наиболее значимых фрагментов длительных турнирных записей для экономии места и упрощения доступа к ключевым моментам.

2.5.2 "Версия 1": Поиск первого хода с помощью компьютерного зрения

Первоначальный подход базировался на амбициозной гипотезе: возможно создать полностью автономную систему, которой для работы требуются только два входных файла – видеозапись партии и соответствующий PGN-файл. Предполагалось, что система сможет самостоятельно, без каких-либо дополнительных метаданных (например, о точном времени начала видеозаписи или специальном формате имен файлов), определить момент совершения первого хода непосредственно путем анализа видеоряда. Идея заключалась в том, чтобы использовать арсенал методов компьютерного зрения (Computer Vision, CV) для "просмотра" видео, обнаружения характерных визуальных изменений, соответствующих первому ходу, и фиксации времени этого события (first_move_time) относительно начала видеофайла. Это найденное время затем использовалось бы как точка отсчета ("якорь") для расчета временных меток всех последующих ходов, временные интервалы до которых брались бы из PGN-файла (относительно первого хода в PGN). Привлекательность такого подхода заключалась в его кажущейся универсальности и отсутствии жестких требований к формату входных данных, кроме наличия самого видео и PGN.

2.5.2.1 Метод работы (детальное описание):

Алгоритм, реализованный в "Версии 1", представлял собой многоступенчатый процесс анализа видео, направленный на идентификацию момента первого хода:

1) Чтение и инициализация видео

Видеофайл, указанный через video_path, открывался с использованием класса cv2.VideoCapture из библиотеки OpenCV. Этот класс предоставляет унифицированный интерфейс для по кадрового чтения видео из различных файловых контейнеров (MP4, AVI, MOV и др.) и с использованием разных кодеков. Получались основные метаданные видео, такие как частота кадров (fps = cap.get(cv2.CAP_PROP_FPS)), которая, однако, могла быть неточной или переменной (VFR). Более важным для определения времени кадра являлось свойство cv2.CAP_PROP_POS_MSEC, возвращающее временную метку текущего кадра в миллисекундах от начала файла. Корректность этого значения зависит от точности временных меток, закодированных в самом видеофайле при его создании. Инициализировались переменные для хранения предыдущего кадра (prev_frame), предыдущего состояния доски (prev_board_gray), времени последнего обнаруженного изменения (last_change_time) и счетчика стабильных кадров (stable_frames_count).

2) Стратегия пропуска кадров (Frame Skipping)

Анализ каждого кадра видео методами CV является крайне ресурсоемкой операцией. Для ускорения процесса поиска первого значимого события (хода) была внедрена стратегия пропуска кадров. Изначально алгоритм анализировал только каждый frame_skip_initial кадр (например, каждый 10-й или 15-й). Это позволяло быстро "промотать" начальные статичные участки видео (например, до начала игры). Как только обнаруживалось первое подозрительное изменение (потенциальный ход), которое могло быть началом реального движения, шаг анализа динамически уменьшался до frame_skip_fine (например, каждый 2-й или каждый кадр). Цель этого двухфазного подхода – быстро локализовать временную область, где предположительно происходит событие, а затем уточнить момент его начала и конца с большей временной разрешающей способностью, минимизируя при этом общие вычислительные затраты.

Однако, даже при frame_skip_fine = 1, реальная точность ограничена частотой кадров видео (например, при 30 fps минимальный интервал между кадрами ~33 мс).

3) Обнаружение шахматной доски (detect_board_by_edges)

Этот этап был необходим для локализации области интереса (Region of Interest, ROI), чтобы последующий анализ изменений производился только в пределах доски, игнорируя изменения на фоне. Преобразование в оттенки серого: cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) конвертировало цветной кадр (BGR в OpenCV) в одноканальное серое изображение. Это стандартный шаг для упрощения изображения и снижения вычислительной нагрузки, так как многие алгоритмы анализа формы и краев работают с интенсивностью, а не цветом. Адаптивное выравнивание гистограммы (CLAHE): cv2.createCLAHE(...) и clahe.apply(gray). Применялось опционально, если обнаруживалась низкая контрастность изображения (например, по стандартному отклонению яркости np.std(gray)). CLAHE (Contrast Limited Adaptive Histogram Equalization) улучшает локальный контраст, обрабатывая изображение по небольшим плиткам (tileGridSize), что помогает выявить слабо видимые края доски в условиях неравномерного освещения, часто встречающегося при съемке партий. Параметр clipLimit предотвращает чрезмерное усиление шума. Размытие по Гауссу: cv2.GaussianBlur(gray, (3, 3), 0) применяло гауссовский фильтр с небольшим ядром 3x3 для подавления мелкого шума (например, зернистость сенсора, артефакты сжатия) перед детекцией краев. Шум мог привести к фрагментации контуров. Размер ядра определяет степень сглаживания. Детекция краев Кэнни: cv2.Canny(blurred, 50, 150) выполнял многоэтапный алгоритм для нахождения резких перепадов интенсивности (краев). Он включает вычисление градиентов, подавление немаксимумов и гистерезисную пороговую фильтрацию с двумя порогами (threshold1=50, threshold2=150). Пиксели с градиентом выше threshold2 считаются надежными краями; пиксели между порогами включаются, только если они связаны с надежными краями. Этот метод хорошо подходит для обнаружения четких геометрических линий доски. Поиск контуров: cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) находил замкнутые контуры на бинарном изображении краев. cv2.RETR_EXTERNAL извлекал только внешние контуры, что помогало

изолировать контур доски от контуров внутри нее (например, от фигур). cv2.CHAIN_APPROX_SIMPLE экономил память, убирая избыточные точки на прямых линиях контура. Аппроксимация и фильтрация контуров: cv2.approxPolyDP(contour, 0.03 * cv2.arcLength(contour, True), True) аппроксимировал найденные контуры многоугольниками с помощью алгоритма Дугласа-Пекера. Параметр epsilon (здесь 3% от периметра контура) контролировал степень упрощения. Затем применялись фильтры: len(approx) == 4 (ищем четырехугольники) и 60000 < cv2.contourArea(approx) < 300000 (фильтр по площади, чтобы отсеять другие четырехугольные объекты – слишком маленькие или слишком большие). Диапазон площади подбирался эмпирически под конкретные условия съемки (разрешение, ракурс). Если доска не находилась, кадр пропускался.

4) Выделение и нормализация области доски (extract_board_region)

После нахождения контура доски, изображение внутри него вырезалось и трансформировалось для унификации и устранения искажений. Перспективное преобразование (Perspective Transform): Шахматная доска в кадре почти всегда видна под углом, что приводит к перспективным искажениям (квадратные клетки выглядят как трапеции). Чтобы сравнивать состояние доски между кадрами независимо от небольших изменений ракурса и для упрощения дальнейшего анализа, применялось перспективное преобразование. Координаты вершин найденного контура (pts = np.array([p[0] for p in board_contour], dtype="float32")) сопоставлялись с углами идеального квадрата (dst_pts = np.array([[0, 0], [size-1, 0], [size-1, size-1], [0, size-1]], dtype="float32")) размером size x size (например, 400x400 пикселей). Матрица преобразования M = cv2.getPerspectiveTransform(pts, dst_pts) вычислялась на основе этих четырех пар точек. Применение преобразования (Warping): board = cv2.warpPerspective(frame, M, (size, size)) применяло матрицу M к исходному кадру frame, создавая новое изображение board, где доска "выпрямлена" и имеет стандартный размер size x size. Обрезка краев (Cropping Margins): board = board[crop_margin:-crop_margin, crop_margin:-crop_margin] удаляло узкую полосу пикселей (например, crop_margin = int(size * 0.05)) с каждой из четырех сторон нормализованного изображения доски. Это делалось для устранения возможных артефактов на краях после

warpPerspective и для того, чтобы гарантированно анализировать только область игровых полей, исключая рамку доски или небольшие захваченные участки фона.

5) Обнаружение изменений на доске (detect_changes_only_in_board)

Этот этап сравнивал нормализованное изображение доски из текущего кадра с таким же изображением из предыдущего проанализированного кадра. Сравнение нормализованных изображений: Сравнение board и prev_board делало алгоритм инвариантным к положению доски в исходном кадре и фокусировало анализ исключительно на игровой зоне. Предобработка перед сравнением: Оба изображения (board, prev_board) снова переводились в оттенки серого и размывались Гауссом, но уже с большим ядром (например, (5, 5) или (7, 7)). Цель этого усиленного размытия – снизить чувствительность алгоритма к незначительным флуктуациям освещения, теням от перемещаемых фигур, бликам, которые могли появиться или исчезнуть между кадрами, но не являются непосредственно перемещением фигуры с поля на поле. Вычисление абсолютной разницы: diff = cv2.absdiff(board_gray, prev_board_gray) вычисляло поэлементную абсолютную разницу интенсивностей пикселей между текущим и предыдущим нормализованными изображениями доски. Большие значения в diff соответствовали областям значительных изменений. Пороговая бинаризация: thresh = cv2.threshold(diff, 35, 255, cv2.THRESH_BINARY)[1] преобразовывало изображение разницы в черно-белое. Пиксели, где разница превышала пороговое значение (здесь 35 – эмпирически подобранный параметр), становились белыми (255), остальные – черными (0). Выбор порога критичен: слишком низкий – детектируется шум и мелкие изменения освещения, слишком высокий – пропускаются реальные, но малоконтрастные ходы. Морфологическое открытие (Morphological Opening): thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel) применяло операцию открытия (эрозия, за которой следует дилатация) с небольшим структурным элементом (kernel = np.ones((3,3),np.uint8)). Эта операция эффективно удаляет небольшие изолированные белые "пятна" (шум), которые могли возникнуть из-за флуктуаций пикселей или мелких артефактов, сохраняя при этом более крупные связные области изменений, соответствующие перемещению фигур. Оценка величины изменений: num_changes = np.sum(thresh == 255) подсчитывало общее количество белых пикселей (т.е. площадь

изменившихся областей) на бинарном изображении после очистки. base_threshold = board_gray.size * 0.05 вычисляло пороговое значение как 5% от общего числа пикселей на нормализованном изображении доски (т.е. 5% площади доски). Если num_changes превышало base_threshold, функция возвращала True, сигнализируя о наличии существенного изменения, предположительно хода. Использование процентного порога делало оценку более устойчивой к изменению параметра size нормализации.

6) Логика определения первого хода (detect_first_move)

Простое обнаружение изменения было недостаточным. Необходимо было идентифицировать именно первое устойчивое изменение, соответствующее игровому ходу, и отфильтровать ложные срабатывания. Поиск первого изменения: В цикле чтения кадров (с пропуском) для каждого кадра выполнялось обнаружение доски. Если доска найдена и есть предыдущий кадр prev_frame, вызывалась detect_changes_only_in_board. Если изменение обнаружено (changes_detected == True) и это первое обнаруженное изменение (last_change_time is None), то текущее время видео (current_time_sec = cap.get(cv2.CAP_PROP_POS_MSEC) / 1000) запоминалось в last_change_time, и шаг пропуска кадров уменьшался до frame_skip_fine для более детального анализа последующих кадров. Проверка стабильности после изменения: Если изменение не обнаружено (changes_detected == False) после того, как last_change_time уже был зафиксирован, это могло означать, что движение завершилось. Счетчик стабильных кадров stable_frames_count увеличивался. Если же изменение снова обнаружено (changes_detected == True), это означало, что движение еще продолжается или началось новое (например, рука убирается с фигуры). В этом случае счетчик стабильности stable_frames_count сбрасывался в 0, а last_change_time обновлялся на текущее время current_time_sec (чтобы зафиксировать момент последнего зарегистрированного движения). Фиксация момента хода: ход считался окончательно завершенным и его время (move_time) зафиксированным, если после момента last_change_time было обнаружено stability_frames (например, 2 или 3) последовательных кадров без изменений (stable_frames_count >= stability_frames). В этот момент переменной move_detected присваивалось True, move_time устанавливался равным

last_change_time (время последнего замеченного движения перед стабилизацией), и цикл поиска прерывался (break). Требование нескольких стабильных кадров помогало отсечь кратковременные визуальные помехи и дождаться момента, когда фигура поставлена на доску и рука игрока, скорее всего, уже убрана.

7) Расчет временных меток для нарезки (extract_move_timestamps)

После того как detect_first_move вернула (предположительно) время первого хода first_move_time (в секундах от начала видео), начинался расчет временных интервалов для вырезки нужных фрагментов. Парсинг PGN: PGN-файл читался с использованием библиотеки python-chess. chess.pgn.read_game(pgn_file) создавало объект игры, по которому можно было итерироваться для доступа к ходам и их комментариям. Поиск базового времени в PGN: алгоритм находил самый первый ход в главной вариации PGN, у которого был комментарий с временной меткой [%ts ...]. Из этого комментария извлекался Unix timestamp (start_time) с помощью регулярного выражения. Этот start_time использовался как временной "ноль" для шкалы PGN. Расчет времени каждого хода в видео: для каждого хода i в запрошенном диапазоне (от start_move до end_move) извлекался его собственный timestamp timestamp_i из комментария [%ts ...] в PGN. Вычислялось время, прошедшее с момента первого хода в PGN: elapsed_time_ms = timestamp_i - start_time. Это относительное время в миллисекундах преобразовывалось в секунды и прибавлялось к времени первого хода, найденному ранее с помощью CV: timestamp_sec = first_move_time + elapsed_time_ms / 1000.0. Полученное timestamp_sec представляло собой расчетное время совершения i-го хода в секундах от начала видеофайла. Формирование интервала вырезки: для каждого timestamp_sec создавался временной интервал (start_sec, duration). Начало интервала start_sec бралось за 3 секунды до расчетного момента хода (max(0, timestamp_sec - 3)), а длительность duration устанавливалась в 6 секунд. Этот 6-секундный интервал вокруг предполагаемого момента хода выбирался как эвристика, чтобы с запасом захватить само движение фигуры, возможно, короткое обдумывание непосредственно перед ходом и реакцию после хода (например, нажатие часов).

8) Объединение близких интервалов (merge_close_timestamps)

Список всех рассчитанных 6-секундных интервалов [(start_sec, duration), ...] сортировался по времени начала start_sec. Алгоритм проходил по отсортированному списку и объединял интервалы, которые были близки друг к другу. Если временной разрыв между концом предыдущего объединенного интервала (prev_start + prev_duration) и началом текущего интервала (start) был меньше или равен gap секунд (например, gap = 10), то текущий интервал "приклеивался" к предыдущему путем обновления длительности последнего: merged[-1] = (prev_start, (start + duration) - prev_start). Если разрыв был больше gap, то текущий интервал начинал новый объединенный сегмент в списке merged. Мотивация: в ситуациях, когда ходы делаются очень быстро (например, в блице или цейтноте), их 6-секундные окна могут сильно перекрываться или идти почти встык. Объединение таких близких интервалов в один более длинный клип предотвращало создание множества очень коротких, "дерганых" фрагментов и делало итоговое видео более гладким и цельным. Это также сокращало количество отдельных вызовов утилиты ffmpeg.

9) Нарезка и склейка с помощью ffmpeg (cut_video_ffmpeg)

Финальный этап, выполняющий фактическую обработку видеофайла на основе рассчитанных и объединенных временных интервалов. Нарезка сегментов: для каждого интервала (start, duration) из списка merged_timestamps выполнялась команда ffmpeg через модуль subprocess.run. Типичная команда выглядела так: ffmpeg -y -i video_path -ss start -t duration -c copy temp_file_i.mp4. -y: перезаписывать выходные файлы без запроса. -i video_path: указать входной видеофайл. -ss start: Задать время начала вырезки (в секундах). Позиция -ss (до или после -i) может влиять на скорость и точность. Постановка перед -i обычно быстрее (использует поиск по ключевым кадрам), но может быть менее точной. Постановка после -i (как в коде) обычно точнее (декодирует до нужной точки), но медленнее. Для режима -c copy точность обычно привязана к ключевым кадрам в любом случае, но поведение может зависеть от версии ffmpeg и формата файла. -t duration: задать длительность вырезаемого фрагмента. -c copy: Указать режим копирования потоков (Stream copy). Это критически важная опция для производительности и сохранения качества. ffmpeg не будет перекодиро-

вать видео и аудио, а просто скопирует соответствующие пакеты данных из исходного файла в выходной. Это происходит очень быстро и без потерь качества, связанных с пережатием. Режим возможен, если не требуется изменение кодеков, разрешения, битрейта и т.д. temp_file_i.mp4: имя временного файла для сохранения i-го вырезанного сегмента.Создание списка файлов для конкатенации: Создавался текстовый файл (file_list.txt), содержащий пути ко всем временным файлам в формате, понятном ffmpeg concat demuxer: file 'temp_file_0.mp4', file 'temp_file_1.mp4'. Конкатенация (склейка) сегментов: выполнялась вторая команда ffmpeg: ffmpeg -y -f concat -safe 0 -i file_list.txt -c copy output_path. -f concat: указать ffmpeg, что входной файл (-i file_list.txt) является списком файлов для демультиплексора конкатенации. -safe 0: разрешить использование небезопасных (например, относительных) путей в файле списка. Необходимо для корректной работы в большинстве случаев. -i file_list.txt: указать файл со списком сегментов. -c copy: снова использовать режим копирования потоков для быстрой склейки без перекодирования. Важно, чтобы все сегменты имели совместимые параметры потоков (что обычно так, если они вырезаны из одного исходного файла). output_path: имя финального видеофайла, содержащего все вырезанные и склеенные фрагменты в нужной последовательности. Очистка временных файлов: после успешного завершения конкатенации все временные файлы (temp_file_i.mp4) и файл списка (file_list.txt) удалялись с диска с помощью os.remove().

2.5.2.2 Используемые алгоритмы и технологии

Рассмотрим используемые алгоритмы и технологии:

1) Компьютерное зрение (CV) - библиотека OpenCV (cv2):

a. Предобработка изображений: Конвертация в оттенки серого (cv2.cvtColor), адаптивное выравнивание гистограммы (cv2.createCLAHE, clahe.apply), гауссовское размытие (cv2.GaussianBlur);

b. Детекция признаков: Обнаружение краев методом Кэнни (cv2.Canny);

c. Анализ форм и контуров: Поиск контуров (cv2.findContours), аппроксимация полигоном (cv2.approxPolyDP - алгоритм Дугласа-Пекера), вычисление площади контура (cv2.contourArea), фильтрация контуров по геометрическим свойствам (количество вершин, площадь);

d. Геометрические преобразования: Вычисление и применение матрицы перспективного преобразования (cv2.getPerspectiveTransform, cv2.warpPerspective);

e. Детекция движения/изменений: Вычисление разницы между кадрами (cv2.absdiff), бинаризация по порогу (cv2.threshold), морфологические операции (cv2.morphologyEx с cv2.MORPH_OPEN) для удаления шума.

2) Внешние утилиты:

a. ffmpeg: Мощнейший кроссплатформенный инструмент командной строки для обработки мультимедиа файлов. В "Версии 1" использовался для двух критически важных операций;

b. Точная и быстрая нарезка: Вырезание временных сегментов из видеофайла без перекодирования (-c copy);

c. Бесшовная конкатенация: Склейка полученных сегментов в единый выходной файл, также без перекодирования (-f concat, -c copy).

2.5.2.3 Преимущества

Концептуальная автономность: Основное теоретическое преимущество заключалось в том, что подход не требовал никакой внешней информации о времени начала видеозаписи. Вся синхронизация должна была происходить автоматически на основе анализа самого видеоконтента. Это могло бы избавить пользователей от необходимости соблюдать строгие конвенции именования файлов или полагаться на другие внешние метаданные. Универсальность (теоретическая): В идеале, такой подход мог бы работать с любым видео и PGN, независимо от источника, при условии, что доска видна достаточно четко.

2.5.2.4 Недостатки и проблемы (детализация):

Фундаментальная неточность CV для определения времени: Практические тесты выявили, что определение момента первого хода (или любого другого хода) с по-

мощью CV было крайне неточным и ненадежным. Погрешности в first_move_time часто достигали 5-10 секунд, а в сложных условиях (плохое освещение, много движений рук) могли превышать и 30-60 секунд. Такая низкая точность абсолютно неприемлема для задачи нарезки коротких 6-секундных фрагментов вокруг конкретных ходов – вырезанный фрагмент мог вообще не содержать нужного хода или показывать совершенно другой момент партии. Экстремальная чувствительность к условиям съемки: алгоритмы CV, особенно основанные на поиске краев, разнице кадров и пороговых значениях, оказались чрезвычайно чувствительны к любым изменениям внешних условий, не связанных напрямую с ходами. Освещение: резкое изменение общего уровня освещенности (включение/выключение света, солнце вышло/зашло за тучу), появление или исчезновение локальных бликов на доске или фигурах, мерцание некоторых источников света – все это могло вызвать массивные ложные срабатывания детектора изменений. Тени: движущиеся тени от рук игроков, их голов, судей, проходящих мимо, или даже от облаков за окном могли быть интерпретированы как значительные изменения на доске. Ракурс и стабильность камеры: алмейшее дрожание камеры (съемка с рук), изменение ракурса или масштаба (зум) могли привести к срыву алгоритма обнаружения доски или вызвать ложное срабатывание детектора изменений по всей площади доски из-за смещения всего изображения. Посторонние объекты и действия: руки игроков, зависающие над доской перед ходом, случайное касание фигур без их перемещения, падение на доску мелких предметов (например, пылинки), появление/исчезновение объектов на фоне за пределами доски, но в кадре – всё это могло быть воспринято как релевантное изменение. Алгоритм не обладал семантическим пониманием того, что является ходом, а что нет. Множество ложных срабатываний детектора изменений: даже при идеальной съемке, типичное поведение игрока перед ходом (поднес руку к фигуре, задумался, отвел руку, снова поднес, взял фигуру, перенес, поставил, убрал руку) генерировало целую серию сигналов об изменении. Логика detect_first_move с проверкой стабильности пыталась справиться с этим, но часто ошибалась, фиксируя либо момент начала движения руки, либо момент ее убирания, а не сам момент постановки фигуры. Сложность, трудоемкость и непереносимость калибровки: все численные параметры алгоритма (пороги для Canny (50,

150), диапазон площади доски (60000, 300000), порог для cv2.threshold при сравнении кадров (35), процент base_threshold (0.05), количество stability_frames (2)) требовали тщательной ручной настройки под конкретные условия съемки: тип камеры, разрешение видео, освещение, расстояние до доски, цвет доски и фигур. Параметры, хорошо работавшие для одного видеофайла, часто оказывались совершенно неэффективными для другого, даже если он был снят в том же помещении, но в другое время суток или с другими игроками (например, из-за разного стиля движения рук). Необходимость такой индивидуальной и непредсказуемой калибровки делала невозможным создание универсального, робастного и готового к использованию решения "из коробки". Высокая вычислительная сложность и медленная работа: полный цикл обработки одного кадра (чтение, предобработка, обнаружение доски, перспективное преобразование, обнаружение изменений) требовал значительных вычислительных ресурсов CPU. Даже с применением стратегии пропуска кадров, анализ видео был очень медленным. Обработка стандартной часовой видеозаписи могла занимать многие минуты, а иногда и часы, только на этапе поиска первого хода. Это делало подход непрактичным для быстрой обработки большого количества партий. Проблема идентификации именно первого игрового хода: алгоритм реагировал на любое достаточно крупное визуальное изменение на доске после начала видеозаписи. Он не мог отличить первый ход партии (например, 1. e4) от предшествующих ему действий: расстановки фигур игроками перед началом игры, случайных поправок фигур до запуска часов, рукопожатия над доской. Если такие действия происходили после старта видеозаписи, но до первого хода, алгоритм мог ошибочно зафиксировать их как "первый ход", что приводило к полной рассинхронизации со шкалой времени PGN.

2.5.2.5 Причина перехода:

Комбинация фундаментальных и практически неустранимых проблем с точностью, надежностью, робастностью к условиям съемки и производительностью подхода, основанного на компьютерном зрении, сделала "Версию 1" непригодной для практического использования в контексте поставленной задачи. Стало очевидно, что попытка "угадать" время первого хода по сложному и зашумленному визуальному

ряду сопряжена со слишком большим количеством неконтролируемых внешних факторов и неопределенностей. В то же время, наличие точных временных меток [%ts] в PGN-файлах указывало на существование гораздо более прямого, детерминированного и потенциально точного пути решения задачи, если удастся надежно определить абсолютное время начала самой видеозаписи. Это привело к стратегическому решению полностью отказаться от использования CV для синхронизации и перейти к разработке методов, использующих внешние, но точные и контролируемые метаданные о времени.

2.5.2.6 Альтернативный подход: Использование встроенных метаданных видеофайлов

Прежде чем окончательно перейти к подходу, основанному на специальном именовании файлов ("Версия 2"), был кратко рассмотрен еще один потенциальный вариант, который также стремился избежать сложностей и неточностей компьютерного зрения, но при этом не требовал бы от пользователей или систем записи соблюдения строгих правил именования файлов. Идея заключалась в том, чтобы попытаться извлечь информацию о времени начала записи из стандартных метаданных, встроенных в сам видеофайл при его создании. Многие современные видеоформаты и контейнеры (например, MP4, MOV) позволяют хранить различную метаинформацию, включая временные метки, связанные с созданием, кодированием или захватом контента.

Предполагаемый метод работы:

Извлечение метаданных: с помощью стандартных библиотек Python (например, os.path.getctime или os.path.getmtime, хотя они часто отражают время создания/модификации файла в файловой системе, а не время записи контента) или, что более перспективно, с использованием специализированных библиотек для анализа медиафайлов, таких как hachoir, pymediainfo (обертка над MediaInfo), или путем прямого вызова утилиты ffprobe (часть пакета ffmpeg) с запросом на вывод метаданных (ffprobe -v quiet -print_format json -show_format -show_streams video_file.mp4). Цель - найти поле метаданных, которое с наибольшей вероятностью соответствует реальному времени начала записи видео. Потенциальные кандидаты могли включать поля вроде

'creation_time' (в тегах QuickTime/MP4), 'encoded_date', 'tagged_date' или другие специфичные для формата метки. Конвертация и синхронизация: извлеченная строка с меткой времени должна была быть распарсена, интерпретирована (с обязательным учетом возможного указания часового пояса, если он присутствует в метаданных, или с предположением о UTC или локальном времени) и преобразована в унифицированный формат, например, в объект datetime Python, приведенный к UTC. Расчет смещения: далее, абсолютно аналогично тому, как это планировалось в "Версии 2", предполагалось вычислить временное смещение (offset) между этой меткой времени начала видео (извлеченной из метаданных файла) и временем первого хода из PGN (полученным из метки [%ts]). Нарезка: использовать рассчитанный offset для определения временных интервалов нарезки всех остальных ходов точно так же, как в "Версии 2".

Причина отказа от подхода:

Несмотря на кажущуюся привлекательность этого метода (использование стандартных полей метаданных избавило бы от необходимости принудительно формировать имена файлов), более тщательное исследование и эксперименты с реальными видеофайлами из разных источников (разные камеры, программы записи, сервисы скачивания) быстро выявили фундаментальную ненадежность и непредсказуемость этого подхода. Были обнаружены следующие системные проблемы: Метаданные часто отсутствуют или некорректны: значительное количество видеофайлов, особенно полученных из непрофессиональных источников или после редактирования/перекодирования, либо вообще не содержали релевантных временных меток в метаданных, либо эти метки были явно неверными (например, установлены в дату по умолчанию, такую как 1904 год, или имели нулевое значение). Неоднозначность семантики временных меток: даже при наличии полей типа 'Creation Date', 'Modification Date', 'Encoded Date', их семантика оказалась крайне неоднозначной. 'Modification Date' чаще всего отражает время последнего сохранения файла в файловой системе (например, после копирования или скачивания), а не время начала записи. 'Creation Date' (например, в MP4) может указывать на время создания файла, но не обязательно начала записи. 'Encoded Date' может соответствовать времени завершения процесса кодирования, которое может произойти значительно позже реальной записи. Не было

найдено ни одного поля метаданных, которое бы гарантированно соответствовало моменту начала видеозахвата во всех или хотя бы в большинстве случаев. Проблемы с часовыми поясами и форматами: временные метки в метаданных часто записывались в локальном времени без явного указания часового пояса или смещения от UTC. Это делало невозможным точное преобразование в UTC для корректного сопоставления с метками [%ts] из PGN, которые обычно подразумевают UTC. Кроме того, формат записи времени мог варьироваться. Отсутствие стандартизации и зависимость от ПО/оборудования: различные форматы видеоконтейнеров (MP4, AVI, MOV, MKV и т.д.), разные кодеки и, что важнее, разное программное обеспечение и оборудование для записи и кодирования видео по-разному обрабатывают и записывают (или не записывают) временные метаданные. Поведение было совершенно непредсказуемым и зависело от всей цепочки создания видеофайла.

В результате этого исследования стало совершенно ясно, что полагаться на встроенные метаданные видеофайлов для задачи точной синхронизации невозможно из-за их крайней непоследовательности, ненадежности и непредсказуемости. Этот вывод окончательно укрепил решение о необходимости использования внешнего, но контролируемого и явного источника информации о времени начала видео. Таким источником могло стать только само имя файла, если оно будет формироваться по строгому соглашению внешней системой записи. Это и привело к разработке подхода, реализованного в "Версии 2".

2.5.3 "Версия 2": Использование временных меток из названия видео и PGN

Столкнувшись с фиаско "Версии 1" из-за фундаментальных проблем точности и надежности компьютерного зрения, разработка перешла к принципиально иному, основанному на данных, подходу. "Версия 2" ставила перед собой цель достичь высокоточной, детерминированной и воспроизводимой синхронизации между событиями шахматной партии (ходами с их временными метками из PGN) и их визуальным представлением в видеозаписи. Ключевая идея заключалась в полном отказе от попыток "угадать" время по визуальным признакам и переходе к использованию двух

независимых, но предположительно синхронизированных по абсолютной шкале времени (UTC) источников:

1) Абсолютное время начала видеозаписи (start_ts): предполагалось, что система, осуществляющая запись видео (например, компьютер, к которому подключена камера и который синхронизирован по NTP, или специализированное оборудование для трансляций), может точно зафиксировать момент начала записи (время первого кадра) и внедрить эту информацию в стандартизированном формате (Unix timestamp в миллисекундах) в имя создаваемого видеофайла;

2) Абсолютное время совершения ходов ([%ts timestamp]): Используя стандартные комментарии [%ts timestamp] в PGN-файлах. Как уже упоминалось, эти метки генерируются системами электронных шахматных досок (например, DGT), которые регистрируют каждое перемещение фигуры и связывают его с точным временем по своим внутренним часам, которые также предполагаются синхронизированными с UTC. Unix timestamp обеспечивает универсальный, не зависящий от часовых поясов, формат для этих меток.

Основная задача "Версии 2" сводилась к простому, но точному математическому расчету временного смещения (offset) между этими двумя временными шкалами: шкалой видео (начинающейся с 0 в момент start_ts) и шкалой PGN (где каждый ход привязан к абсолютной метке [%ts]). Найдя это смещение для одного опорного события (например, первого хода партии, для которого известны и start_ts видео, и [%ts] хода), можно было точно рассчитать положение любого другого хода из PGN на временной оси видео. Этот метод обещал полностью устранить все неопределенности и зависимости от условий съемки, присущие "Версии 1", и привязать точность результата исключительно к точности исходных временных меток, предоставляемых внешними системами записи видео и ходов.

2.5.3.1 Метод работы (детальное описание с дополнениями):

"Версия 2" реализовала этот новый, основанный на метаданных, подход следующим образом:

1) Соглашение об именовании файлов

Введение строгого формата именования видеофайлов <start_ts>-<end_ts>.<ext> стало краеугольным камнем и абсолютной необходимостью для работы всего подхода. Это не просто соглашение для удобства, а жизненно важный механизм передачи критически важной информации о временных границах видео. start_ts: эта метка времени (Unix timestamp в миллисекундах, UTC) должна максимально точно соответствовать абсолютному времени первого кадра в видеофайле. Любое расхождение между значением start_ts в имени файла и реальным временем начала видеоконтента приведет к систематической ошибке во всех последующих расчетах смещения и, как следствие, к неправильной нарезке всех фрагментов. end_ts: метка времени конца записи (Unix timestamp в миллисекундах, UTC). Хотя "Версия 2" напрямую не использовала end_ts для расчета смещения ходов, её наличие в формате имени было предусмотрено для возможности базовой проверки консистентности (например, end_ts должен быть больше start_ts) и потенциального использования в будущих версиях или других инструментах (как это произошло в "Версии 4", где end_ts используется для определения временных границ файла при работе с несколькими файлами). Точность start_ts определяет предел точности всего метода. Если система записи видео имеет погрешность в фиксации времени начала даже на долю секунды (например, из-за задержек запуска кодировщика или неточной синхронизации системных часов), эта погрешность напрямую перенесется на результат нарезки. Таким образом, надежность и точность "Версии 2" полностью зависят от качества и точности работы внешней системы, генерирующей видеофайлы и их имена. Ответственность за предоставление корректных start_ts лежит вне данного алгоритма.

2) Парсинг имени файла (parse_video_start_time)

Эта функция выполняла роль надежного парсера и валидатора имени файла, гарантируя, что входные данные соответствуют ожидаемому формату. os.path.basename(video_path) извлекало только имя файла из полного пути, обеспечивая корректную работу независимо от формата входного video_path. Регулярное выражение re.match(r"(\d+)-\d+.\w+", base) выполняло две задачи одновременно: валидация: проверяло, соответствует ли начало имени файла шаблону "число-дефис-число-точка-расширение". Если нет, re.match возвращало None; извлечение: если формат

совпадает, группа захвата (\d+) изолировала и возвращала строку, содержащую start_ts. Генерация исключения ValueError в случае несоответствия формата имени файла являлась важной защитой от обработки некорректных входных данных, которая могла бы привести к непредсказуемым ошибкам на последующих этапах (например, при попытке преобразовать нечисловую строку в int). Преобразование извлеченного start_timestamp (строки) в int, деление на 1000 (перевод из миллисекунд в секунды) и создание объекта datetime с использованием datetime.utcfromtimestamp() — это стандартный и единственно правильный способ работы с Unix timestamps в Python для получения времени в UTC. Использование utcfromtimestamp вместо fromtimestamp критически важно для обеспечения глобальной согласованности времени, так как метки [%ts] из PGN также обычно подразумевают UTC или должны быть приведены к нему заранее. Это устраняет зависимость от локальных настроек часового пояса сервера, где выполняется скрипт.

3) Получение времени первого хода из PGN (get_first_move_timestamp:

Эта функция отвечала за нахождение "якоря" синхронизации со стороны шахматной партии – абсолютного времени первого хода. Использование библиотеки python-chess (chess.pgn.read_game) позволяло работать с PGN не как с простым текстом, а как со структурированными данными (дерево ходов, комментарии, заголовки). Это значительно упрощало навигацию по партии и извлечение нужной информации. Алгоритм целенаправленно проходил по основной (главной) вариации партии (game.mainline_nodes()) и искал первый узел (ход), у которого в поле comment содержалась метка [%ts ...]. Выбор именно первого хода в качестве якоря логичен, так как он является естественной точкой отсчета для относительных временных интервалов в PGN. (Примечание: если у самого первого хода метки нет, функция, скорее всего, найдет метку у следующего хода, что может внести небольшую погрешность, если время между первым и этим ходом было значительным). Регулярное выражение re.search(r"[%ts (\d+)]", node.comment) надежно извлекало числовое значение timestamp из строки комментария, игнорируя любой другой текст, который мог там присутствовать. Наличие хотя бы одной метки [%ts] (в идеале — у первого или одного

из первых ходов) в PGN-файле являлось абсолютно необходимым условием для работы этого метода. Если PGN-файл не содержал таких меток, "Версия 2" была бы неспособна выполнить синхронизацию и должна была бы выдать ошибку.

4) Расчет смещения первого хода в видео (calculate_offset_in_video:

Это был кульминационный этап синхронизации, где две независимые временные шкалы (видео и PGN) соединялись через общее событие – первый ход. Получив video_start_time (абсолютное время начала видео из имени файла) и first_move_time (абсолютное время первого хода из PGN) в виде объектов datetime (в UTC), можно было выполнять точные арифметические операции с временем. offset_timedelta = first_move_time - video_start_time вычисляло разницу между двумя моментами времени, результатом являлся объект timedelta, который точно представлял временной интервал между началом видео и совершением первого хода. offset = offset_timedelta.total_seconds() преобразовывало этот интервал timedelta в скалярное значение – количество секунд. Это значение offset и представляло собой время на шкале видео (относительно начала видеофайла, т.е. от 0), соответствующее моменту совершения первого хода. Это и было искомое смещение, которое позволяло перевести любой момент времени из абсолютной шкалы PGN в относительную шкалу видео. Проверка if offset < 0: была фундаментальной проверкой консистентности входных данных. Отрицательное смещение означало бы, что первый ход был сделан до начала записи видео, что физически невозможно в корректно собранных данных. Такой результат однозначно указывал на серьезную ошибку: либо start_ts в имени файла был неверным (завышенным), либо [%ts] первого хода в PGN был неверным (заниженным), либо часы системы, записывающей видео, и часы системы, генерирующей PGN, были существенно рассинхронизированы. Игнорирование этой проверки могло привести к попытке вырезать видео с отрицательными временными метками на последующих этапах, что вызвало бы ошибку ffmpeg.

5) Извлечение временных меток для нарезки (extract_move_timestamps:

Эта функция использовала рассчитанное смещение first_move_offset для определения временных координат каждого целевого хода (в диапазоне от start_move до end_move) на временной шкале видеофайла. Внутри этой функции снова находился

start_time (timestamp первого хода в PGN), который служил точкой отсчета для относительных временных интервалов в PGN. Время любого хода i можно было рассчитать как: время_i_в_PGN = start_time + (время_прошедшее_от_start_time_до_i). Для каждого хода i в заданном диапазоне: извлекался его абсолютный timestamp timestamp_i из комментария [%ts ...]; рассчитывалось время, прошедшее с момента первого хода в PGN: elapsed_time_ms = timestamp_i - start_time; это относительное время переводилось в секунды: elapsed_time_in_seconds = elapsed_time_ms / 1000.0; ключевая формула проецирования: Рассчитывалось время i-го хода на шкале видео: timestamp_sec = first_move_offset + elapsed_time_in_seconds. Эта формула брала относительное время хода i внутри партии (elapsed_time_in_seconds) и добавляла его к моменту времени первого хода на шкале видео (first_move_offset). Результат timestamp_sec — это точное (в пределах точности исходных start_ts и [%ts]) время i-го хода в секундах от начала видеофайла; Формирование 6-секундного интервала (start_sec, duration) вокруг timestamp_sec (с началом start_sec = max(0, timestamp_sec - 3) и duration = 6) оставалось такой же эвристикой, как и в "Версии 1", предназначенной для визуализации хода с небольшим контекстом до и после. Длительность (6с) и смещение начала (-3с) являлись настраиваемыми параметрами, которые можно было бы вынести в конфигурацию для адаптации под разные нужды (например, для блица интервалы можно делать короче, для глубокого анализа — длиннее, захватывая больше времени на обдумывание). Защита max(0, ...) предотвращала получение отрицательного времени начала для самых первых ходов.

6) Объединение близких интервалов (merge_close_timestamps:

Эта функция выполняла ту же роль, что и в "Версии 1": улучшение визуального восприятия итогового видео путем устранения очень коротких пауз между фрагментами, соответствующими быстро сделанным ходам.Параметр gap (например, 10 секунд) определял максимальный допустимый разрыв между концом одного фрагмента и началом следующего для их объединения. Выбор значения gap влиял на результат: меньшее значение gap приводило к меньшему числу слияний и более "фрагментированному" видео, точно следующему за каждым ходом; большее значение gap созда-

вало более длинные непрерывные куски, которые могли включать и паузы между ходами, но выглядели более плавно. Оптимальное значение gap могло зависеть от временного контроля партии (блиц, рапид, классика) и целевого назначения видео (динамичные хайлайты vs детальный разбор).Алгоритм слияния (сортировка списка интервалов по времени начала и последующий итеративный проход с проверкой разрыва и обновлением последнего элемента в списке merged) являлся стандартным и эффективным решением для этой задачи.

7) Нарезка и склейка (cut_video_ffmpeg):

Эта функция, как и в "Версии 1", делегировала всю фактическую работу по манипуляции с видеоданными мощной внешней утилите ffmpeg. Вызов ffmpeg с опциями -ss start -t duration выполнял точную (в пределах возможностей формата и ключевых кадров при -c copy) нарезку сегмента по времени. Опция -c copy оставалась критически важной для обеспечения: высокой скорости: операции копирования потоков на порядки быстрее любого перекодирования; сохранения исходного качества: отсутствие пережатия гарантировало, что качество вырезанных фрагментов идентично качеству исходного видео; требование совместимости: этот режим работал только при условии, что не требуется изменять кодеки или основные параметры видео/аудио потоков между входом и выходом, что идеально подходило для задачи простой нарезки и склейки. Метод конкатенации через временный файл file_list.txt и команду ffmpeg -f concat ... -c copy ... являлся стандартным и рекомендованным ffmpeg способом для быстрой и корректной склейки файлов без перекодирования, особенно важным для сохранения правильных временных меток пакетов (PTS/DTS) в итоговом файле.

2.5.3.2 Используемые алгоритмы и технологии

В ходе разработки системы обработки видео были использованы технологии:

1) Парсинг и валидация данных: Использование регулярных выражений (re) для точного извлечения временных меток из строго форматированных строк (имя файла, комментарии PGN). Использование python-chess для семантически корректного парсинга структуры PGN;

2) Точная временная арифметика: Активное использование стандартного модуля datetime для представления абсолютных моментов времени (как datetime объекты в UTC) и timedelta для представления временных интервалов. Это позволяло выполнять вычисления с субсекундной точностью, необходимой для точной синхронизации. Стандартизация на UTC обеспечивала независимость от локальных часовых поясов;

3) Взаимодействие с внешними процессами: Использование стандартного модуля subprocess (subprocess.run) для надежного запуска ffmpeg, передачи ему параметров и ожидания завершения;

4) Высокопроизводительная обработка видео: Применение ffmpeg в режиме копирования потоков (-c copy) для максимально быстрой нарезки и склейки без деградации качества видео. Использование concat demuxer для эффективной конкатенации;

5) (Отличие от “Варианта 1”): Полный отказ от OpenCV и алгоритмов компьютерного зрения для основной задачи синхронизации.

2.5.3.3 Преимущества

Высокая точность: Главное и решающее преимущество над "Версией 1". Точность синхронизации стала ограничена только точностью исходных временных меток (start_ts в имени файла и [%ts] в PGN) и точностью работы ffmpeg при нарезке. При качественных исходных данных достигалась субсекундная точность. Надежность и детерминизм: Результат стал полностью детерминированным и воспроизводимым. При одинаковых входных файлах (видео с корректным именем и PGN с метками) результат нарезки всегда был бы одинаковым, независимо от условий съемки, освещения, движения рук и т.д. Значительно возросшая скорость: Отказ от ресурсоемких операций компьютерного зрения на каждом кадре привел к кардинальному ускорению процесса. Основное время теперь тратилось на быстрые операции ffmpeg (-c copy), что позволило обрабатывать видео значительно быстрее. Относительная простота алгоритма: Логика расчета смещения и проецирования временных меток была значительно проще в реализации, понимании и отладке по сравнению со сложным и капризным конвейером CV-алгоритмов из "Версии 1".

2.5.3.4 Недостатки и проблемы

Полная зависимость от качества и наличия метаданных: Вся надежность и точность "Версии 2" переместились с алгоритма на входные данные. Метод работал корректно только при одновременном выполнении следующих условий: Видеофайл имел имя строго в формате <start_ts>-<end_ts>.<ext>. Значение start_ts в имени файла точно соответствовало реальному времени начала видео (UTC). PGN-файл содержал комментарии [%ts timestamp] хотя бы для первого хода и всех последующих ходов в интересующем диапазоне. Значения timestamp в PGN были точными и синхронизированы с той же временной шкалой (UTC), что и start_ts. Любая ошибка, неточность или отсутствие хотя бы одного из этих элементов (опечатка в имени файла, неверный start_ts, отсутствие меток [%ts], рассинхронизация часов систем записи) приводила к полностью некорректному результату нарезки. Требовалась внешняя система контроля качества и консистентности входных данных. Критическое ограничение: работа только с одним видеофайлом: Алгоритм был разработан исходя из предположения, что вся партия записана в один непрерывный видеофайл. Он был неспособен обрабатывать сценарии, когда запись партии была разделена на несколько видеофайлов (например, из-за ограничений на размер файла, перезапуска записи или использования нескольких камер). Это являлось серьезным функциональным недостатком, так как на практике, особенно на длительных турнирах, партии часто записываются именно на несколько файлов.

2.5.3.5 Причина перехода

Несмотря на успешное решение ключевой проблемы точности синхронизации, критическое ограничение на работу только с одним видеофайлом сделало "Версию 2" недостаточно гибкой и практичной для многих реальных сценариев использования (например, обработки полных турнирных записей). Необходимость поддержки мультифайловых записей стала главным стимулом для дальнейшей разработки и привела к созданию "Версии 4". "Версия 3" была разработана как промежуточный шаг, сфокусированный на исследовании возможностей оптимизации производительности "Версии 2" с использованием асинхронного программирования, прежде чем добавлять более сложную логику мультифайловости.

2.5.4 "Версия 3": Асинхронная оптимизация

"Версия 3" не вносила принципиальных изменений в сам алгоритм синхронизации и расчета временных меток, унаследовав его напрямую из "Версии 2". Основной и единственной целью этой итерации была оптимизация производительности всего процесса нарезки видео, особенно для случаев, когда требовалось вырезать большое количество фрагментов (например, всю партию целиком или длинный ее участок). Идея заключалась в применении асинхронного программирования с использованием библиотеки asyncio в Python для достижения конкурентного (а потенциально и параллельного) выполнения тех операций, которые в синхронном коде "Версии 2" могли бы блокировать выполнение и приводить к простоям:

1) Операции ввода-вывода (I/O): Чтение PGN-файла с диска, запись временного файла file_list.txt, удаление множества временных видеофрагментов (temp_file_i.mp4) после завершения. В синхронном коде поток выполнения блокируется, ожидая завершения каждой из этих дисковых операций. Во время этого ожидания CPU может простаивать;

2) Запуск и ожидание завершения внешних процессов (ffmpeg): Наиболее значимый потенциал для оптимизации лежал в этапе нарезки множества сегментов. В "Версии 2" каждый вызов ffmpeg для вырезки очередного фрагмента выполнялся последовательно: скрипт запускал ffmpeg, ждал его завершения, затем запускал следующий. Каждый такой запуск и выполнение занимают некоторое время. Выполнение этих операций конкурентно (т.е. запуск нескольких процессов ffmpeg без ожидания завершения предыдущих) могло бы существенно сократить общее время, затрачиваемое на этап нарезки, особенно на многоядерных системах.

Таким образом, "Версия 3" стремилась использовать неблокирующие вызовы и цикл событий asyncio для более эффективной утилизации системных ресурсов (CPU, диск) и сокращения общего времени ожидания, необходимого для выполнения всей задачи. Логика расчета что и когда резать осталась прежней, изменился только как это исполняется.

2.5.4.1 Метод работы

"Версия 3" адаптировала синхронный код "Версии 2" к асинхронной модели выполнения asyncio:

1) Переход на синтаксис asyncio

Все основные функции, выполняющие I/O операции или запускающие внешние процессы, были переписаны с использованием ключевых слов async def, превращая их в корутины (coroutines). Внутри корутин, вызовы потенциально блокирующих операций (как асинхронных версий I/O, так и ожидания процессов) предварялись ключевым словом await. Это позволяло приостановить выполнение текущей корутины, не блокируя весь поток, и передать управление циклу событий (event loop) asyncio. Основная точка входа программы использовала asyncio.run(main()) для запуска цикла событий, который координировал выполнение всех созданных асинхронных задач (корутин). Цикл событий отслеживал, какие задачи готовы к выполнению (т.е. не ожидают завершения await), и поочередно передавал им управление.

2) Асинхронный ввод-вывод с использованием aiofiles

Стандартные синхронные функции для работы с файлами (open, file.read, file.write, os.remove) были заменены их асинхронными аналогами из библиотеки aiofiles (например, aiofiles.open, async with aiofiles.open(...) as f: await f.read(), await aiofiles.os.remove()). Использование aiofiles позволяло выполнять дисковые операции, не блокируя цикл событий. Когда корутина доходила до await aiofiles_operation(), ее выполнение приостанавливалось, но цикл событий мог немедленно переключиться на выполнение других готовых корутин (например, запуск следующего ffmpeg или обработку данных). Когда дисковая операция завершалась (например, данные были прочитаны или записаны), цикл событий получал уведомление и мог возобновить выполнение ожидавшей корутины с того места, где она была приостановлена.

3) Конкурентный запуск процессов ffmpeg

Ключевая оптимизация была реализована на этапе нарезки сегментов. Вместо subprocess.run использовался asyncio.create_subprocess_exec. Эта функция запускала внешний процесс (ffmpeg), но не блокировала вызывающую корутину, а немедленно возвращала объект Process. Была создана асинхронная функция-обертка, например,

async def run_ffmpeg_command(...), которая инкапсулировала запуск одного процесса ffmpeg с нужными параметрами через asyncio.create_subprocess_exec и последующее асинхронное ожидание его завершения (stdout, stderr = await process.communicate()). Эта обертка также могла включать логику проверки кода возврата process.returncode и обработки ошибок. Для запуска нарезки всех сегментов создавался список задач (корутин) – по одной задаче run_ffmpeg_command(...) для каждого временного интервала (start, duration). Затем использовалась функция asyncio.gather(*cut_tasks), которая принимала этот список задач и запускала их конкурентно. Эффект asyncio.gather: цикл событий asyncio начинал выполнять все эти задачи "одновременно". На многоядерной системе это могло привести к реальному параллелизму: несколько процессов ffmpeg могли действительно работать одновременно на разных ядрах CPU, каждый обрабатывая свой сегмент видео. Степень реального параллелизма ограничивалась количеством доступных ядер CPU и пропускной способностью дисковой подсистемы (особенно если все сегменты читались из одного и того же исходного видеофайла на одном диске). Даже на одноядерной системе конкурентность могла дать выигрыш за счет того, что пока один процесс ffmpeg ожидал данных с диска (I/O wait), цикл событий мог передать управление другому процессу ffmpeg или другим асинхронным задачам. Финальная стадия конкатенации одним процессом ffmpeg оставалась синхронной точкой (после asyncio.gather), так как она требовала наличия всех временных файлов.

4) Асинхронная очистка временных файлов

Аналогично нарезке, удаление временных файлов (temp_file_i.mp4 и file_list.txt) также было обернуто в асинхронные вызовы (aiofiles.os.remove) и могло выполняться конкурентно с помощью asyncio.gather, что могло немного ускорить самый последний этап работы скрипта, особенно при большом количестве временных файлов.

5) Профилирование производительности

Для объективной оценки эффективности асинхронного подхода было важно измерять общее время выполнения скрипта (например, с использованием

time.perf_counter() в начале и конце main функции) и сравнивать его со временем выполнения синхронной "Версии 2" на тех же входных данных и том же оборудовании. Это позволяло понять, дает ли асинхронность реальный выигрыш в конкретных условиях.

2.5.4.2 Используемые алгоритмы и технологии (детализация)

Используем следующие алгоритмы:

1) Асинхронное программирование (asyncio): Основная парадигма управления конкурентностью в "Версии 3". Использовались ключевые компоненты asyncio: цикл событий, корутины (async def), оператор await для неблокирующего ожидания, функции asyncio.run, asyncio.create_subprocess_exec для запуска внешних процессов и asyncio.gather для конкурентного выполнения множества задач;

2) Асинхронный ввод-вывод (aiofiles): Библиотека, предоставляющая asyncio-совместимые версии стандартных функций для работы с файлами, позволяющие избежать блокировки цикла событий при дисковых операциях;

3) Базовая логика и инструменты (унаследованы из "Версии 2"): Алгоритмы расчета временного смещения и интервалов нарезки, использование re, datetime, python-chess, ffmpeg (для фактической нарезки и склейки) остались абсолютно теми же, что и в "Версии 2". Асинхронность повлияла только на способ их оркестрации и выполнения.

2.5.4.3 Преимущества:

Потенциальное значительное ускорение: Основное и единственное преимущество "Версии 3". Теоретически, конкурентная (и потенциально параллельная на многоядерных CPU) нарезка сегментов видео с помощью ffmpeg могла значительно сократить общее время выполнения задачи, особенно для длинных партий с большим количеством ходов в заданном диапазоне нарезки. Выигрыш мог быть особенно заметен на системах с быстрыми дисками и несколькими ядрами CPU. Более эффективное использование системных ресурсов: Асинхронный подход позволял лучше утилизировать CPU во время ожидания завершения операций ввода-вывода (диск, сеть) или внешних процессов. Современный подход и лучшая интеграция: Использование asyncio является современным стандартом для написания высокопроизводительных

I/O-bound приложений на Python. Это облегчало бы потенциальную интеграцию данного модуля в более крупные асинхронные системы или веб-сервисы.

2.5.4.4 Недостатки и проблемы:

Функциональные ограничения "Версии 2" не устранены: Самый главный недостаток – "Версия 3" не решала проблему неспособности работать с несколькими видеофайлами. Она просто делала быстрее то, что делала "Версия 2". Также сохранялась полная зависимость от качества и наличия входных метаданных (имя файла, метки [%ts]). Значительное усложнение кода: Асинхронный код с использованием asyncio объективно сложнее для написания, понимания, отладки и поддержки по сравнению с прямолинейным синхронным кодом. Концепции цикла событий, корутин, управления задачами, обработки исключений в асинхронном контексте требуют дополнительных знаний и более внимательного подхода к проектированию. Отладка асинхронного кода может быть сложнее из-за нелинейного потока выполнения и менее интуитивных стектрейсов. Производительность не гарантирована и зависит от среды: Выигрыш в производительности от асинхронности не был гарантирован и сильно зависел от конкретного оборудования (количество ядер CPU, скорость диска) и характера нагрузки (количество сегментов, их длительность). На системах с медленным диском (который становился бы узким местом) или малым числом ядер CPU эффект от параллелизации ffmpeg мог быть незначительным или даже отсутствовать (из-за накладных расходов самого asyncio). Требовалось тщательное тестирование и профилирование в целевой среде для подтверждения преимуществ.

2.5.4.5 Причина перехода

Несмотря на потенциальные выгоды в производительности, которые могла дать "Версия 3", она по-прежнему оставалась функционально неполной из-за неспособности обрабатывать мультифайловые видеозаписи, что являлось критическим требованием для практического применения. Решение этой функциональной проблемы было признано более приоритетным, чем дальнейшая преждевременная оптимизация производительности на данном этапе. Поэтому основной фокус разработки сместился на создание "Версии 4", которая должна была добавить поддержку нескольких видеофайлов, при этом, по возможности, сохранив и адаптировав асинхронную основу,

заложенную в "Версии 3", для использования ее преимуществ в более сложном сценарии. "Версия 3" послужила полезным полигоном для отработки техник асинхронного запуска ffmpeg.

2.5.5 "Версия 4": Поддержка партий, разделенных на несколько видеофайлов

"Версия 4" была разработана как финальное и наиболее функционально полное решение в рамках описанной эволюции. Ее главная цель – устранить основное критическое ограничение всех предыдущих версий – неспособность корректно работать с видеозаписями шахматных партий, которые состоят из нескольких последовательных видеофайлов. Такое разделение на файлы является частой практикой при длительных записях (например, на турнирах). Таким образом, целью "Версии 4" было создание практически применимого инструмента, который бы обладал следующими ключевыми характеристиками:

1) Корректная обработка списка видеофайлов: Способность принимать на вход упорядоченный список путей к видеофайлам, представляющим одну непрерывную (или почти непрерывную) запись партии, и правильно определять, в каком из файлов находится каждый интересующий ход;

2) Сохранение высокой точности синхронизации: Использование того же надежного подхода к синхронизации, что и в "Версии 2", и в "Версии 3", основанного на временных метках start_ts/end_ts из имен файлов и [%ts] из PGN;

3) Использование асинхронной обработки: Сохранение и адаптация асинхронной архитектуры из "Версии 3" для потенциального повышения производительности при нарезке сегментов, теперь уже из разных исходных файлов;

4) Функциональная полнота: Предоставление решения, готового к использованию в реальных сценариях обработки шахматных видеозаписей.

2.5.5.1 Метод работы (детальное описание с дополнениями):

"Версия 4" существенно расширила и усложнила асинхронную архитектуру "Версии 3", добавив логику для навигации и выбора правильного исходного видеофайла для каждого хода:

1) Список видеофайлов как основной входной параметр

Алгоритм теперь принимал на вход не один video_path, а упорядоченный (по времени) список путей ко всем видеофайлам, составляющим запись данной партии: video_files = ['/path/to/part1.mp4', '/path/to/part2.mp4', ...]. Требование к формату именования <start_ts>-<end_ts>.<ext> стало критически важным теперь для каждого файла в этом списке. Без этой информации (как начала start_ts, так и конца end_ts для каждого файла) было бы невозможно определить, какой временной интервал покрывает каждый файл и, следовательно, в каком файле искать конкретный ход по его [%ts] метке.

2) Предварительный сбор и кэширование метаданных файлов (реализация parse_video_start_end)

На этапе инициализации, перед началом обработки ходов, алгоритм должен был однократно пройти по всему списку video_files. Для каждого video_path из списка вызывалась функция (например, parse_video_start_end), которая парсила имя файла и извлекала из него обе временные метки: время начала (start_ms) и время конца (end_ms) в миллисекундах UTC. Эта извлеченная информация (имя файла, его start_ms и end_ms) сохранялась (кэшировалась) в памяти, например, в виде списка кортежей или словаря: video_metadata = [(filename1, start_ms1, end_ms1), (filename2, start_ms2, end_ms2), ...]. Этот список, скорее всего, также сортировался по start_ms для ускорения последующего поиска. Цель кэширования: избежать многократного парсинга имен одних и тех же файлов внутри цикла обработки каждого хода, что повышало эффективность.

3) Поиск нужного видеофайла по timestamp'у хода (find_video_segment)

Эта новая функция выполняла ключевую операцию маршрутизации: для заданного абсолютного времени хода timestamp_ms (из [%ts] PGN) она должна была определить, в каком из видеофайлов из списка video_files находится этот момент времени. Функция итерировала по кэшированным метаданным video_metadata. Для каждого файла (filename, start_ms, end_ms) проверялось условие: if start_ms <= timestamp_ms < end_ms:. Это условие точно определяло принадлежность timestamp_ms временному интервалу, покрываемому данным файлом. Использование строгого неравенства < для end_ms важно для однозначной обработки ситуаций, когда timestamp_ms точно

совпадает с границей между двумя файлами (он будет отнесен к файлу, в котором он является началом или серединой, а не концом). Если подходящий файл найден, функция возвращала его метаданные (например, filename, start_ms, end_ms). Обработка ошибок: Если ни один файл из video_metadata не содержал заданный timestamp_ms (т.е. ход попадал во временной "пробел" между файлами или выходил за пределы всех файлов), функция должна была сигнализировать об ошибке (например, возвращать None или генерировать исключение ValueError), так как вырезать этот ход было невозможно. Надежность find_video_segment зависела от предположения о целостности и непрерывности (или наличии лишь небольших допустимых перекрытий/пропусков) временного покрытия, предоставляемого списком video_files.

4) Извлечение временных меток с привязкой к конкретному файлу (extract_move_timestamps)

Это фундаментальное отличие от логики "Версии 2" и "Версии 3" Теперь расчет времени хода выполнялся не относительно одного глобального first_move_offset, а относительно начала того видеофайла, который содержит данный ход. Для каждого хода i в заданном диапазоне (start_move до end_move). Извлекался его абсолютный timestamp_i (в миллисекундах) из PGN. Вызывалась функция find_video_segment(timestamp_i, video_metadata) для определения, в каком файле (video_file) и с какими границами (video_start_ms, video_end_ms) находится этот ход. Если файл не найден, ход пропускался или генерировалась ошибка. Рассчитывалось локальное время начала хода внутри найденного video_file: start_in_video_ms = timestamp_i - video_start_ms. Это время переводилось в секунды: start_in_video_sec = start_in_video_ms / 1000.0. Формировался интервал для нарезки, как и раньше, со смещением -3 секунды и длительностью 6 секунд, но теперь он был привязан к конкретному файлу: start_in_video_adjusted = max(0, start_in_video_sec - 3), duration = 6. В итоговый список интервалов для нарезки добавлялся кортеж из трех элементов: (video_file, start_in_video_adjusted, duration). Теперь каждый интервал "знал", из какого исходного файла его нужно вырезать и с какого локального времени начинать.

5) Объединение близких интервалов с учетом файла (merge_close_timestamps)

Алгоритм объединения был адаптирован для работы с новым форматом данных (filename, start, duration). Критически важно: Перед объединением список интервалов должен был быть отсортирован сначала по имени файла, а затем по времени начала внутри файла: timestamps.sort(key=lambda x: (x[0], x[1])). Это гарантировало, что алгоритм будет рассматривать последовательные интервалы из одного файла подряд. Условие слияния теперь проверяло не только близость по времени ((start - (last_start + last_duration)) <= gap), но и совпадение имени файла: if file == last_file and (start - (last_start + last_duration)) <= gap:. Это гарантировало, что объединяются только сегменты, которые будут вырезаны из одного и того же физического файла. Попытка "склеить" интервалы через границу между разными файлами на этом этапе была бы некорректной.

6) Асинхронная нарезка сегментов из разных исходных файлов (cut_video_segments)

Функция, отвечающая за запуск ffmpeg, теперь оперировала списком кортежей (file, start, dur). При формировании команд ffmpeg для конкурентного запуска через await asyncio.gather(...): параметр -i (входной файл) динамически подставлялся из file текущего кортежа, параметр -ss (время начала) динамически подставлялся из start текущего кортежа (локальное время внутри file), прараметр -t (длительность) подставлялся из dur, имя временного выходного файла (temp_file_i.mp4) по-прежнему генерировалось уникальным для каждого сегмента. Это позволяло asyncio и ffmpeg конкурентно (и потенциально параллельно) вырезать фрагменты из разных исходных видеофайлов. Теоретически, если файлы part1.mp4, part2.mp4 и т.д. лежали на разных физических дисках, это могло бы дать дополнительное ускорение за счет параллельного дискового I/O, помимо параллелизма CPU. Этапы создания file_list.txt (который теперь содержал имена временных файлов, созданных из разных источников) и финальной конкатенации с помощью ffmpeg -f concat ... -c copy ... оставались практически прежними, так как concat demuxer без проблем работает с последовательностью файлов, даже если они были изначально созданы из разных источников, при условии их совместимости по кодекам и параметрам.

2.5.5.2 Используемые алгоритмы и технологии (детализация):

Базис "Версии 3": Сохранено использование asyncio для асинхронной оркестрации, aiofiles для асинхронного I/O, re для парсинга, datetime для временных расчетов, python-chess для работы с PGN, ffmpeg для нарезки/склейки в режиме -c copy. Ключевые дополнения и модификации:

1) Управление метаданными списка файлов: Эффективное кэширование и структура данных для хранения временных рамок (start_ms, end_ms) для каждого файла из входного списка;

2) Алгоритм поиска интервала: Реализация функции find_video_segment для определения нужного файла по абсолютному timestamp'у;

3) Адаптация алгоритмов расчета и слияния: Модификация extract_move_timestamps для расчета локального времени внутри файла и merge_close_timestamps для корректной сортировки и слияния с учетом имени файла;

4) Адаптация логики нарезки: Динамическая генерация команд ffmpeg в cut_video_segments с подстановкой правильного имени входного файла (-i) и локального времени начала (-ss) для каждого сегмента.

2.5.5.2 Преимущества:

Практическая применимость и функциональная полнота: Решение ключевой проблемы мультифайловости сделало "Версию 4" готовой к использованию в реальных сценариях обработки записей с шахматных турниров и других мероприятий, где видео часто делится на части. Сохранение высокой точности: Точность синхронизации, основанная на временных метках, осталась на том же высоком уровне, что и в "Версии 2", и в "Версии 3", при условии качественных входных данных. Сохранение потенциала асинхронной производительности: Использование asyncio для конкурентной нарезки, теперь уже из разных файлов, сохранило потенциал для ускорения обработки по сравнению с полностью синхронным подходом. Объединение лучших качеств: "Версия 4" интегрировала точность "Версии 2", производительность"Версии 3" и добавила необходимую функциональность для работы с реальными данными.

2.5.5.3 Недостатки и проблемы:

Максимальные требования к качеству и полноте метаданных: Зависимость от метаданных стала еще более критичной. Теперь каждый видеофайл в последовательности должен был иметь имя в строгом формате <start_ts>-<end_ts>.<ext>, и все метки start_ts, end_ts должны были быть точными и согласованными между собой (т.е. end_ts одного файла должен примерно соответствовать start_ts следующего, без больших необъяснимых пропусков или перекрытий). PGN-файл по-прежнему должен был содержать точные и синхронизированные метки [%ts]. Система генерации PGN и видеофайлов должна была работать строго синхронно. Ошибка или неточность хотя бы в одном элементе (одно имя файла, одна метка [%ts]) могла нарушить всю цепочку обработки или привести к пропуску ходов. Максимальная сложность кода: Сочетание асинхронного программирования (asyncio) с нетривиальной логикой управления списком файлов, кэширования метаданных, поиска нужного сегмента и адаптации всех этапов обработки сделало код "Версии 4" наиболее сложным для понимания, отладки, поддержки и дальнейшего расширения по сравнению со всеми предыдущими версиями. Предположения о целостности и непрерывности видеозаписи: Алгоритм неявно предполагал, что предоставленный список video_files образует непрерывное или почти непрерывное покрытие времени партии в интересующем диапазоне ходов. Наличие значительных временных пропусков ("дыр") между файлами (например, если запись прерывалась на длительное время) привело бы к тому, что ходы, попадающие в эти пропуски, не были бы найдены функцией find_video_segment и, соответственно, не были бы вырезаны. Наличие значительных временных перекрытий между файлами могло бы вызвать неоднозначность при поиске файла (хотя текущая реализация find_video_segment с условием start_ms <= timestamp_ms < end_ms выберет первый подходящий файл в отсортированном списке, что может быть не всегда желаемым поведением, если требуется более сложная логика выбора при перекрытии). Требовался внешний контроль или предварительная обработка для обеспечения целостности и корректности последовательности видеофайлов перед их передачей в "Версию 4".

2.5.6 Итоги

Эволюция алгоритма автоматической нарезки видео шахматных партий, представленная в версиях от 1 до 4, наглядно демонстрирует классический инженерный процесс поиска оптимального решения задачи через последовательные итерации, выявление и устранение недостатков предыдущих подходов.

Начальная попытка ("Версия 1") использовать передовые, но сложные методы компьютерного зрения для достижения полностью автономной синхронизации, хотя и была концептуально привлекательной из-за отсутствия требований к внешним данным, оказалась на практике несостоятельной. Фундаментальные проблемы с точностью, робастностью к изменениям условий съемки и высокая вычислительная сложность сделали CV-подход непригодным для надежного решения поставленной задачи.

Решающим концептуальным сдвигом стал переход ("Версия 2") к использованию точных временных метаданных, предоставляемых внешними, но контролируемыми источниками: времени начала видео из специально сформированного имени файла (start_ts) и абсолютных временных меток ходов ([%ts]) из PGN-файла. Этот подход, основанный на математическом расчете смещения, обеспечил необходимую точность и детерминизм, но его практическое применение было серьезно ограничено возможностью работы только с одним видеофайлом на партию.

Следующая итерация ("Версия 3") была целиком сфокусирована на оптимизации производительности решения из "Версии 2". Путем внедрения асинхронного программирования (asyncio) удалось организовать конкурентное выполнение операций ввода-вывода и, что наиболее важно, параллельный запуск множества процессов нарезки ffmpeg. Это позволило значительно сократить потенциальное время обработки, особенно для партий с большим количеством ходов, эффективно используя ресурсы многоядерных систем.

Финальная разработка ("Версия 4") успешно решила ключевое функциональное ограничение предыдущих версий, добавив поддержку обработки партий, записанных на несколько последовательных видеофайлов. Она элегантно интегрировала меха-

низм поиска нужного файла для каждого хода по его временной метке и расчета относительного времени внутри этого файла в асинхронную архитектуру "Версии 3". Это позволило сохранить как высокую точность синхронизации, так и потенциальные выгоды от асинхронной обработки.

Таким образом, итоговое решение, представленное в "Версии 4", является наиболее полным, точным и практически применимым результатом данной разработки. Оно способно эффективно (благодаря asyncio) и точно (благодаря timestamps) генерировать видеофрагменты из набора исходных файлов, покрывающих шахматную партию. Однако важно подчеркнуть, что его успешная и надежная эксплуатация неразрывно связана с необходимостью обеспечения исключительно высокого качества, полноты и согласованности входных метаданных: строго стандартизированных имен всех видеофайлов с точными и непрерывными временными рамками (start_ts, end_ts) и PGN-файла с корректными, синхронизированными с видео метками [%ts]. Контроль качества и целостности этих исходных данных является абсолютным предварительным условием для получения надежных и осмысленных результатов с помощью разработанного IT-решения. Без выполнения этих условий даже самый совершенный алгоритм нарезки даст некорректный выход.

2.6 Программная реализация серверной части

Для разработки было решено создать репозиторий и разместить в нём все нужные папки: для разработки backend, frontend, и модели. Но скоро стало ясно, что такая разработка не очень удобна:

1) нужно будет создавать много веток и подветок под каждую часть проекта;
2) немного сложнее работать с отдельной частью проекта, из-за того что приходится работать во всём проекте сразу.

Поэтому было решено использовать такую возможность гита как submodules. Она позволяет создать репозиторий для каждой части проекта, а в главном репозитории сделать ссылки на нужные коммиты каждой части. Так получается решить обе проблемы, но возникает новая: обновление модулей. Его придётся делать вручную, что не очень удобно. На помощь здесь пришла такая возможность гита, как github actions. Она позволяет автоматизировать некоторые действия с репозиторием, в том

числе и обновление с модулей. Я настроил её так, чтобы она каждые 12 часов проверяла каждый репозиторий на обновление ветки main. И при необходимости обновляла главный репозиторий. Также для хранения всех этих репозиториев была создана организация нашей команды.

2.6.1 Архитектура проекта

2.6.1.1 Drawio

Далее было решено заняться разработкой архитектуры проекта. Так как в требованиях к оформлению нужна была красивая диаграмма, для начала было решено изучить инструмент под названием drawio. Там уже были встроены элементы для отображения архитектуры по нотации C4. Также гибкая система позволяла настроить буквально все начиная текстом на элементах заканчивая цветом фона.

2.6.1.2 Первый уровень по нотации C4

Первый уровень показывает системный контекст программы. Как она вписывается в окружающий мир. Вы можете ознакомиться с данным уровнем рисунке 2.5.

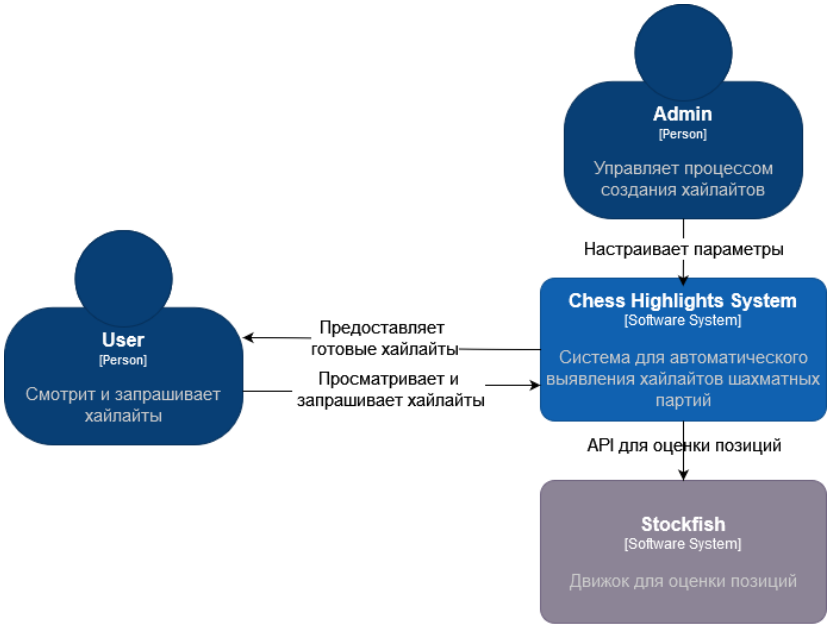

Рисунок 2.5 - Второй уровень архитектуры

2.6.1.3 Второй уровень по нотации C4

Второй уровень диаграммы - контейнеры. То есть она отображает общую форму архитектуры, распределение функций и обязанностей. В нашем случае архитектура должна была состоять из 4 главных компонентов:

1) Сайт, или же фронтенд. На нем соответственно должны были отображаться все обработанные партии. Также сайт должен был предоставлять интерфейс для запуска анализа, а также просмотра загруженной шахматной партии;

2) Backend. Это сердце нашего проекта. Отсюда происходят все взаимодействия с базой данных, сайтом и моделькой. Здесь реализованы все конечные точки, так же находится вся бизнес логика;

3) ML. Здесь будет разрабатываться модель для анализа шахматной партии. Она реализована как отдельный сервис для того, чтобы не мешайте ее разработке;

4) База данных. Будет использоваться PostgreSQL, так как это одна из самых популярных бесплатных баз с открытым исходным кодом. Она отлично подходит под нашу задачу. Нужна для того, чтобы хранить сами шахматные партии, результаты анализа, видео, а также нарезанные видео.

Более подробно можно ознакомиться на рисунке 2.6. На нем представлена диаграмма второго уровня архитектуры по нотации C4.

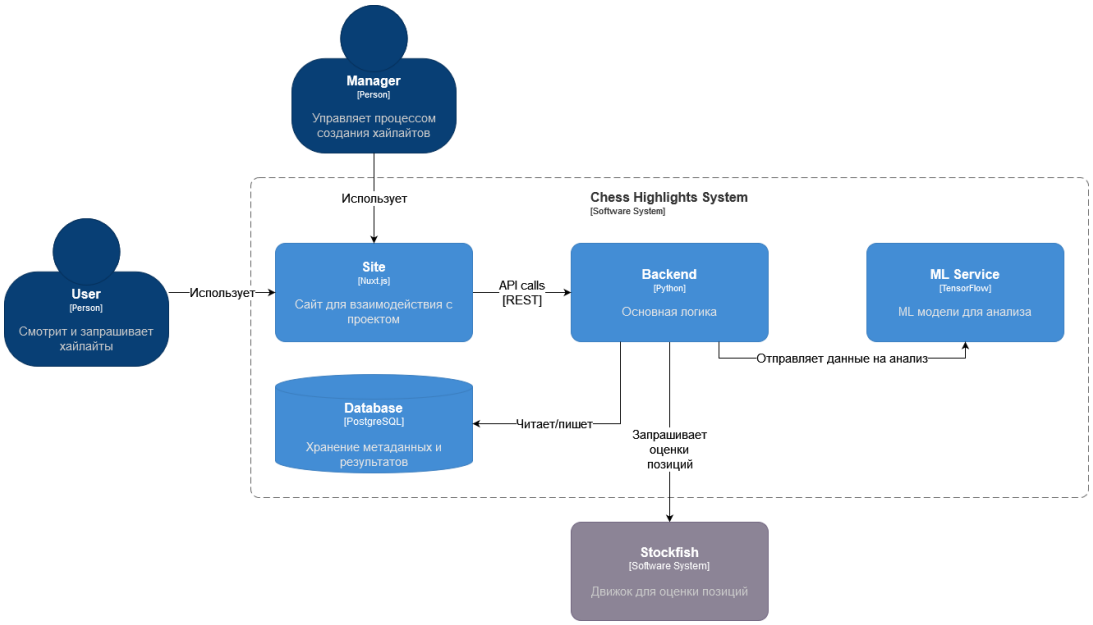

Рисунок 2.6 - Второй уровень архитектуры

2.6.1.4 Третий уровень по нотации C4

Третий уровень диаграммы - компоненты - показывает устройство контейнера. Рассмотрим третий уровень для бэкенда. Он состоит из нескольких модулей:

1) Core. Ядро приложения, содержащее основные модели данных и бизнес-логику. Этот модуль определяет основные сущности системы и правила их взаимодействия, независимо от конкретных технологий хранения данных или представления;

2) Utils. Вспомогательные утилиты. Содержит общие функции и классы, используемые в различных частях приложения. Обеспечивает повторное использование кода и стандартизацию общих операций;

3) DB. Модуль для работы с базой данных. Обеспечивает абстракцию доступа к данным, управление транзакциями и инкапсуляцию деталей хранения. Использует асинхронные возможности SQLAlchemy для эффективной работы с базой данных без блокировки основного потока выполнения;

4) Analysis. Модуль для анализа шахматных партий. Реализует различные стратегии и алгоритмы для оценки позиций, поиска ошибок, выявления важных моментов и предоставления рекомендаций. Поддерживает как традиционные методы анализа, так и подходы на основе машинного обучения;

5) Video. Модуль для обработки видео. Предоставляет функциональность для работы с видеозаписями шахматных партий, включая загрузку, обработку и сегментацию. Использует библиотеки для обработки видео (например, FFmpeg) через Python-обертки.

Более подробно можно ознакомиться на рисунке 2.7. На нем представлена диаграмма второго уровня архитектуры по нотации C4.

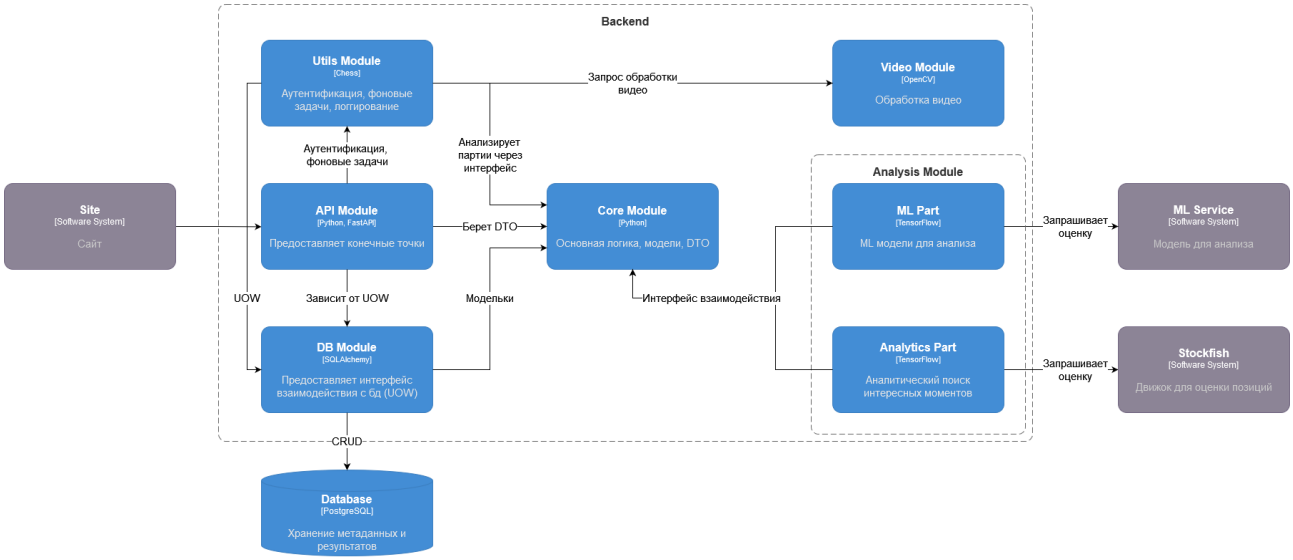

Рисунок 2.7 - Третий уровень архитектуры

2.6.2 Структура базы данных

В ходе размышлений был придуман такой дизайн:

1) User: Пользователи системы с ролями. Хранит учетные данные, настройки и связи с другими сущностями. Включает механизмы безопасного хранения паролей с использованием хеширования;

2) Game: Шахматные партии с PGN-данными. Содержит метаданные партии (название, событие, дата, игроки) и полную запись ходов в формате PGN. Связана с пользователем-владельцем, видеозаписями, важными моментами и задачами анализа;

3) Highlight: Важные моменты в партиях. Представляет собой отрезок партии (от начального до конечного хода), который содержит интересную или поучительную ситуацию. Включает описание и информацию о том, как был обнаружен этот момент (аналитически или с помощью ИИ);

4) Video: Видеозаписи партий. Хранит ссылки на оригинальные и обработанные видеофайлы, а также информацию о статусе обработки. Связана с конкретной партией и может содержать несколько сегментов;

5) VideoSegment: Сегменты видео. Представляет собой фрагмент видеозаписи, соответствующий определенному отрезку партии. Содержит временные метки начала и конца, а также ссылку на файл сегмента. Может быть связан с важным моментом;

6) Task: Фоновые задачи для анализа, обработки и т.д. Отслеживает состояние длительных операций, таких как анализ партий или обработка видео. Содержит информацию о типе задачи, статусе, стратегии анализа и сообщениях об ошибках;

7) Log: Системные логи. Хранит записи о событиях в системе, включая действия пользователей, системные операции и ошибки. Классифицирует логи по типам и связывает их с пользователями.

Подробнее можно ознакомиться на рисунке 2.8:

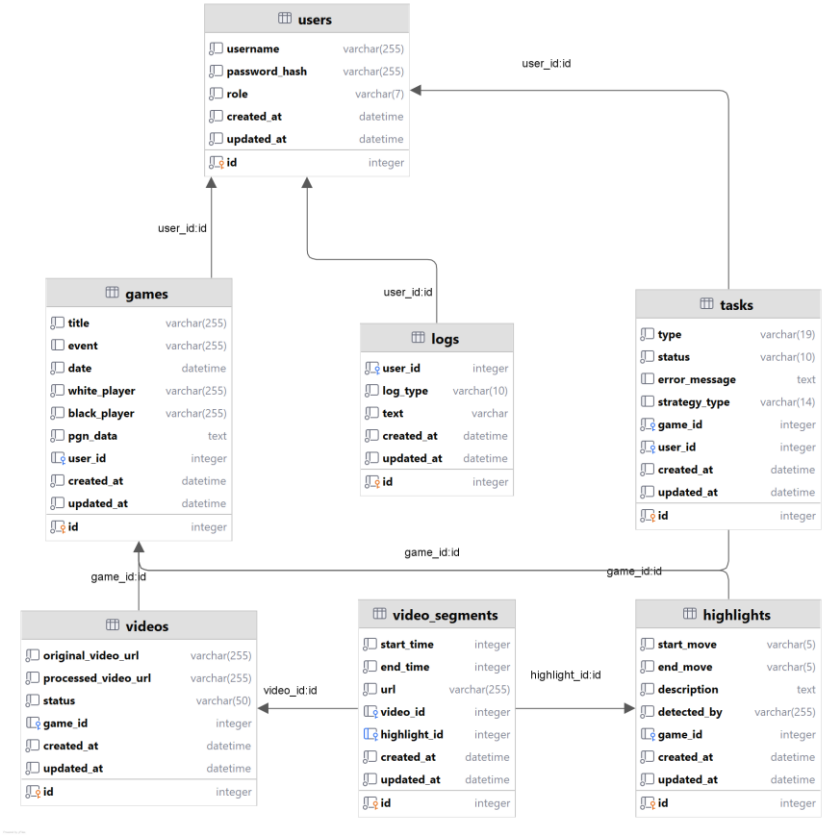

Рисунок 2.8 - Схема базы данных

2.6.3 Выбор библиотек

FastAPI — это современный, высокопроизводительный веб-фреймворк для создания API с Python 3.7+, основанный на стандартных аннотациях типов Python. FastAPI предоставляет автоматическую валидацию данных, сериализацию, документацию (с Swagger UI и ReDoc) и многое другое. Используется как основной фреймворк для создания API. FastAPI выбран из-за его высокой производительности, простоты использования, автоматической генерации документации API и встроенной поддержки асинхронного программирования. Это делает его идеальным для создания современных веб-приложений.

Uvicorn — это легкий, быстрый ASGI-сервер, реализованный на uvloop и httptools. Он предназначен для запуска асинхронных веб-приложений и фреймворков, таких как FastAPI. Используется для запуска FastAPI приложения. Uvicorn выбран из-за его высокой производительности и совместимости с FastAPI. Он обеспечивает асинхронную обработку запросов, что значительно повышает производительность приложения.

SQLAlchemy — это SQL-инструментарий и ORM (Object-Relational Mapping) для Python. Он предоставляет полный набор хорошо известных шаблонов корпоративного уровня, разработанных для эффективного и высокопроизводительного доступа к базе данных. Используется для взаимодействия с базой данных. SQLAlchemy выбран из-за его гибкости, мощности и широкой поддержки различных баз данных. Он позволяет работать с базами данных на высоком уровне абстракции, что упрощает разработку и поддержку кода. Также SQLAlchemy поддерживает асинхронность.

Pydantic — это библиотека для валидации данных и управления настройками, использующая аннотации типов Python. Pydantic-Settings — это расширение Pydantic для работы с настройками приложения. Используется для валидации данных и управления конфигурацией приложения. Pydantic-Settings используется для создания классов настроек и загрузки конфигурации из файла config.toml. Pydantic выбран из-за его интеграции с FastAPI и мощных возможностей валидации данных. Pydantic-Settings упрощает управление конфигурацией приложения, позволяя загружать настройки из различных источников.

Loguru — это библиотека для логирования в Python, которая стремится сделать логирование простым и приятным. Она предоставляет красивый, удобный и готовый к использованию API для логирования. Также предоставляет гибкие настройки для всех аспектов логирования. Используется для логирования в приложении. Loguru выбран из-за его простоты использования, богатого функционала и красивого форматирования логов. Он значительно упрощает отладку и мониторинг приложения.

AsyncPG — это асинхронный драйвер для PostgreSQL, оптимизированный для скорости и эффективности. AioSQLite — это асинхронная библиотека для работы с SQLite. Используется для подключения к базам данных. SQLite во время разработки, PostgreSQL - во время продакшена. Эти библиотеки выбраны для обеспечения асинхронного доступа к базам данных, что согласуется с асинхронной природой FastAPI и Uvicorn. Это позволяет приложению эффективно обрабатывать множество запросов одновременно.

Passlib — это библиотека для хеширования и проверки паролей. Bcrypt — это алгоритм хеширования паролей. Python-Jose — это библиотека для работы с JSON Web Tokens (JWT). Используются для аутентификации и авторизации пользователей.

Эти библиотеки выбраны для обеспечения безопасной аутентификации и авторизации в приложении. Bcrypt обеспечивает надежное хеширование паролей, а JWT (реализованный через Python-Jose) — стандартный механизм для создания токенов доступа.

HTTPX — это полнофункциональный HTTP-клиент для Python 3, который предоставляет синхронный и асинхронный API. Он поддерживает HTTP/1.1 и HTTP/2. Используется для выполнения HTTP-запросов к внешним API или сервисам. HTTPX выбран из-за его поддержки асинхронных запросов, что согласуется с асинхронной природой FastAPI. Он также предоставляет более современный и удобный API по сравнению с традиционными библиотеками, такими как requests.

NumPy — это фундаментальный пакет для научных вычислений в Python. OpenCV-Python — это библиотека компьютерного зрения, которая предоставляет инструменты для обработки и анализа изображений и видео. Используются для обработки и анализа видео, связанных с играми или другим контентом. NumPy выбран из-за его эффективных структур данных и алгоритмов для работы с многомерными массивами. OpenCV выбран из-за его богатого набора функций для обработки изображений и компьютерного зрения.

Chess — это библиотека для работы с шахматами в Python. Она предоставляет функции для представления шахматной доски, ходов, проверки правил и т.д. Используется для работы с шахматными играми и движком stockfish. Chess выбран из-за его полной реализации правил шахмат и удобного API для работы с шахматными играми. Это позволяет сосредоточиться на бизнес-логике приложения, а не на реализации правил шахмат.

G4F (GPT for Free) — это библиотека для взаимодействия с различными моделями искусственного интеллекта, включая GPT, без необходимости использования платных API. Используется для анализа шахматной партии - один из способов ана-

лиза. G4F выбран для интеграции возможностей искусственного интеллекта в приложение без необходимости использования платных API. Это позволяет снизить затраты на разработку и эксплуатацию приложения.

Python-Multipart — это библиотека для парсинга multipart/form-data, которая часто используется для загрузки файлов через HTTP. Используется для обработки загрузки файлов через API. Python-Multipart выбран из-за его совместимости с FastAPI и простоты использования для обработки загрузки файлов.

Pytest — это фреймворк для написания и запуска тестов в Python. Он упрощает написание небольших тестов, но также может масштабироваться для поддержки сложных функциональных тестов. Используется для написания и запуска тестов для приложения. Pytest выбран из-за его простоты использования, богатого функционала и широкой поддержки в сообществе Python. Он позволяет писать более читаемые и поддерживаемые тесты по сравнению с стандартной библиотекой unittest.

Выбор библиотек в проекте отражает современный подход к разработке веб-приложений с использованием Python. FastAPI и Uvicorn обеспечивают высокопроизводительную основу для API, SQLAlchemy и асинхронные драйверы баз данных обеспечивают эффективную работу с данными, а Pydantic упрощает валидацию данных и управление конфигурацией. Библиотеки для безопасности (Passlib, Python-Jose) обеспечивают надежную аутентификацию и авторизацию, а специализированные библиотеки (Chess, NumPy, OpenCV, G4F) добавляют функциональность, специфичную для домена приложения.

Все эти библиотеки хорошо интегрируются друг с другом и следуют современным практикам разработки, таким как асинхронное программирование и строгая типизация. Это делает проект более поддерживаемым, производительным и безопасным.

2.6.4 Разработка

Основной модуль приложения, содержащий всю логику бэкенда для шахматной аналитической платформы. Этот модуль организован по принципу чистой архитек-

туры, разделяя код на слои с четкими зонами ответственности. Он обеспечивает асинхронную обработку запросов, что позволяет эффективно обрабатывать множество одновременных соединений и длительные операции анализа шахматных партий.

2.6.4.1 app.core

Начали с ядра приложения. Здесь хранятся модели и абстрактные базовые классы. По файлам:

1) models.py: Определения моделей данных с использованием SQLAlchemy для объектно-реляционного отображения (ORM);

2) DTO.py: Объекты передачи данных для API. Определяет структуры данных для входящих запросов и исходящих ответов, обеспечивая четкий контракт между клиентом и сервером. Использует Pydantic для валидации данных, документирования и сериализации/десериализации. Включает модели для аутентификации, управления пользователями, игр, анализа и задач;

3) analysis_base: Базовые классы для функциональности анализа. Определяет абстрактные интерфейсы и базовые реализации для различных стратегий анализа шахматных партий. Использует паттерн "Стратегия" для обеспечения гибкости и расширяемости системы анализа.

Паттерн Стратегия — это поведенческий паттерн проектирования, который позволяет выбирать реализацию алгоритма во время выполнения программы. Он определяет семейство алгоритмов, инкапсулирует каждый из них и делает их взаимозаменяемыми в контексте объекта. Этот паттерн позволяет алгоритму изменяться независимо от клиентов, которые его используют.

Основные компоненты паттерна Стратегия:

1) Интерфейс Стратегии: Абстрактный класс или интерфейс, который определяет общий метод(ы), который должны реализовать все конкретные стратегии;

2) Конкретные Стратегии: Классы, реализующие интерфейс стратегии, каждый из которых предоставляет различную реализацию алгоритма;

3) Контекст: Класс, который содержит ссылку на объект стратегии и делегирует ему алгоритмическое поведение.

Преимущества паттерна Стратегия:

1) Гибкость: Приложение может переключаться между различными алгоритмами анализа во время выполнения;

2) Расширяемость: Новые стратегии могут быть добавлены без изменения существующего кода (Принцип открытости/закрытости);

3) Изоляция: Каждый алгоритм инкапсулирован в своем собственном классе, что делает их более легкими для тестирования и поддержки;

4) Устранение условных операторов: Вместо использования сложных условных операторов для выбора поведения, паттерн использует полиморфизм.

Как этот паттерн реализован в коде?

1) abstract_strategy.py

Этот файл определяет абстрактный базовый класс, который определяет интерфейс, который должны реализовать все конкретные стратегии — метод analyze, который принимает данные шахматной партии в формате PGN и возвращает результаты анализа.

В кодовой базе реализовано несколько конкретных стратегий, каждая с различным подходом к анализу шахматных партий.

2) chess_analyzer.py. Реализует класс ChessAnalyzer, который служит контекстом. Он:

a. Содержит ссылку на объект стратегии;

b. Предоставляет способ изменения стратегии во время выполнения через сеттер;

c. Делегирует фактическую работу по анализу текущей стратегии;

d. analysis_interface.py: Определяет общий интерфейс для всех стратегий анализа, включая методы для инициализации, выполнения анализа и получения результатов. Им и пользуются остальные части проекта.

2.6.4.2 app.db

Рассмотрим модуль взаимодействия с базой данных.

Паттерн Repository — это шаблон проектирования, который выступает посредником между доменным слоем и слоем доступа к данным. Он предоставляет абстракцию слоя данных и централизует обработку доменных объектов.

Ключевые характеристики:

1) Абстракция: Репозитории абстрагируют базовые механизмы хранения данных (базы данных, веб-сервисы и т.д.) от остальной части приложения;

2) Инкапсуляция: Они инкапсулируют логику, необходимую для доступа к источникам данных;

3) Разделение ответственности: Репозитории отделяют бизнес-логику от логики доступа к данным;

4) Ориентация на домен: Они позволяют работать с доменными сущностями, а не с таблицами базы данных или структурами данных.

Преимущества:

1) Тестируемость: Упрощает модульное тестирование, позволяя создавать мок-объекты репозиториев;

2) Сопровождаемость: Изменения в логике доступа к данным изолированы в репозиториях;

3) Гибкость: Позволяет переключаться между различными источниками данных с минимальным влиянием на бизнес-логику;

4) Согласованность: Обеспечивает единообразный способ доступа к данным во всем приложении.

В проекте репозиторий реализован в файле repository.py. Он предоставляет абстрактный базовый класс и конкретные реализации для доступа к различным сущностям в базе данных. Инкапсулирует логику запросов и манипуляций с данными, скрывая детали ORM от бизнес-логики. Поддерживает сложные запросы, фильтрацию, сортировку и пагинацию. Далее в каждом файле сделан репозиторий для своей модели (app.db.crud):

1) game.py: Операции с играми. Реализует создание, чтение, обновление и удаление шахматных партий. Включает специализированные методы для поиска партий по различным критериям, импорта из PGN и экспорта в различные форматы;

2) highlight.py: Операции с важными моментами. Управляет созданием, получением и обновлением важных моментов в партиях. Поддерживает связывание моментов с видеосегментами и поиск моментов по партиям или характеристикам;

3) task.py: Операции с задачами. Обрабатывает создание задач, обновление их статуса и получение информации о выполнении. Включает методы для фильтрации задач по типу, статусу и пользователю;

4) user.py: Операции с пользователями. Реализует регистрацию, аутентификацию и управление профилями пользователей. Включает методы для проверки учетных данных, обновления ролей и управления связанными сущностями;

5) video.py: Операции с видео. Управляет записями о видеофайлах, включая создание, обновление статуса и связывание с партиями. Поддерживает поиск видео по различным критериям;

6) video_segment.py: Операции с сегментами видео. Обрабатывает создание, получение и обновление сегментов видеозаписей. Включает методы для связывания сегментов с важными моментами и поиска сегментов по временным меткам.

Паттерн Unit of Work отслеживает изменения, внесенные в объекты в рамках бизнес-транзакции, и координирует запись этих изменений.

Ключевые характеристики:

1) Управление транзакциями: управляет транзакциями базы данных, чтобы гарантировать, что все операции в рамках бизнес-транзакции либо успешно завершаются, либо полностью отменяются;

2) Отслеживание изменений: отслеживает изменения (вставки, обновления, удаления), которые происходят во время бизнес-транзакции;

3) Координация репозиториев: координирует работу нескольких репозиториев для поддержания согласованности данных;

4) Атомарные операции: обеспечивает, чтобы группа связанных операций рассматривалась как единое целое.

Преимущества:

1) Целостность данных: гарантирует, что связанные изменения нескольких объектов сохраняются или откатываются как единая операция;

2) Производительность: может оптимизировать операции с базой данных путем пакетной обработки изменений;

3) Простота: упрощает управление транзакциями в приложении;

4) Согласованность: обеспечивает сохранение базы данных в согласованном состоянии.

В проекте unit of work реализован в файле unit_of_work.py. Он предоставляет абстрактный базовый класс AbstractUnitOfWork и его наследника SQLAlchemyUnitOfWork, который, в свою очередь, предоставляет доступ к репозиториям в контексте текущей транзакции.

Паттерны Repository и Unit of Work часто используются вместе:

1) Репозитории предоставляют методы для получения и манипулирования доменными объектами;

2) Unit of Work координирует работу нескольких репозиториев и управляет транзакциями.

В типичном рабочем процессе:

1) Приложение получает экземпляр Unit of Work;

2) Unit of Work предоставляет доступ к репозиториям;

3) Приложение использует репозитории для получения и изменения доменных объектов;

4) Когда изменения завершены, Unit of Work фиксирует транзакцию, сохраняя все изменения одновременно;

5) Если возникают ошибки, Unit of Work откатывает транзакцию, отменяя все изменения.

Такая комбинация создает чистую, поддерживаемую архитектуру, которая разделяет ответственность и обеспечивает согласованность данных.

Как было сказано выше, именно unit of work дает доступ к репозитоиям и управляет сессией базы данных. Осталось еще два файла:

1) connection.py: Настройка подключения к базе данных. Создает и управляет пулом соединений с базой данных, обеспечивая эффективное использование ресурсов. Поддерживает различные типы баз данных (PostgreSQL, MySQL, SQLite) через конфигурацию. Включает механизмы повторных попыток подключения и обработки ошибок соединения;

2) dependencies.py: Внедрение зависимостей для доступа к базе данных. Определяет функции-зависимости FastAPI для предоставления компонентов доступа к данным (репозитории, unit of work) в обработчики маршрутов. Обеспечивает автоматическое управление жизненным циклом соединений и транзакций.

2.6.4.3 app.api

Модуль, отвечающий за API-интерфейс приложения. Содержит маршруты и middleware для обработки HTTP-запросов. Реализован с использованием FastAPI, что обеспечивает автоматическую валидацию данных, документацию OpenAPI и высокую производительность.

app.api.routes: содержит определения всех API-маршрутов, организованных по функциональным областям:

1) auth.py: Аутентификация и авторизация пользователей. Включает эндпоинты для регистрации, входа в систему, обновления токенов и выхода. Использует JWT-токены для безопасной аутентификации и поддерживает различные уровни доступа на основе ролей пользователей (пользователь, менеджер, администратор);

2) profile.py: Управление профилями пользователей. Предоставляет эндпоинты для просмотра и редактирования профилей, изменения пароля и управления настройками пользователя. Поддерживает загрузку аватаров и настройку предпочтений анализа;

3) game_content.py: Управление контентом игр. Обрабатывает запросы, связанные с дополнительными материалами к шахматным партиям, такими как комментарии, аннотации и учебные материалы. Позволяет добавлять, редактировать и удалять контент, связанный с конкретными партиями;

4) games_managment.py: Управление играми. Предоставляет эндпоинты для создания, чтения, обновления и удаления шахматных партий. Поддерживает фильтрацию, сортировку и пагинацию при получении списка партий. Обрабатывает загрузку PGN-файлов и экспорт партий в различные форматы;

5) analysis.py: API для анализа шахматных партий. Содержит эндпоинты для запуска различных типов анализа (традиционный, ML-анализ), получения результатов анализа и управления настройками анализа. Поддерживает как синхронный, так и асинхронный режимы анализа;

6) tasks.py: Управление фоновыми задачами. Предоставляет API для создания, мониторинга и управления длительными операциями, такими как анализ партий, обработка видео и выявление важных моментов. Поддерживает отмену задач, получение статуса и обработку ошибок.

app.api.middlewares: Промежуточное ПО для обработки запросов:

1) LoggingMiddleware: Регистрирует все входящие запросы и исходящие ответы, включая метаданные запроса, время выполнения и коды состояния. Это помогает в отладке, мониторинге производительности и аудите безопасности. Настраиваемые уровни детализации логирования позволяют контролировать объем собираемой информации;

2) Дополнительные middleware для обработки ошибок, проверки аутентификации и ограничения скорости запросов обеспечивают надежность и безопасность API.

2.6.4.4 app.analysis

analytics: Традиционные алгоритмы анализа шахматных партий. Использует шахматные движки (например, Stockfish) для оценки позиций и поиска оптимальных ходов. Включает:

a) engine_strategy.py;

b) position_analyzer.py;

c) mistake_detector.py;

d) opening_classifier.py;

e) interface.py : реализует наследника AbstractAnalysisStrategy из app.core для аналитического анализа.

ml: Анализ на основе машинного обучения. Использует как модели стронних организаций, так и модель собственной разработки для оценки позиций, классификации стилей игры и выявления паттернов. Включает, собственно, запросы к моделями

и реализацию интерфейсов, все так же наследников. AbstractAnalysisStrategy из app.core.

fake_strategy.py: Мок-стратегия для тестирования. Имитирует работу стратегий анализа без фактического выполнения вычислений. Используется в тестах, разработке и демонстрациях для быстрого получения предсказуемых результатов. Поддерживает настройку задержек и типов результатов для имитации различных сценариев.

2.6.4.5 app.video

В этом модуле содержатся следующие файлы:

1) cut.py: Функциональность для нарезки видео на сегменты;
2) download.py: Функциональность для загрузки видео. Включает:

a. Загрузку видео из различных источников (локальные файлы, URL, YouTube);

b. Проверку и валидацию видеофайлов;

c. Извлечение метаданных (разрешение, длительность, кодеки);

d. Предварительную обработку (нормализация формата, оптимизация размера);

e. Управление хранилищем видеофайлов (организация, очистка).

2.6.4.6 app.utils

Файл authentication.py содержит утилиты для аутентификации и реализует следующий функционал:

1) Создание и проверку JWT-токенов;
2) Хеширование и проверку паролей;
3) Управление сессиями пользователей;
4) Декораторы и зависимости для защиты маршрутов;
5) Интеграцию с внешними системами аутентификации (OAuth, LDAP).

Файл logging.py содержит утилиты для логирования и предоставляет:

1) Настройку логгеров с различными уровнями детализации;
2) Форматирование и маршрутизацию логов (консоль, файлы, внешние системы);
3) Контекстные менеджеры для отслеживания выполнения операций.

Файл helpers.py содержит вспомогательные асинхронные функции для анализа шахматных игр и обработки видеозаписей этих игр. В файле реализованы две основные функции:

1) run_analysis - функция для запуска анализа шахматной игры. Использует интерфейс для выбора нужного метода анализа

2) run_video_cut - функция для нарезки видеозаписей на основе найденных в анализе ключевых моментов игры. Ее реализация оказалась не простой из-за того, что партия может храниться в нескольких видео. Рассмотрим ее подробнее:

- Функция проверяет статус задачи анализа и ждет, пока она не будет завершена;

- Статус задачи нарезки видео устанавливается в "PROCESSING" (обработка);

- Получение информации об игре и связанных с ней видеозаписях (и проверка наличия этих самых видеозаписей. Также получаем моменты для нарезки;

- Для каждого видео извлекаются временные метки из имени файла. В итоге создается список видеозаписей с информацией о начальном и конечном времени. Потом видеозаписи сортируются по времени начала;

- Из PGN-данных игры извлекаются временные метки каждого хода;

- Для каждого ключевого момента находятся соответствующие сегменты видео. Близкие или перекрывающиеся сегменты объединяются

- Выполняется нарезка и объединение видеосегментов;

- Создается запись о видеосегменте в базе данных;

- После успешной обработки всех ключевых моментов, статус задачи устанавливается в "COMPLETED" (завершено);

2.6.4.7 app.config.py

Конфигурация приложения с использованием Pydantic для валидации и типизации настроек обеспечивает загрузку конфигурации из файлов TOML, переменных окружения и других источников. Это позволяет централизованно управлять настройками приложения с проверкой типов и значений по умолчанию.

FastAPISettings включает конфигурацию для FastAPI, содержащую список разрешенных CORS-источников для безопасного взаимодействия с фронтендом.

DatabaseSettings содержит конфигурацию подключения к базе данных, включая строку подключения с учетными данными и параметрами, а также настройки пула соединений, такие как размер, время жизни и таймауты.

SecuritySettings включает настройки безопасности, такие как секретный ключ для подписи JWT-токенов, алгоритм шифрования, время жизни токенов доступа и параметры их обновления.

AnalysisSettings содержит конфигурацию для модуля анализа, включая стратегию анализа по умолчанию и путь к шахматному движку.

2.6.4.8 app.main.py

Точка входа в приложение отвечает за настройку и запуск FastAPI-приложения, подключение middleware, маршрутов и систем логирования. Она обеспечивает корректную инициализацию всех компонентов приложения и обработку сигналов завершения.

Основные функции включают создание экземпляра FastAPI с настройками из конфигурации и настройку CORS для безопасного взаимодействия с фронтендом. Также подключается middleware для логирования, аутентификации и обработки ошибок, регистрируются все маршруты API из различных модулей и настраивается система логирования. Инициализируется подключение к базе данных, и сервер запускается с помощью Uvicorn с настраиваемыми параметрами, такими как хост, порт и количество воркеров.

2.6.4.9 Отказ от sqlmodel

В начале разработки проекта мы рассматривали различные инструменты для организации работы с базой данных. SQLModel от тианголо, разработчика FastAPI, привлек внимание как решение, объединяющее возможности SQLAlchemy и Pydantic. Основная идея библиотеки заключается в устранении дублирования кода: вместо создания отдельных классов для моделей ORM и схем валидации данных, SQLModel позволяет использовать единый класс.

Изначальное решение использовать SQLModel было продиктовано желанием упростить кодовую базу. При стандартном подходе с SQLAlchemy и Pydantic приходится поддерживать минимум три класса для каждой сущности (модель БД, входящая

и исходящая схемы данных), что создает избыточность и увеличивает вероятность рассинхронизации.

Однако по мере развития проекта начали проявляться ограничения SQLModel:

1) Недостаток гибкости при использовании расширенных возможностей SQLAlchemy. Когда потребовалось применить специфичные настройки столбцов, пришлось обращаться к параметру sa_column_kwargs даже для простых свойств, как nullable, что негативно сказалось на типизации и читаемости кода;

2) Сложности при реализации абстрактных базовых классов для репозиториев. При попытке создать ABC для унификации доступа к данным возникли многочисленные ошибки типизации. Проблема связана с тем, что SQLModel использует собственную систему типов, которая не всегда корректно взаимодействует со стандартными механизмами Python, особенно при использовании дженериков и протоколов. В итоге полсотни ошибок типизации очень раздражали.

В результате было принято решение вернуться к проверенному сочетанию SQLAlchemy и Pydantic. Несмотря на некоторое дублирование кода, этот подход обеспечил более предсказуемое поведение, лучшую поддержку IDE и инструментов статического анализа, а также позволил использовать все возможности SQLAlchemy без обходных путей.

Тем не менее, стоит отметить, что SQLModel остается перспективной библиотекой для более простых проектов, где не требуется глубокая кастомизация на уровне БД и сложная асинхронная логика.

2.6.5 Тестирование

2.6.5.1 Тесты для базы данных

В рамках проекта было критически важно обеспечить надежное взаимодействие с базой данных. Поскольку архитектура приложения строилась на паттернах Repository и Unit of Work, требовалась тщательная проверка их функциональности. Для организации тестирования был выбран фреймворк pytest как оптимальное решение, предоставляющее гибкие возможности для асинхронного тестирования и управления тестовыми данными.

Прежде всего, для изоляции тестов от рабочей базы данных была настроена in-memory SQLite база. Это позволило создавать чистое тестовое окружение для каждого запуска без загрязнения основной БД.

Особое внимание было уделено асинхронной природе приложения, что потребовало использования специализированных фикстур pytest-asyncio. Для корректной работы с асинхронными сессиями SQLAlchemy были созданы фикстуры для предоставления сессий и экземпляров UnitOfWork.

Реализация тестов репозиториев включала проверку всех основных операций CRUD:

1) Создание записей - проверка корректного сохранения данных и генерации ID;
2) Получение записей - как по ID, так и с применением фильтров;
3) Обновление - корректность применения изменений и их сохранения;
4) Удаление - проверка успешного удаления записей.

Тесты UnitOfWork были направлены на проверку транзакционной логики:

1) Инициализация репозиториев - проверка доступности всех необходимых репозиториев;
2) Фиксация транзакций - тестирование commit с подтверждением сохранения данных;
3) Откат транзакций - проверка rollback функциональности;
4) Обработка исключений - автоматический откат при возникновении исключений.

Особенно важным был тест обработки исключений, гарантирующий целостность базы данных при сбоях.

2.6.5.2 Применение HTTP-языка JetBrains в процессе тестирования API

В ходе разработки активно использовался инструмент HTTP-язык от JetBrains. Данный инструмент представляет собой специализированный язык для описания и тестирования HTTP-запросов, интегрированный в среды разработки JetBrains. Основное преимущество данного подхода заключается в возможности документирования

запросов к API прямо в коде, что позволяет поддерживать тесную связь между документацией и реальной имплементацией.

На этапе разработки FastAPI-сервиса для анализа шахматных партий HTTP-язык применялся для верификации ключевых компонентов системы. В частности, были протестированы следующие элементы:

1) Механизмы авторизации и аутентификации пользователей;
2) Полный набор CRUD-операций для управления записями шахматных партий;
3) API-эндпоинты запуска и мониторинга процессов анализа;
4) Корректность возвращаемых данных при получении интересных моментов игры.

Использование HTTP-файлов позволило значительно ускорить цикл разработки за счет быстрой проверки работоспособности API без необходимости создания полноценных интеграционных тестов на ранних этапах. Важным аспектом стала возможность сохранения истории запросов и их параметров, что упростило отладку при изменениях в контрактах API.

2.6.6 Итоговая архитектура

2.6.6.1 Архитектура приложения

Приложение построено на основе FastAPI и использует следующие архитектурные паттерны:

1) Repository: Для абстракции доступа к данным. Каждый тип сущности имеет соответствующий репозиторий, который инкапсулирует логику доступа к данным и скрывает детали ORM от бизнес-логики. Это обеспечивает гибкость при изменении способа хранения данных и упрощает тестирование;
2) Unit of Work: Для управления транзакциями. Обеспечивает атомарность операций с базой данных, группируя несколько операций в одну транзакцию. Автоматически управляет началом, фиксацией и откатом транзакций, что упрощает обработку ошибок и обеспечивает целостность данных;

3) Dependency Injection: Для внедрения зависимостей. Использует систему зависимостей FastAPI для предоставления компонентов (репозитории, сервисы, конфигурация) в обработчики маршрутов. Это упрощает тестирование, улучшает модульность и уменьшает связанность компонентов;

4) Strategy: Для различных стратегий анализа шахматных партий. Позволяет выбирать и заменять алгоритмы анализа во время выполнения без изменения клиентского кода. Поддерживает традиционные алгоритмы на основе шахматных движков и подходы на основе машинного обучения;

5) Factory: Для создания объектов. Используется для создания экземпляров стратегий анализа, репозиториев и других компонентов на основе конфигурации или параметров запроса;

6) Middleware: Для обработки запросов. Реализует сквозную функциональность, такую как логирование, аутентификация и обработка ошибок, которая применяется ко всем запросам без дублирования кода в обработчиках маршрутов.

2.6.6.2 Функциональность приложения

Приложение предоставляет следующую функциональность:

1) Аутентификация и авторизация пользователей:

a. Регистрация новых пользователей с валидацией данных;

b. Вход в систему с использованием имени пользователя и пароля;

c. Аутентификация на основе JWT-токенов с настраиваемым временем жизни;

d. Обновление токенов доступа с использованием refresh-токенов;

e. Контроль доступа на основе ролей (пользователь, менеджер, администратор);

f. Защита маршрутов API с проверкой аутентификации и авторизации.

2) Управление шахматными партиями:

a. Загрузка партий в формате PGN с валидацией и нормализацией данных;

b. Просмотр партий с метаданными (игроки, событие, дата) и последовательностью ходов;

c. Редактирование метаданных и аннотаций к партиям;

d. Поиск и фильтрация партий по различным критериям (игроки, даты, результаты).

3) Анализ шахматных партий:

a. Оценка позиций с использованием шахматных движков (например, Stockfish);

b. Выявление ошибок, неточностей и упущенных возможностей;

c. Предложение альтернативных ходов и вариантов;

d. Анализ стиля игры и характеристик партии с использованием машинного обучения;

e. Классификация дебютов и сравнение с теоретическими вариантами;

f. Визуализация результатов анализа с графиками оценок и ключевыми моментами.

4) Обработка видеозаписей партий:

a. Загрузка видеозаписей из Яндекс Диска;

b. Извлечение метаданных и предварительная обработка видео;

c. Нарезка видео на сегменты, соответствующие определенным моментам партии;

d. Применение эффектов и аннотаций к видеосегментам (замедление, выделение, текст);

e. Объединение сегментов в новые видеофайлы с настраиваемыми параметрами.

5) Выделение важных моментов в партиях:

a. Асинхронное выполнение длительных операций (анализ партий, обработка видео);

b. Отслеживание статуса и прогресса выполнения задач;

c. Уведомления о завершении задач через веб-интерфейс;

d. Обработка ошибок и автоматические повторные попытки;

e. Отмена и приостановка выполняющихся задач.

Приложение использует асинхронный подход с SQLAlchemy для работы с базой данных, что обеспечивает высокую производительность и масштабируемость.

RESTful API, реализованный с помощью FastAPI, предоставляет четко документированный интерфейс для взаимодействия с фронтендом, включая автоматическую валидацию данных и генерацию документации OpenAPI.

2.7 Программная реализация клиентской части

Работая над проектом, передо мной стояла цель создать удобный в использовании и визуально привлекательный сайт. Первый разработанный дизайн стал важным шагом в понимании принципов интерфейсного проектирования. После анализа аудитории, функциональных задач и современных трендов я пришла к выводу, что полученный результат не пригоден для использования в проекте и требует доработки. Далее я опишу причины отказа от первой версии, расскажу, чего именно не хватало тому дизайну и поделюсь личными выводами о процессе создания интерфейса.

Одной из первых и важных задач для нашей команды стал выбор названия для нашего проекта. Далее я расскажу, как мы пришли к такому названию.

После долгих обсуждений и рассмотрения десятков различных вариантов, мы пришли к названию HiChess. Здесь «Hi» означает приветствие, знак открытости и дружелюбия. Также «Hi» созвучно с главной задачей проекта – просмотром хайлайтов шахматных партий. «Chess» — тема проекта. Объединив их, мы хотели передать идею сообщества, где каждый может поделиться своими хайлайтами, получить совет и просто наслаждаться игрой, учиться и развиваться в приятной атмосфере.

Логотип с силуэтом пешки подсказывает тему проекта, а кроме того, напоминает латинскую букву в названии.

2.7.1 Дизайн-концепция

Основные цвета — оттенки серого, белый. Так было сделано для дальнейшего выбора цветов. Серые тона определяют насыщенность будущих выбранных оттенков. Было принято решение использовать яркие акцентные градиенты близких на цветовом круге оттенков для наглядности статуса обработки загруженного хайлайта: «В обработке», «Завершено», «Отменено». Но как оказалось такое решение было не лучшей идеей, ведь пользователя интересуют подобные статусы только в момент за-

грузки очередной партии, и при дальнейшем использовании такие статусы только мешались бы и отвлекали от основной цели использования сайта. Рассмотрим плюсы, минусы и возможные улучшения первоначального дизайна:

1) Плюсы:
    a. Простота восприятия;
    b. Минимализм.
2) Минусы:

a. Монотонность. Серый фон не передает энергии шахматных баталий, а также скучен и не вызывает интереса и желания дальнейшего использования;

b. Оттенки не соответствуют привычному представлению о шахматах (шахматы ассоциируются с черно-белым полем, деревом).

3) Возможные улучшения:

a. Ввести черно-белую гамму как основу, добавив зеленый для акцентов, а также золотистый и оттенки для отрисовки шахматной доски. Это сделает картинку живее и натуральнее, а также будет лучше ассоциироваться с шахматами;

b. Использовать яркие акценты только для уведомлений или ошибок, появляющихся во время взаимодействия пользователя с интерфейсом.

Известно, что при создании веб-интерфейса использование черного в большинстве случаев является дурным тоном. По этой причине, при создании дизайна сайта для нашего проекта, я избегала черного, и все элементы, включая шрифты, окрашены в темно-серые тона. Выбор шрифта в этот раз был не самой сложной задачей. Были выбраны простые, приятные глазу и знакомые многим шрифты.

1) Плюсы:
    a. Читаемость;
    b. Соответствие минимальным требованиям доступности.
2) Минусы:
    a. Отсутствие уникальности. Шрифты не связаны с тематикой шахмат;
    b. Низкая контрастность.
3) Возможные улучшения:

a. Для большей привязанности к теме проекта можно было бы выбрать шрифты с «шахматной» эстетикой (например, serif-шрифты);

b. Увеличить контрастность: темный текст на светлом фоне, жирные заголовки.

Принято при рисовании макета использовать специально нарисованные именно для этого проекта иконки в едином стиле, с одинаковой толщиной линий. На начальных этапах мне стало интересно самой создать иконки для проекта. Я посмотрела, как это можно сделать, установила все необходимое и попробовала нарисовать несколько. У меня получилось нарисовать сет иконок, но в итоге было принято решение использовать уже готовый сет, открытый для использования в некоммерческих проектах.

В текущей версии дизайна не был описан внешний вид самих шахматных фигур, что является большим упущением, ведь это самые главные “фигуры” проекта.

1) Минусы:

a. Нет иконок шахматных фигур

b. Некоторые иконки недостаточно наглядные (например, иконка плей не указывает на видео).

2) Возможные улучшения:

a. Для лучшей связи с тематикой можно заменить стандартные иконки на шахматные аналоги (например, ферзь вместо сердца, шахматная доска вместо документа). Идея довольно смелая, но есть риск не быть правильно понятым пользователями, так что этот вариант навряд ли будет применен к новой версии дизайна.

b. Необходимо подумать над иконками шахматных фигур.

c. Добавить визуальные метафоры (например, часы для таймлайна, флаги стран для межнациональных матчей).

Когда я работала над созданием первой версии макета, я еще не до конца понимала, какие элементы вообще нужны в веб-интерфейсе и тем более не понимала, как они должны располагаться и выглядеть. Таким образом, я упустила крайне важный элемент - шахматную доску. Рассмотрим детально каждую получившуюся страницу. На рисунке 2.9 изображена страница просмотра партии.

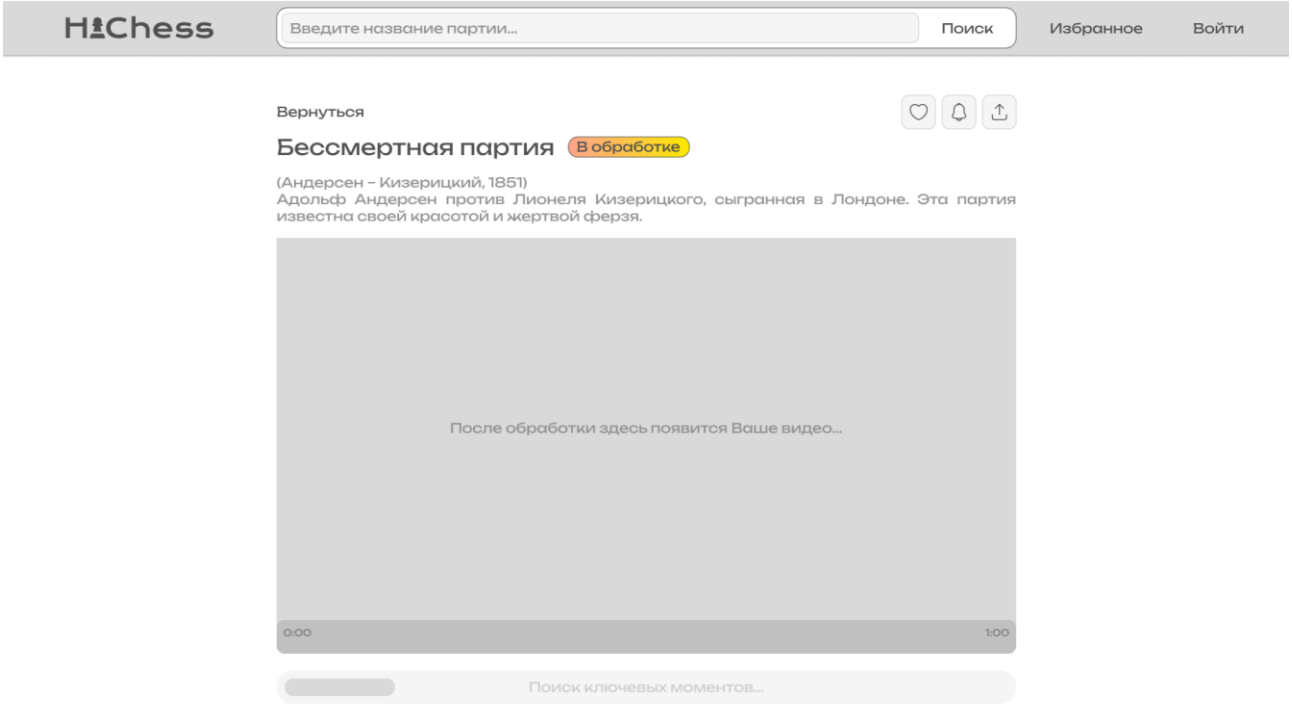

Рисунок 2.9 – Страница просмотра партии (старый дизайн)

Одним из недостатков является отсутствие обратной связи, из-за чего пользователь не видит, что будет происходить после загрузки видео. Также отсутствуют визуальные подсказки, такие как мини-доски для отображения текущего хода. Необходимо реализовать шахматную доску и продумать внешний вид просмотра партии.

Возможные улучшения включают добавление временного слоя с мини-доской, которая будет менять позицию фигур в реальном времени. Также можно включить прогресс-бар обработки видео с оценкой оставшегося времени.

На рисунке 2.10 мы видим страницу просмотра загруженных партий

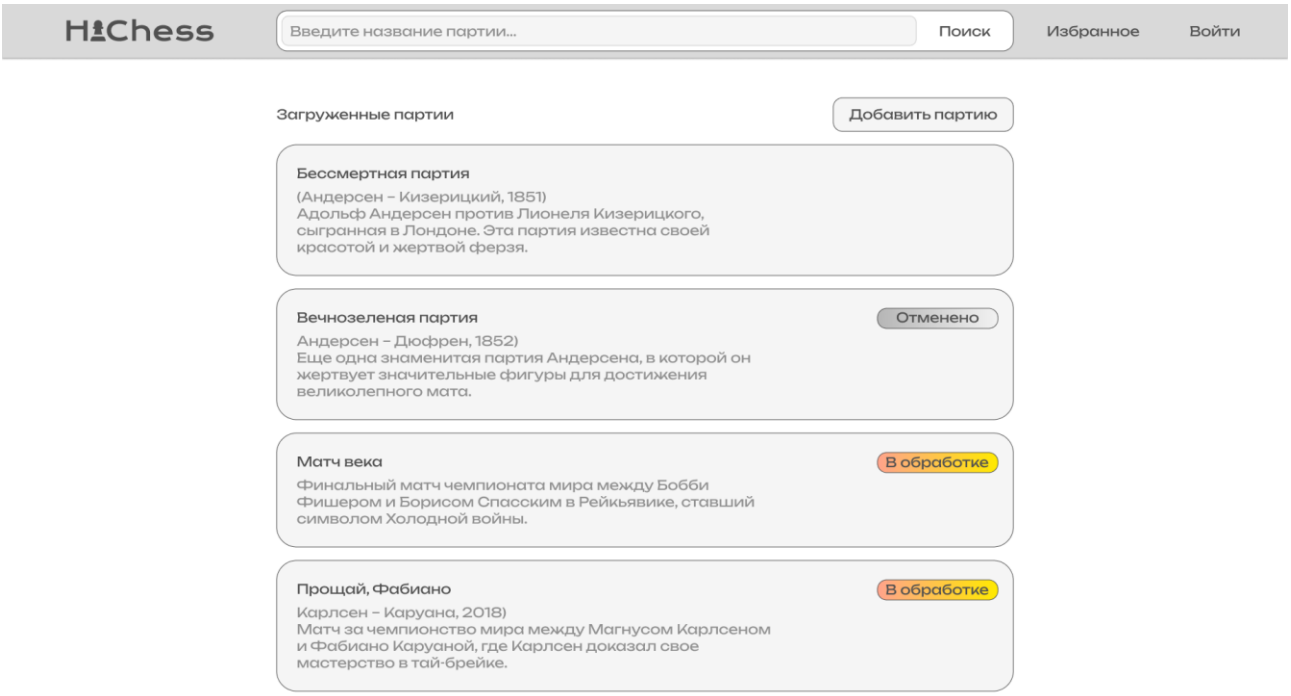

Рисунок 2.10 – Страница просмотра загруженных партий при наличии партий (старый дизайн)

На рисунке 2.11 изображена страница просмотра загруженных партий при отсутствии партий.

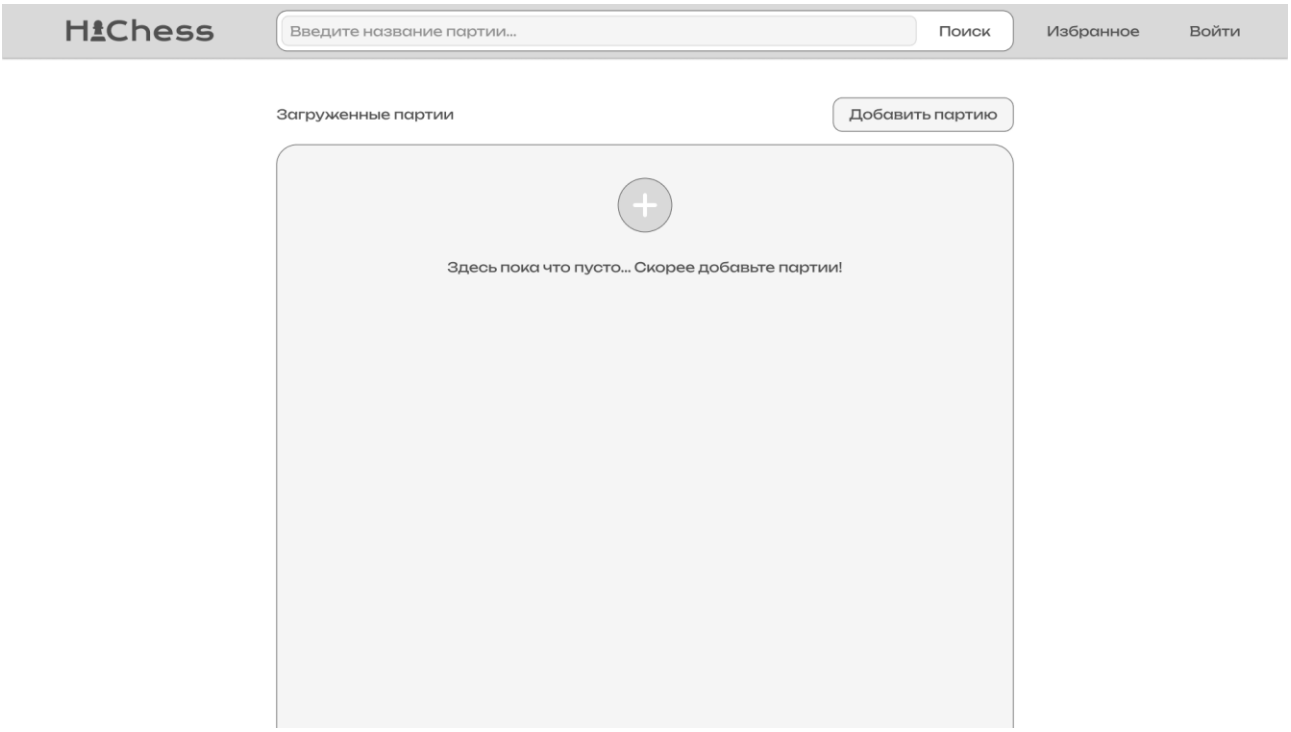

Рисунок 2.11 – Страница просмотра загруженных партий при отсутствии партий (старый дизайн)

Одним из недостатков является отсутствие мотивации для пользователя, так как сообщение «Здесь пока что пусто…» звучит как провал.

Возможные улучшения включают добавление примеров популярных партий с возможностью быстрой загрузки. Также можно добавить подбадривающее сообщение, которое сподвигнет пользователя к загрузке партии. Кроме того, можно визуализировать процесс добавления партии через анимацию.

Страница загрузки файла показана на рисунке 12.

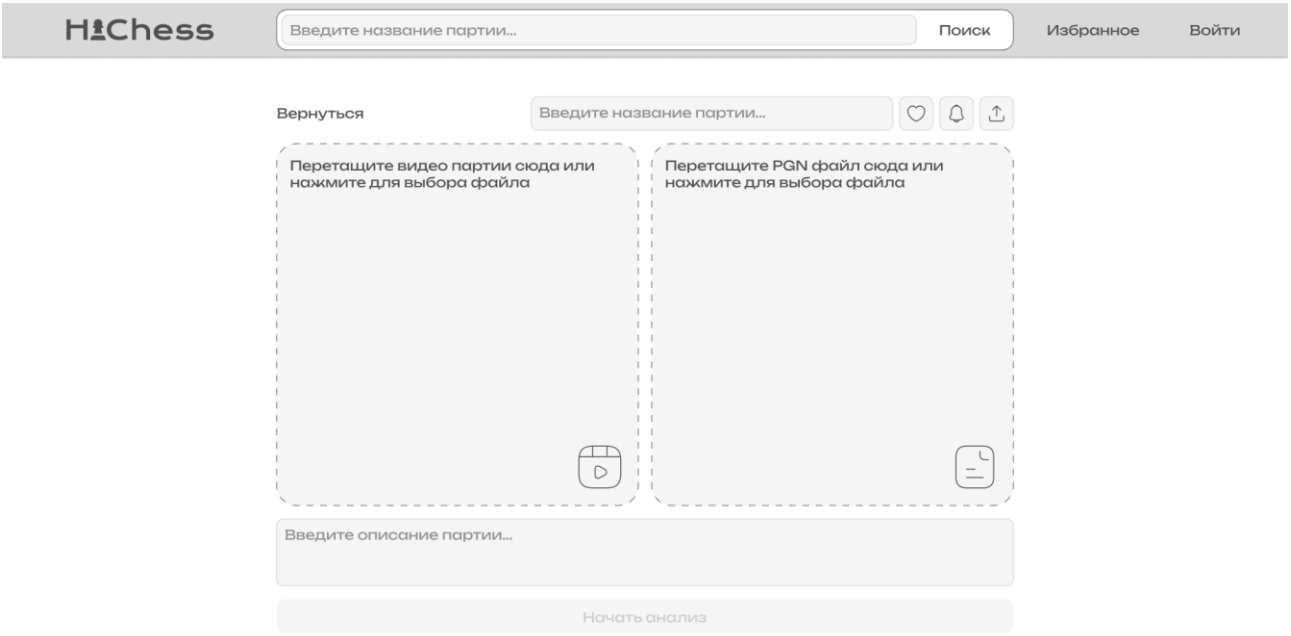

Рисунок 12 – Страница загрузки файла (старый дизайн)

Одним из недостатков является слишком громоздкий дизайн.

Возможные улучшения включают добавление графических подсказок, таких как изображение шахматной доски внутри зоны загрузки. Также можно включить подсветку зоны при наведении курсора.

При создании макета сайта удалось следовать принципам минимализма и гармонии компонентов.

Однако были нарушены принципы юзабилити, так как отсутствует обратная связь, и пользователь не знает, сколько времени осталось до завершения обработки. Также нарушен принцип эмоционального отклика, поскольку дизайн не вызывает интереса к шахматам.

Для улучшения можно добавить микроинтеракции, такие как небольшие анимированные элементы интерфейса, которые реагируют на действия пользователя, добавляя интерактивность и живость в пользовательский опыт. Например, анимация загрузки и подсветка активных элементов. Также можно включить элементы геймификации.

Почему дизайн был забракован? Во-первых, из-за низкой вовлеченности из-за отсутствия визуальных метафор. Во-вторых, из-за недостатка информации о процессе (статус «В обработке» без временных рамок). Кроме того, цветовая схема не соответствовала тематике и получилась довольно скучной и не контрастной. Главной же причиной стала недоработка и непродуманность дизайна в целом. Многие важные элементы не были описаны на макете.

Какие улучшения все же стоит принять во внимание и внедрить в дизайн? Одни из самых заметных и важных пунктов - необходимо добавить шахматную доску, просмотр ходов, корректировать цветовую гамму (ранее было предложено использовать черно-белую гамму с зелеными и золотистыми акцентами, что придаст жизни, яркости и соответствие тематике дизайну, а также поможет удерживать внимание пользователя). Необходимо также усилить обратную связь, а именно добавить прогресс-бары. Стоит поработать над шрифтами, иконками и целостностью макета. Важно будет учесть все элементы необходимые для удобной работы с функциональностью проекта

Процесс работы над первой версией показал, что даже простой дизайн требует глубокого понимания аудитории, идеи и функционала проекта. Важно не только решить задачу, но и сделать процесс взаимодействия пользователей с веб-интерфейсом приятным и вдохновляющим. Шахматы — это игра ума, и интерфейс должен отражать эту энергию через визуальные и эмоциональные сигналы, а также давать мотивацию и интерес к изучению шахмат.

После долгих обсуждений и тестирований мы пересмотрели дизайн нашего проекта HiChess. Новый интерфейс стал результатом анализа ошибок старого прототипа, изучения аналогичных платформ (например, Lichess и Chesskit) и стремления создать пространство, которое будет вдохновлять шахматистов любого уровня. Вот как все сложилось.

Основной цвет — ярко-зеленый (головная полоса, кнопки). Он символизирует дружелюбность («Hi» в названии) и движение вперед, как ходы на доске. Это отсылка к таким площадкам, как Lichess, где зеленый ассоциируется с активностью и жизнью.

Шахматная доска выполнена в классических бежево-коричневых тонах, чтобы сохранить связь с традициями и сделать процесс приятным и привычным. Акценты в виде черных фигур добавляют контраст, делая интерфейс «живым».

Мы хотели избежать монотонности старого серого дизайна. Зеленый вдохновляет, поддерживает, успокаивает, а коричневый создает уют — сочетание, которое помогает сосредоточиться на игре и не отвлекаться на чересчур яркие или бледные цветовые решения.

Как и в первой версии макета, шрифты были выбраны стандартные, не отвлекающие от главного, но все же другие, а именно: 'Segoe UI', Tahoma, Geneva, Verdana. Элементы были увеличены в размере для лучшей читаемости.

Так как для нашего проекта необходимо было использовать символы, обозначающие не только действия, но и шахматные фигуры, было принято решение использовать соответствующие unicode-символы. Я была удивлена, что там нашлись все необходимые символы, и при этом нарисованы они довольно симпатично и отличимо (в unicode присутствуют как белые, так и черные шахматные фигуры).

Макеты:

1) Страница загрузки игры представлена на рисунках 2.13 и 2.14

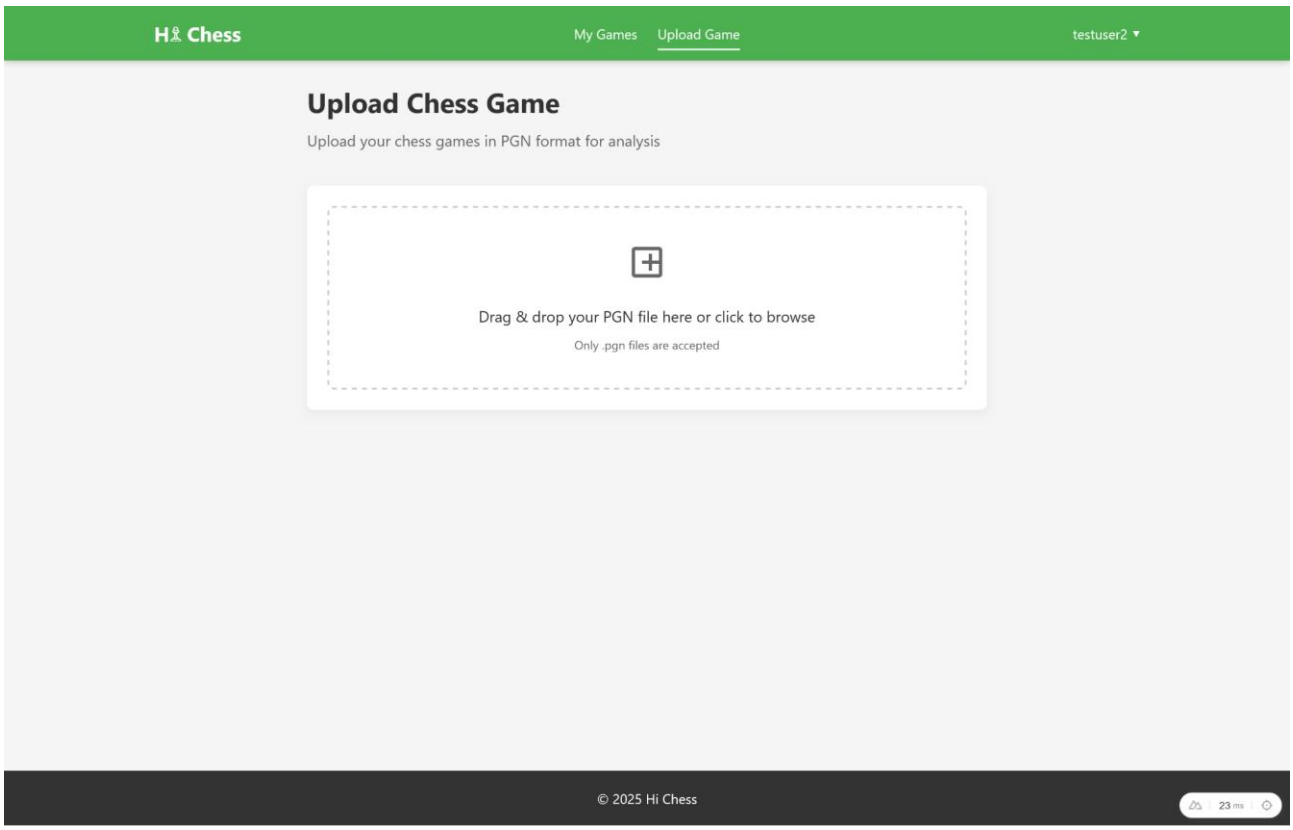

Рисунок 2.13 – Страницы загрузки игры: загрузка файла (новый дизайн)

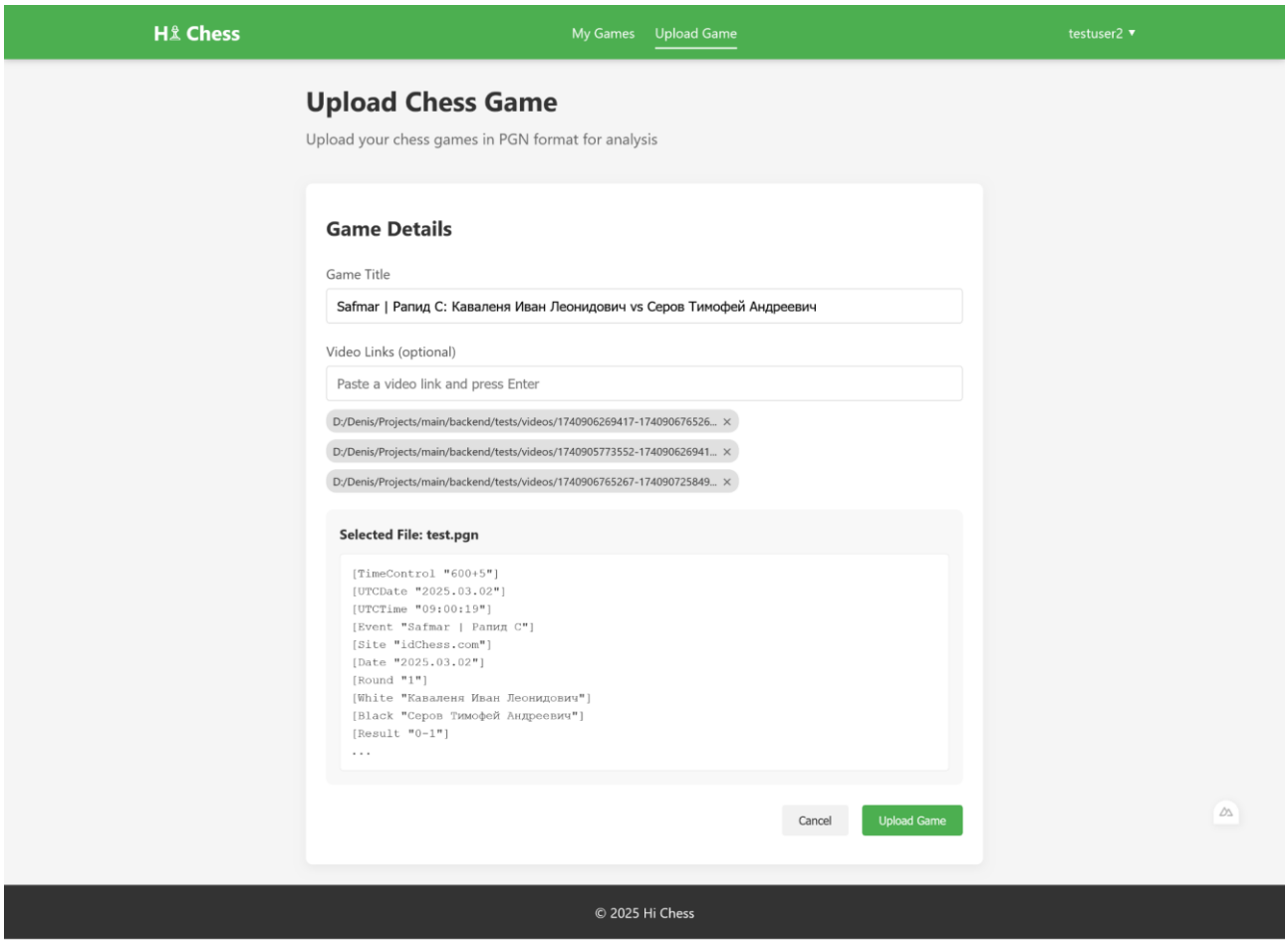

Рисунок 2.14 – Страница загрузки игры: добавление игры (новый дизайн)

На первом плане — большие поля для ввода данных (название партии, ссылки на видео). Под ними — список выбранных файлов с возможностью их удаления. Кнопка «Upload Game» выделена ярким зеленым, чтобы ее сразу же заметили и нажали.

В старом дизайне все было размыто: пустые окна, невнятные подсказки. Теперь все структурировано. Пользователь сразу понимает, что делать — даже если он видит сайт впервые.

2) Страница просмотра партии представлена на рисунках 2.15-2.19

В верхней части страницы находится информация о сражении, тэги, в том числе дата битвы, далее кнопка возвращения на страницу просмотра всех добавленных партий. Ниже визуально понятно отображены соперники и их фигуры. Доска визуализирует каждый совершенный ход, при этом выделяя его цветом, а правее указан список всех ходов и интервал интересных.

При просмотре игры, текущий ход подсвечивается на доске и в списке. Это помогает следить за событиями, не теряясь в деталях.

Видео встроено в отдельный блок с возможностью просмотра в режиме «Full screen» с таймлайном. Также был добавлен индикатор выполнения для наглядности оставшегося времени загрузки.

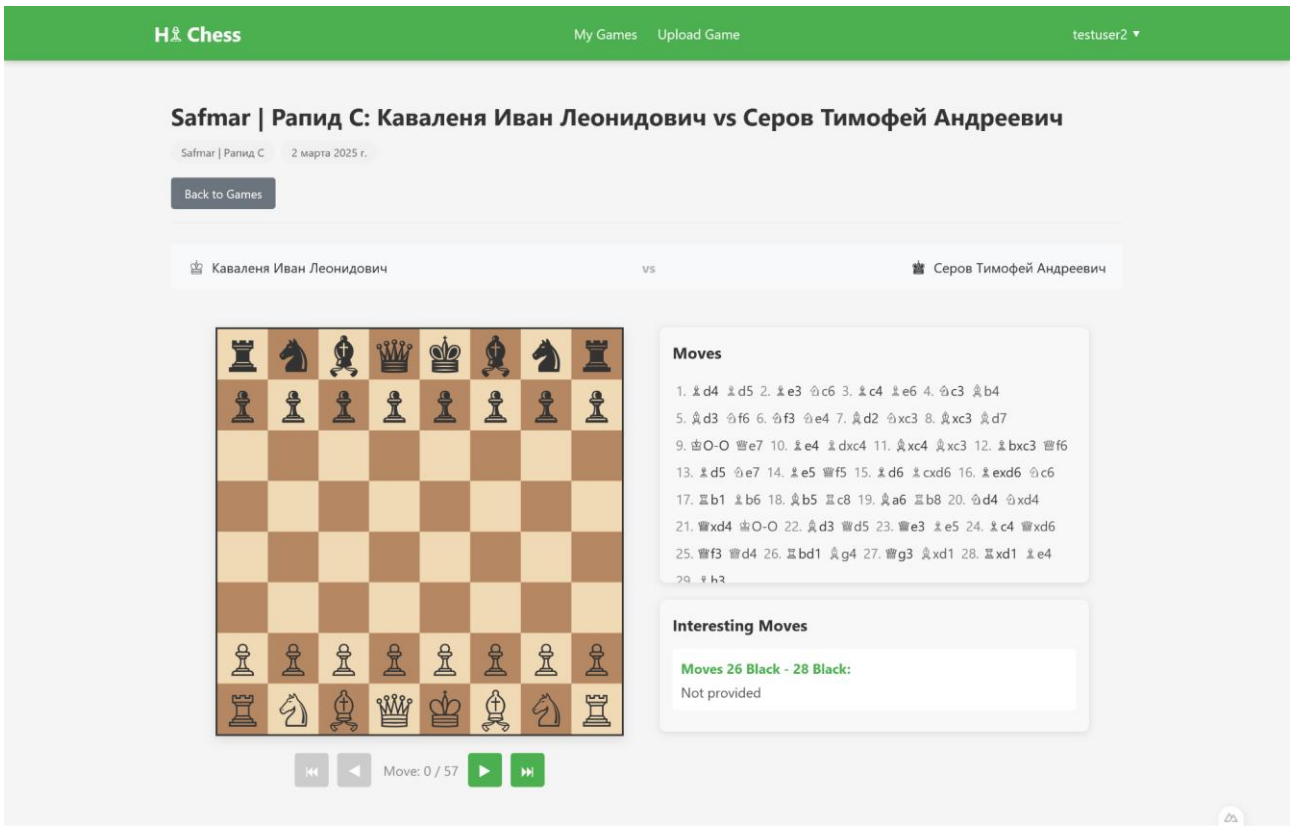

Рисунок 2.15 – Страница просмотра партии (новый дизайн)

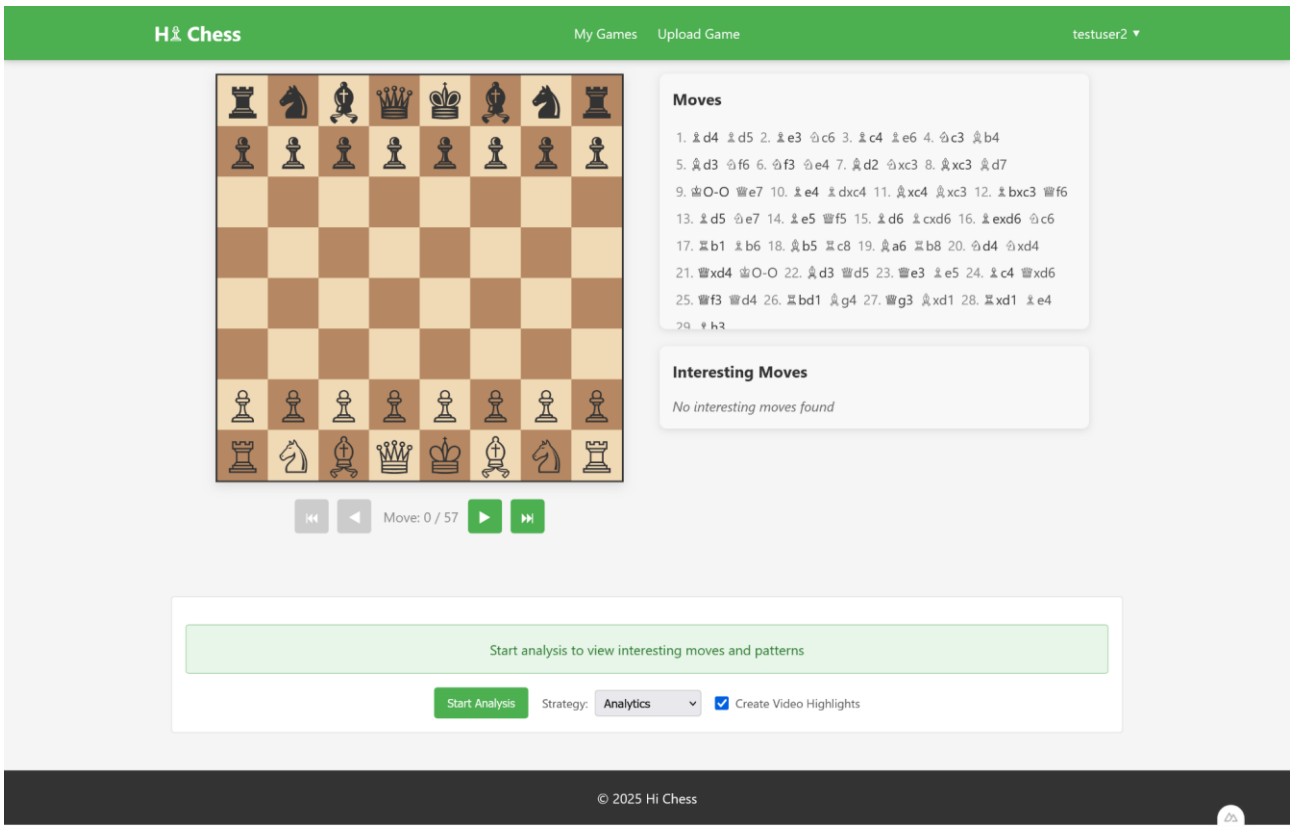

Рисунок 2.16 – Блок «Start analysis» (новый дизайн)

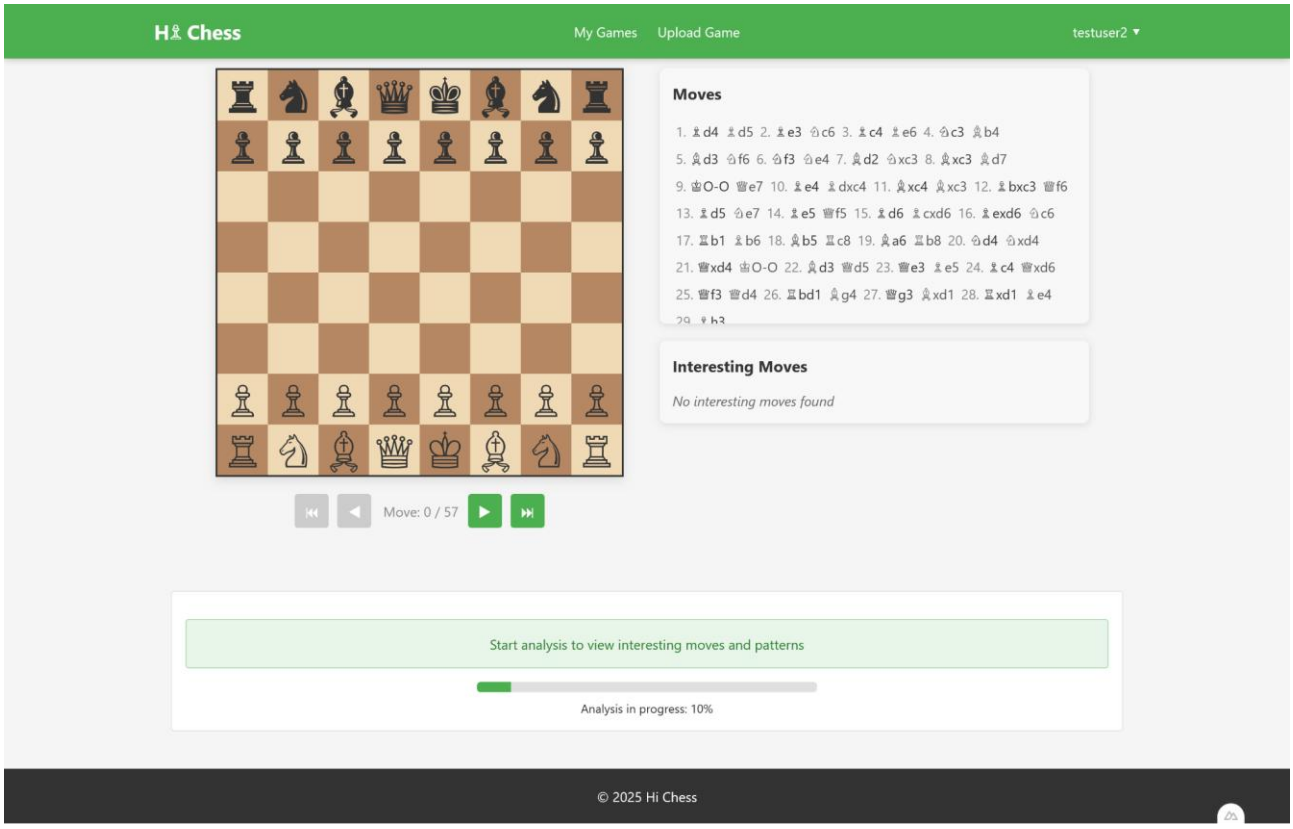

Рисунок 2.17 – Процесс анализа партии (новый дизайн)

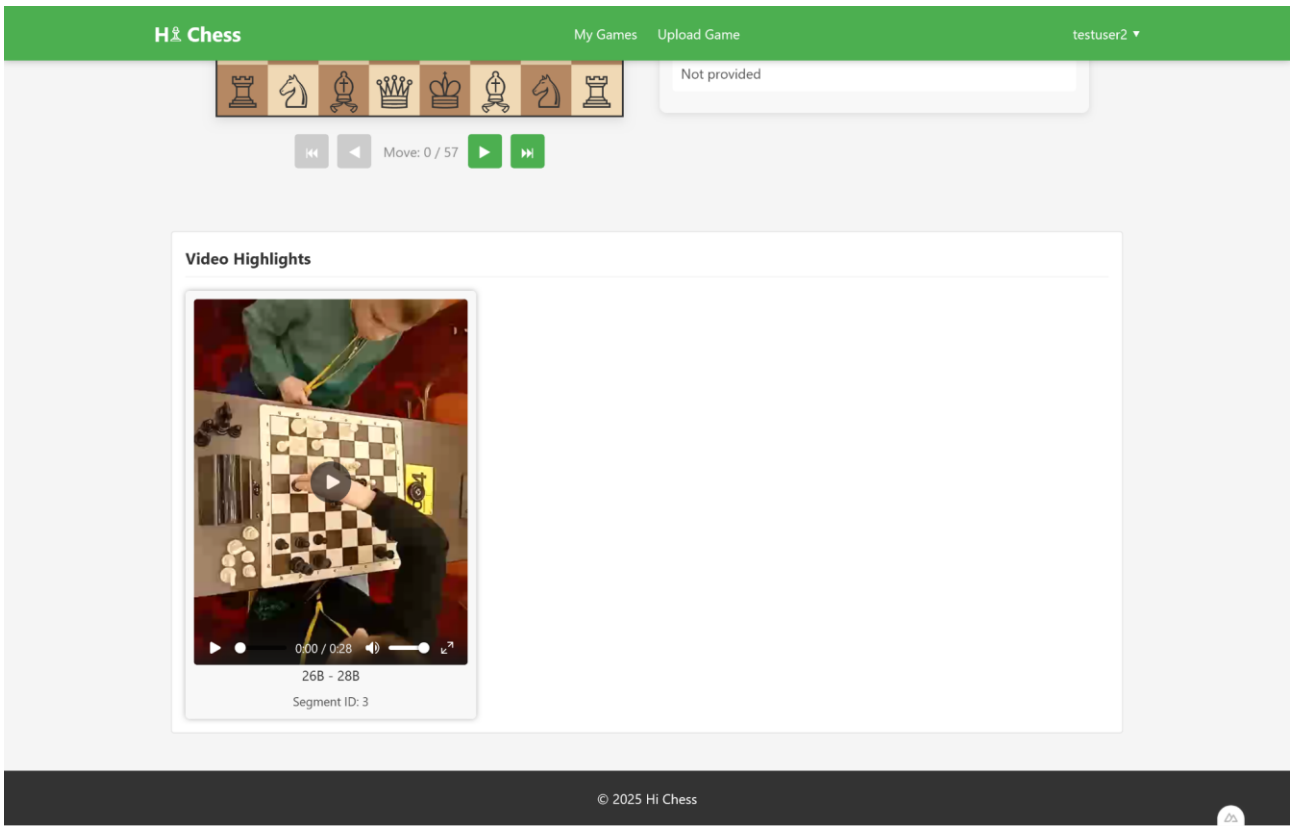

Рисунок 2.18 – Просмотр видео-хайлайтов партии (новый дизайн)

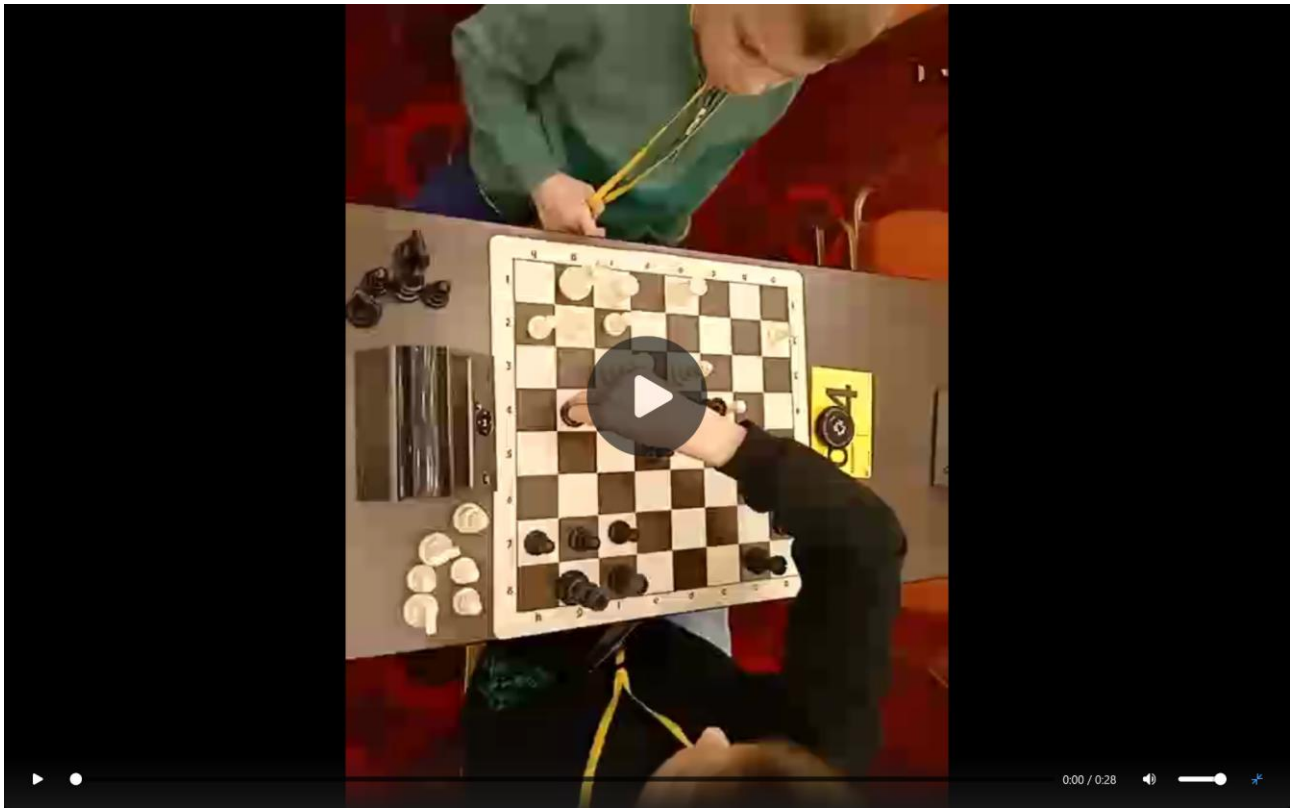

Рисунок 2.19 – Плеер

3) Страница «Мои игры»

Был изменен вид пустой страницы, а также полностью переделан вид страницы, наполненной несколькими партиями. Теперь партии располагаются не списком друг под другом, а в виде соседствующих карточек, что выглядит удобнее, компактнее и позволяет пользователю видеть большее количество игр на экране. Сами карточки дают увидеть максимальное количество информации на минимальном размере, но выглядит это уместно и не загромождает экран.

Если партий нет — появляется иконка пешки, что напоминает о теме продукта, и текст: «No games yet. Upload your first chess game to get started». Если игры есть — карточки с названиями, датами и кнопками «View Analysis»/«Delete».

Старый дизайн встречал пользователя пустым окном с надписью «Здесь пока что пусто…», что могло демотивировать. Теперь сообщение звучит мягче, призывает к совершению действия, а сама страница выглядит цельной.

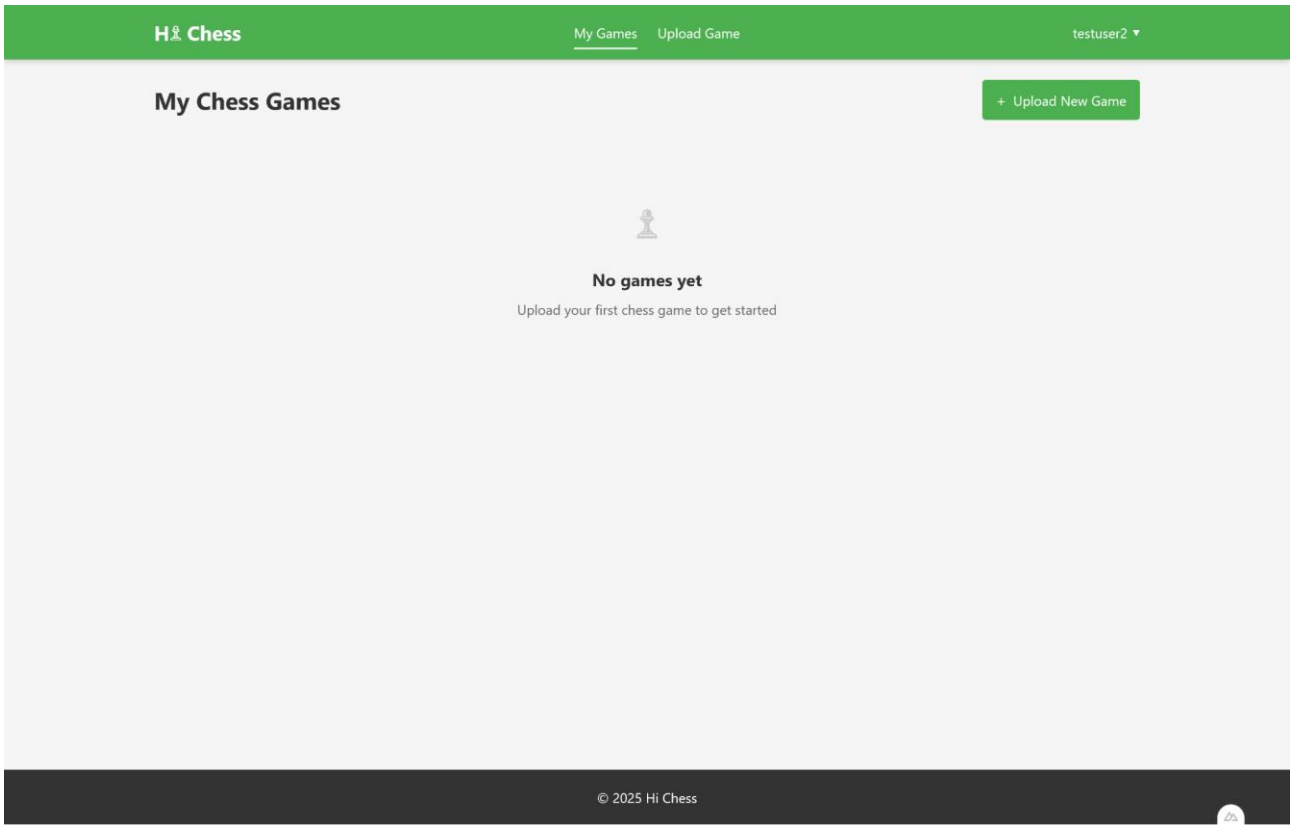

Рисунок 2.20 – Страница «Мои игры» при отсутствии партий (новый дизайн)

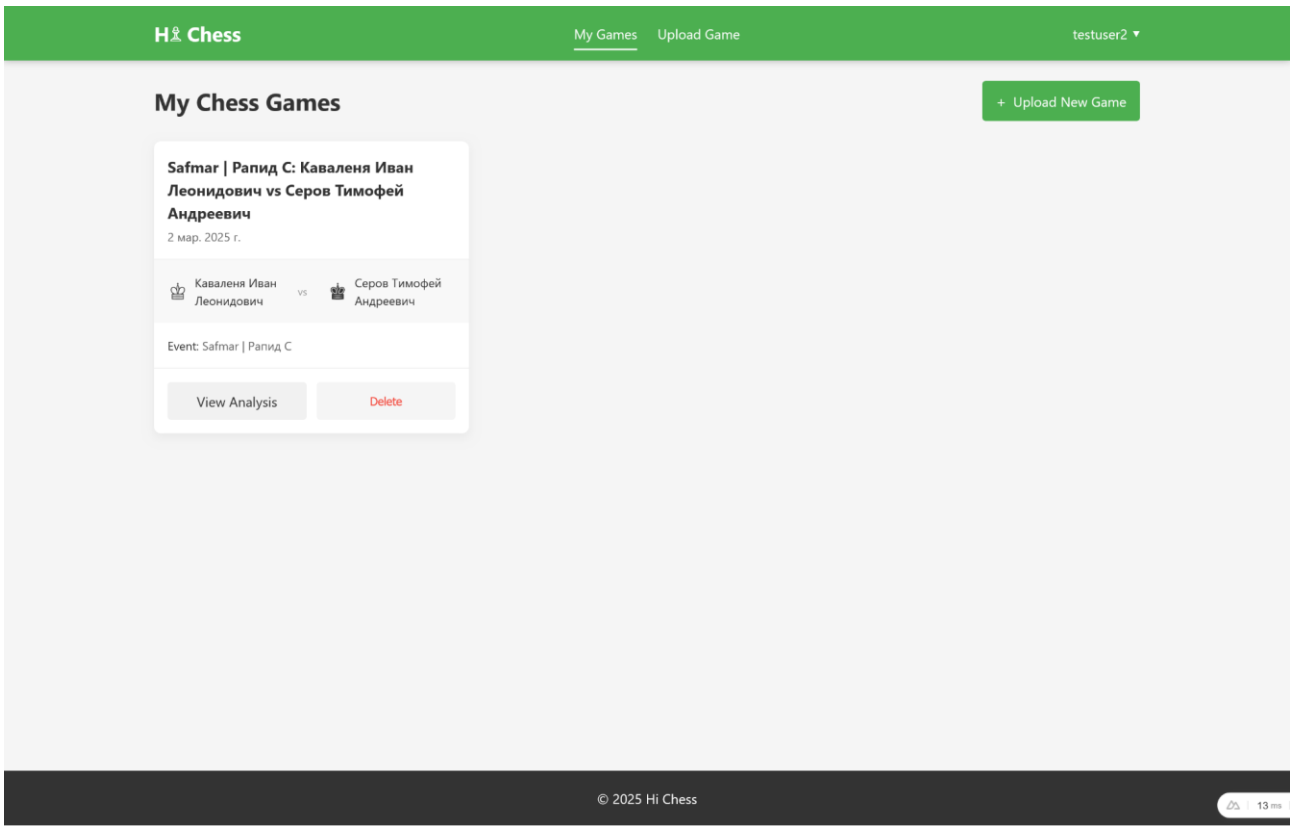

Рисунок 2.21 – Страница «Мои игры» при наличии партий (новый дизайн)

Отличия от аналогов:

1) Lichess — мощный инструмент для анализа, но его дизайн строгий, с кучей различных инструментов. Мы же сделали акцент на эмоциях: зеленый цвет, большие кнопки, понятные подсказки, более расслабленный и понятный минималистичный дизайн;

2) Chesskit — тоже технически продвинутый, но интерфейс здесь более минималистичный. У нас же появились визуальные метафоры (например, временная шкала в виде доски).

Новый дизайн HiChess — это баланс между функциональностью и эстетикой. Я сохранила ключевые элементы (загрузка PGN, просмотр видео и так далее), но сделали их более наглядными. Поработала над устранением недостатков прошлого дизайна и постаралась учесть возможные улучшения. Дизайн стал более продуманным, удобным, приятным и интуитивно понятным. Цвета вдохновляют, макеты помогают сосредоточиться, а название говорит о нашей миссии: сделать шахматы доступными и приятными.

2.7.2 Фреймворки и библиотеки проекта HiChess

Проект HiChess представляет собой современную платформу анализа шахматных партий, построенную на тщательно подобранном стеке технологий.

Основные технологии:

1) Nuxt.js 3.16.0:

a. Обеспечивает серверный рендеринг для оптимальной SEO и быстрой загрузки;

b. Автоматическая файловая маршрутизация упрощает организацию кода;

c. Модульная архитектура позволяет легко интегрировать новые функции;

d. Поддерживает как статическую генерацию (SSG), так и серверный рендеринг (SSR).

2) Vue.js 3.3.4:

a. Использует Composition API для эффективной организации логики компонентов;

b. Декларативный подход с директивами (v-if, v-for, v-model) упрощает создание интерфейсов;

c. Компонентная архитектура обеспечивает модульность и переиспользование кода;

d. Компонент ChessBoard инкапсулирует логику отображения и взаимодействия с шахматной доской.

3) Pinia 3.0.1:

a. Современное решение для управления состоянием, заменившее Vuex;

b. Отличная поддержка TypeScript из коробки;

c. Интуитивный API без сложных концепций мутаций;

d. Модульная структура для организации различных аспектов состояния;

e. Интеграция с Vue DevTools для удобной отладки.

4) Chess.js 1.0.0-beta.3:

a. Обеспечивает валидацию ходов по правилам ФИДЕ;

b. Поддерживает парсинг PGN и генерацию FEN нотации;

c. Управляет состоянием игры, включая специальные правила (рокировка, взятие на проходе);

d. Надежная и проверенная библиотека с активной поддержкой.

2.7.3 Архитектура

Перед детальным рассмотрением каждого компонента системы, важно понимать общую архитектуру frontend части Hi Chess. Проект построен на основе Nuxt.js фреймворка и следует принципам модульной организации кода.

Структура проекта отражает современные подходы к разработке веб-приложений, где каждая директория имеет четко определенную роль:

Общая структура

```
frontend/
├── components/
│   ├── ChessBoard.vue
│   └── PgnUploader.vue
├── layouts/
│   └── default.vue
├── middleware/
│   ├── auth.js
│   ├── check-auth.js
│   └── guest.js
├── pages/
│   ├── games/
│   │   ├── [id].vue
│   │   └── index.vue
│   ├── chess-board.vue
│   ├── index.vue
│   ├── login.vue
│   └── register.vue
├── services/
```

```
│   └── api.js
├── store/
│   └── auth.js
└── nuxt.config.ts
```

Архитектура следует принципу разделения ответственности: компоненты отвечают за UI логику, страницы определяют маршруты, middleware обеспечивает безопасность, а store управляет глобальным состоянием. Такая организация обеспечивает масштабируемость и поддерживаемость кода.

Каждая директория взаимодействует с другими через четко определенные интерфейсы: компоненты импортируются в страницы, middleware автоматически применяется к маршрутам, а store предоставляет реактивные данные всему приложению.

Теперь рассмотрим детально каждый элемент этой архитектуры и его роль в общей экосистеме приложения.

2.7.4 Конфигурационные файлы

2.7.4.1 package.json

Файл package.json является фундаментальным элементом любого Node.js проекта, и в случае с HiChess он содержит не только базовые метаданные, но и тщательно подобранный набор зависимостей, каждая из которых играет свою роль в общей архитектуре приложения. Проект получил название "HiChess", что четко отражает его предназначение как платформы для анализа шахматных партий.

Скрипты, определенные в package.json, образуют полный жизненный цикл разработки и развертывания приложения. Скрипт dev запускает приложение в режиме разработки с горячей перезагрузкой, что значительно ускоряет процесс разработки. Build создает оптимизированную версию для продакшена, применяя различные оптимизации, включая минификацию кода и tree-shaking. Start запускает уже собранное приложение, а generate создает статические файлы для развертывания на CDN или статических хостингах.

2.7.4.2 nuxt.config.ts

Конфигурационный файл nuxt.config.ts представляет собой центральную точку управления всем поведением приложения. Здесь определяется не только базовая информация о приложении, но и сложные взаимосвязи между различными модулями и плагинами.

Название приложения "HiChess" было выбрано не случайно - оно отражает дружелюбный и доступный подход к анализу шахматных партий. Метаданные для SEO включают подробное описание "A platform for uploading and analyzing chess games", что помогает поисковым системам правильно индексировать приложение и привлекать целевую аудитуру.

Особое внимание уделено конфигурации модулей. Модуль @pinia/nuxt интегрируется на уровне всего приложения, обеспечивая доступность хранилищ состояния во всех компонентах без необходимости дополнительной настройки. Плагин ~/services/api.js регистрируется глобально, что позволяет всем компонентам приложения использовать единый интерфейс для взаимодействия с backend API.

Настройки сборки включают transpile для chess.js, что необходимо для корректной работы этой библиотеки в браузерном окружении. Конфигурация API с базовым URL создает единую точку входа для всех API-запросов, что упрощает как разработку, так и последующее развертывание приложения.

2.7.4.3 tsconfig.json

Файл tsconfig.json, хотя и кажется простым, играет важную роль в обеспечении типобезопасности приложения. Расширение конфигурации ./.nuxt/tsconfig.json является рекомендуемым подходом для проектов на Nuxt 3, поскольку это обеспечивает автоматическую генерацию типов для маршрутов, компонентов и других элементов приложения. Это особенно важно для больших проектов, где типобезопасность помогает предотвратить множество потенциальных ошибок на этапе разработки.

2.7.5 Страницы (pages)

2.7.5.1 index.vue

Главная страница приложения представляет собой сложный интерактивный интерфейс, который служит основной точкой входа для пользователей. Эта страница

демонстрирует глубокую интеграцию между различными системами приложения, начиная от управления файлами и заканчивая взаимодействием с backend API.

Интерфейс загрузки файлов реализован с использованием современных веб-технологий drag-and-drop, что обеспечивает интуитивно понятный пользовательский опыт.

Особенностью реализации является интеграция с системой аутентификации. Страница динамически адаптируется в зависимости от статуса пользователя: для неаутентифицированных пользователей отображаются кнопки входа и регистрации, в то время как аутентифицированные пользователи получают доступ к полному функционалу загрузки и анализа партий.

Функциональность добавления видео-ссылок представляет собой уникальную особенность платформы, позволяющую пользователям связывать свои партии с видеозаписями игр для их дальнейшей нарезки системой. Эта система работает через реактивный массив ссылок, который синхронизируется с формой загрузки и передается на сервер вместе с PGN-файлом.

Предварительный просмотр PGN-файла реализован через FileReader API, что позволяет пользователям убедиться в корректности загружаемого файла до его отправки на сервер. Этот механизм также служит дополнительным уровнем валидации, поскольку пользователь может визуально проверить содержимое файла.

Взаимодействие с backend происходит через централизованный API-сервис, который обеспечивает единообразную обработку запросов, включая автоматическое добавление токенов аутентификации и обработку различных типов ответов сервера.

2.7.5.2 login.vue и register.vue

Страницы аутентификации представляют собой критически важные компоненты системы безопасности приложения. Обе страницы построены по схожей архитектуре, но каждая имеет свои особенности в плане взаимодействия с системой управления состоянием.

Страница входа в систему реализует классический flow аутентификации с использованием имени пользователя и пароля. Особенностью реализации является ин-

теграция с middleware 'guest', который автоматически перенаправляет уже аутентифицированных пользователей, предотвращая ненужные попытки повторного входа. Форма использует реактивные объекты для управления состоянием полей ввода, что обеспечивает мгновенную обратную связь с пользователем.

Процесс аутентификации включает несколько этапов: сначала данные отправляются на сервер через API-сервис, затем полученный токен сохраняется в хранилище Pinia и localStorage для обеспечения персистентности сессии. После успешной аутентификации происходит автоматическое перенаправление на страницу со списком игр, что создает плавный пользовательский опыт.

Страница регистрации расширяет базовую функциональность входа дополнительной логикой автоматического входа после успешного создания аккаунта. Это решение устраняет необходимость для пользователя выполнять два отдельных действия.

Обе страницы включают комплексную систему обработки ошибок, которая предоставляет пользователям понятную обратную связь в случае проблем с аутентификацией. Ошибки обрабатываются как на уровне клиента (валидация формы), так и на уровне сервера (неверные учетные данные, проблемы с сетью).

2.7.5.3 games/index.vue

Страница списка игр представляет собой центральный хаб для управления коллекцией шахматных партий пользователя. Эта страница демонстрирует сложное взаимодействие между системой аутентификации, API-сервисом и компонентами пользовательского интерфейса.

Архитектура страницы построена вокруг концепции реактивного списка игр, который автоматически обновляется при изменениях в коллекции пользователя. Каждая игра отображается в виде информативной карточки, содержащей ключевые метаданные: имена игроков, дату проведения партии, результат и статус анализа.

Система управления играми включает функциональность удаления партий с подтверждением действия, что предотвращает случайную потерю данных. Удаление происходит через API-запрос с последующим обновлением локального состояния, что обеспечивает консистентность данных между клиентом и сервером.

Навигация между страницами реализована через Vue Router с использованием программной навигации, что позволяет передавать дополнительные параметры и состояние между страницами.

2.7.5.4 games/[id].vue

Страница детального просмотра партии представляет собой наиболее сложный компонент приложения с точки зрения взаимодействия различных систем. Эта страница объединяет отображение шахматной доски, результаты анализа, управление видео-контентом и навигацию по ходам партии.

Динамическая маршрутизация через параметр [id] обеспечивает уникальную URL для каждой партии, что важно для SEO и возможности прямого доступа к конкретным играм. При загрузке страницы происходит сложная последовательность API-запросов: сначала загружаются базовые данные партии, затем проверяется наличие результатов анализа, и наконец, при необходимости, загружается PGN-контент.

Взаимодействие с компонентом ChessBoard происходит через систему props, где родительский компонент передает PGN-данные и результаты анализа. Эта архитектура обеспечивает четкое разделение ответственности: страница отвечает за загрузку и управление данными, а компонент доски - за их отображение и интерактивность.

Система анализа партий интегрирована на уровне страницы и включает отслеживание статуса анализа, отображение прогресса и обработку результатов. Результаты анализа включают не только текстовые комментарии, но и визуальные индикаторы на доске, что создает богатый интерактивный опыт для пользователя.

Интеграция с видео-системой позволяет отображать связанные видеоматериалы и синхронизировать их с ходами партии. Эта функциональность требует сложной координации между различными медиа-элементами и состоянием игры.

2.7.6 Компоненты (components)

2.7.6.1 ChessBoard.vue

Компонент шахматной доски объединяет сложную шахматную логику с современными возможностями веб-интерфейсов. Этот компонент служит центральным

элементом пользовательского опыта и демонстрирует глубокую интеграцию между различными технологиями.

Архитектура компонента построена вокруг библиотеки chess.js, которая обеспечивает всю шахматную логику, включая валидацию ходов, определение шаха и мата, обработку специальных правил. Компонент создает абстракцию над этой библиотекой, предоставляя интуитивно понятный интерфейс для взаимодействия с шахматными позициями.

Визуальное представление доски реализовано через систему вложенных div-элементов, каждый из которых представляет отдельную клетку. Фигуры отображаются с использованием Unicode-символов, что обеспечивает кроссплатформенную совместимость и высокое качество отображения без необходимости загрузки дополнительных графических ресурсов.

Система навигации по ходам представляет собой сложный механизм управления состоянием, который включает сохранение позиций для каждого хода, обеспечение возможности перехода к любому моменту партии и синхронизацию между визуальным представлением доски и списком ходов. Этот механизм требует тщательного управления памятью и производительностью, особенно для длинных партий.

Подсветка ходов реализована через динамическое применение CSS-классов, которые выделяют клетки, задействованные в последнем ходе. Эта система работает в реальном времени и обновляется при каждом изменении позиции на доске.

Интеграция с результатами анализа происходит через систему "интересных ходов", которые выделяются особым образом и сопровождаются текстовыми комментариями. Эта функциональность требует координации между данными анализа и визуальным представлением доски.

Адаптивный дизайн компонента обеспечивает корректное отображение на различных устройствах, от настольных компьютеров до мобильных телефонов. Это достигается через использование относительных единиц измерения и медиа-запросов CSS.

2.7.6.1 PgnUploader.vue

Компонент загрузки PGN-файлов представляет собой самостоятельную систему, которая инкапсулирует весь процесс от выбора файла до получения результатов анализа. Этот компонент демонстрирует интересный подход к архитектуре, где некоторые функции реализованы независимо от централизованного API-сервиса.

Интерфейс drag-and-drop реализован через обработку нативных событий браузера, включая dragover, dragenter, dragleave и drop. Эта система требует тщательной обработки различных состояний перетаскивания и предоставления визуальной обратной связи пользователю.

Валидация файлов происходит на нескольких уровнях: сначала проверяется MIME-тип и расширение файла, затем содержимое файла анализируется для подтверждения его соответствия формату PGN. Этот процесс включает парсинг PGN-заголовков для извлечения информации о событии, именах игроков, дате проведения партии и других важных деталях.

Процесс загрузки файла использует FormData API для создания multipart/form-data запросов, что обеспечивает эффективную передачу файлов на сервер. Компонент отслеживает прогресс загрузки и предоставляет пользователю информацию о статусе операции.

Система анализа партий интегрирована непосредственно в компонент и включает инициацию анализа, отслеживание прогресса через polling механизм и обработку результатов. Использование setInterval для периодической проверки статуса анализа обеспечивает обновление интерфейса в реальном времени.

Использование localStorage для сохранения содержимого PGN и результатов анализа создает механизм кэширования, который улучшает производительность приложения и обеспечивает доступность данных между сессиями.

2.7.7 Сервисы (services)

API-сервис представляет собой центральную точку взаимодействия между frontend и backend частями приложения. Этот сервис реализован как плагин Nuxt.js, что обеспечивает его доступность во всех компонентах приложения без необходимости дополнительного импорта.

Архитектура сервиса построена вокруг концепции единого интерфейса для всех типов API-запросов. Функция customFetch создает обертку над стандартным $fetch Nuxt.js, автоматически добавляя необходимые заголовки аутентификации и обрабатывая различные типы контента.

Система аутентификации интегрирована на уровне каждого запроса через автоматическое добавление Bearer токена из localStorage. Этот механизм обеспечивает прозрачную аутентификацию для всех защищенных эндпоинтов без необходимости ручного управления токенами в каждом компоненте.

Методы сервиса организованы по функциональным группам: аутентификация, управление играми, анализ партий и работа с видео-сегментами. Каждая группа инкапсулирует специфическую логику взаимодействия с соответствующими эндпоинтами backend API.

Обработка различных типов контента включает поддержку JSON для большинства запросов, FormData для загрузки файлов и Blob для получения бинарных данных, таких как видео-сегменты. Эта универсальность позволяет сервису обрабатывать все типы данных, используемых в приложении.

Система обработки ошибок включает автоматическое логирование проблем и предоставление структурированной информации об ошибках компонентам-потребителям. Это обеспечивает консистентную обработку ошибок во всем приложении.

2.7.8 Хранилища (store)

Хранилище аутентификации представляет собой центральную систему управления состоянием пользователя, которая координирует взаимодействие между различными компонентами приложения и обеспечивает консистентность данных аутентификации.

Архитектура хранилища построена на принципах Pinia, что обеспечивает реактивность состояния и автоматическое обновление всех компонентов при изменении данных аутентификации. Состояние включает информацию о пользователе, токен доступа и флаг аутентификации, которые используются по всему приложению.

Методы входа и регистрации инкапсулируют сложную логику взаимодействия с API, включая отправку учетных данных, обработку ответов сервера, сохранение токенов и получение профиля пользователя. Эта централизация логики аутентификации обеспечивает консистентность поведения во всех частях приложения.

Система персистентности состояния использует localStorage для сохранения токена между сессиями браузера. Это включает проверки на доступность localStorage (для SSR совместимости) и автоматическое восстановление состояния аутентификации при инициализации приложения.

Интеграция с жизненным циклом приложения обеспечивает автоматическую проверку существующей аутентификации при загрузке приложения, что создает seamless пользовательский опыт для возвращающихся пользователей.

Метод выхода включает не только очистку локального состояния, но и удаление токена из localStorage, что обеспечивает полную деаутентификацию пользователя.

2.7.9 Макеты (layouts)

Основной макет приложения представляет собой архитектурную основу, которая определяет общую структуру и поведение всех страниц приложения. Этот макет демонстрирует глубокую интеграцию с системой аутентификации и обеспечивает консистентный пользовательский интерфейс.

Шапка приложения реализует адаптивную навигационную систему, которая динамически изменяется в зависимости от статуса аутентификации пользователя. Для неаутентифицированных пользователей шапка скрыта, что создает чистый интерфейс для страниц входа и регистрации. Для аутентифицированных пользователей отображается полная навигационная панель с ссылками на основные разделы приложения.

Система пользовательского меню включает dropdown-интерфейс с дополнительными опциями, такими как профиль пользователя и выход из системы. Этот механизм требует управления состоянием видимости меню и обработки кликов вне области меню для его закрытия.

Интеграция с хранилищем аутентификации происходит через вычисляемые свойства, которые автоматически обновляют интерфейс при изменении статуса

аутентификации. Это обеспечивает реактивность навигационных элементов без необходимости ручного управления их состоянием.

Область основного контента использует slot механизм Vue.js для отображения содержимого конкретных страниц. Эта архитектура обеспечивает четкое разделение между общими элементами интерфейса и специфическим контентом страниц.

Глобальные стили CSS, определенные в макете, создают единообразный визуальный язык для всего приложения. Это включает типографику, цветовую схему, отступы и другие визуальные элементы, которые используются во всех компонентах.

Адаптивный дизайн макета обеспечивает корректное отображение на различных устройствах через использование медиа-запросов и гибких единиц измерения. Максимальная ширина контента ограничена для обеспечения оптимальной читаемости на больших экранах.

2.7.10 Взаимодействие компонентов и архитектурные решения

Архитектура приложения HiChess демонстрирует продуманный подход к организации взаимодействия между различными компонентами системы. Центральную роль в этой архитектуре играет система управления состоянием на базе Pinia, которая обеспечивает единую точку истины для критически важных данных, таких как информация об аутентификации пользователя.

Взаимодействие между страницами и компонентами происходит через несколько механизмов. Система props обеспечивает передачу данных от родительских компонентов к дочерним, что особенно важно для компонента ChessBoard, который получает PGN-данные и результаты анализа от родительской страницы. Система событий позволяет дочерним компонентам уведомлять родительские о важных изменениях состояния.

API-сервис служит единой точкой взаимодействия с backend, что обеспечивает консистентность в обработке запросов и ответов. Все компоненты используют этот сервис для взаимодействия с сервером, что упрощает поддержку и развитие приложения.

Система маршрутизации Vue Router интегрирована с middleware для обеспечения контроля доступа. Middleware auth защищает страницы, требующие аутентификации, в то время как middleware guest предотвращает доступ аутентифицированных пользователей к страницам входа и регистрации.

Особенностью архитектуры является гибридный подход к взаимодействию с API: большинство компонентов использует централизованный API-сервис, но некоторые компоненты, такие как PgnUploader, делают прямые запросы. Это решение обеспечивает баланс между консистентностью и гибкостью в зависимости от специфических требований компонентов.

2.7.11 Итоговый фронтенд

В процессе разработки фронтенд-части проекта были решены следующие ключевые задачи:

1) Создание системы аутентификации — разработка страниц регистрации и входа, интеграция с JWT-аутентификацией на бэкенде;

2) Разработка компонента загрузки PGN-файлов — реализация интерфейса для загрузки шахматных партий с поддержкой drag-and-drop и валидацией файлов;

3) Создание интерактивной шахматной доски — разработка компонента для визуализации партий с возможностью пошагового просмотра ходов;

4) Визуализация интересных моментов — реализация механизма подсветки и навигации по ключевым моментам партии, выявленным алгоритмами анализа;

5) Разработка страницы управления партиями — создание интерфейса для просмотра, управления и анализа загруженных пользователем партий;

6) Обеспечение отзывчивого дизайна — адаптация интерфейса для различных размеров экрана и устройств.

2.8 План разработки проекта «HiChess»

Календарный план разработки проекта представлен в таблице 2.5.

Таблица 2.5 — Календарный план разработки проекта «Название проекта»

Содержание работы	Результат	Начало	Окончание	Исполнитель
Анализ требований и постановка задач	Техническое задание, обоснование цели и задач	15.10.2025	25.10.2025	Шаталов М.А. (Teamlead)
Разработка алгоритма DetectForks	Описание и реализация алгоритма для детекции вилок	26.10.2025	05.11.2025	Голосов Г.С. (Аналитик)
Разработка алгоритма PinDetector	Описание и реализация алгоритма для детекции связок	06.11.2025	15.11.2025	Голосов Г.С. (Аналитик)
Разработка алгоритма DetectTrappedPieces	Описание и реализация алгоритма для детекции пойманных фигур	16.11.2025	25.11.2025	Пономарев А.А. (ML-инженер)
Разработка алгоритма Stockfish-Moments	Описание и реализация алгоритма для анализа скачков оценки	26.11.2025	05.12.2025	Ивченко М.С. (ML-инженер)

Продолжение таблицы 2.5

Разработка алгоритма DetectSacrifices	Описание и реализация алгоритма для детекции жертв	06.12.2025	15.12.2025	Голосов Г.С. (Аналитик)
Подготовка данных для LLM-решения (парсинг PGN, генерация промптов)	Готовый набор данных и шаблоны промптов для LLM	16.12.2025	25.12.2025	Агафонов А.С. (ML-инженер)
Интеграция LLM с API (Google Gemini)	Рабочий модуль для взаимодействия с LLM	26.12.2025	05.01.2026	Агафонов А.С. (ML-инженер)
Разработка ML-модели для предсказания "интересности" ходов	Обученная модель и метрики качества	06.01.2026	15.01.2026	Пономарев А.А. (ML-инженер) и Ивченко М.С. (ML-инженер)

Продолжение таблицы 2.5

Реализация обработки видеозаписи (версия 1: компьютерное зрение)	Модуль для поиска первого хода на видео	16.01.2026	25.01.2026	Мельцова В.А. (Backend)
Реализация обработки видеозаписи (версия 2: временные метки)	Модуль для синхронизации видео и PGN по меткам	26.01.2026	05.02.2026	Мельцова В.А. (Backend)
Реализация обработки видеозаписи (версия 3: асинхронная оптимизация)	Оптимизированный модуль для обработки видео	06.02.2026	15.02.2026	Мельцова В.А. (Backend)

Продолжение таблицы 2.5

Реализация обработки видеозаписи (версия 4: поддержка нескольких видеофайлов)	Модуль для работы с партиями, разделёнными на несколько видео	16.02.2026	25.02.2026	Мельцова В.А. (Backend)
Разработка архитектуры серверной части (C4, база данных)	Схема архитектуры и структура БД	26.02.2026	05.03.2026	Авраменко Д.А. (Backend)
Реализация API (FastAPI, SQLAlchemy)	Готовые endpoints для клиентской части	06.03.2026	15.03.2026	Авраменко Д.А. (Backend)
Разработка дизайн-концепции клиентской части	Макеты интерфейса и UX-прототипы	16.03.2026	25.03.2026	Лапенко К.А. (Frontend)

Продолжение таблицы 2.5

Реализация страниц и компонентов (Nuxt.js, Vue.js)	Готовый интерфейс для загрузки PGN и просмотра партий	26.03.2026	05.04.2026	Лапенко К.А. (Frontend)
Интеграция клиентской и серверной частей	Полностью работоспособное решение	06.04.2026	15.04.2026	Авраменко Д.А. (Backend)
Тестирование системы (юнит-тесты, интеграционные тесты)	Отчёт о тестировании и устранённые ошибки	16.04.2026	25.04.2026	Шаталов М.А. (Teamlead)
Подготовка отчёта (документирование, оформление)	Полный отчёт по проекту	26.04.2026	10.05.2026	Шаталов М.А. (Teamlead)

План включает 20 ключевых этапов, распределённых между членами команды в соответствии с их ролями. Учтены все аспекты разработки: от алгоритмов и ML-моделей до фронтенда и тестирования.

Участники команды представлены в таблице 2.6

Таблица 2.6 — Участники команды разработки проекта «Название проекта»

Роль в команде	ФИО	Группа	Функционал
Teamlead	Шаталов М.А.	М8О-215Б-23	Координация работы, контроль сроков и качества, документация и тестирование
Backend-разработчик	Авраменко Д.А.	М8О-215Б-23	Разработка серверной части и API сервиса
ML-инженер	Агафонов А.С.	М8О-215Б-23	Разработка алгоритмов и интеграция LLM
Аналитик	Голосов Г.С.	М8О-215Б-23	Анализ требований, подготовка данных, разработка алгоритмов
ML-инженер	Ивченко М.С.	М8О-207Б-23	Разработка алгоритмов и ML-моделей
Frontend-разработчик	Лапенко К.А.	М8О-215Б-23	Разработка пользовательского интерфейса и веб приложения

Продолжение таблицы 2.6

Backend-разработчик	Мельцова В.А.	М8О-214Б-23	Разработка серверной части и API обработки видео
ML-инженер	Пономарев А.А.	М8О-213Б-23	Разработка алгоритмов и ML-моделей

Команда сбалансирована и включает всех необходимых специалистов. Роли распределены таким образом, чтобы обеспечить покрытие всех направлений разработки.

Для эффективной разработки проекта были использованы различные программные компоненты и среды, перечисленные в таблице 2.8.3.

Таблица 2.7 — Среда разработки проекта «HiChess»

Компонент среды разработки	Описание	Комментарий
Канбан-доска	Miro (https://miro.com)	Планирование и координация работы
Информационный обмен	Telegram-канал	Оперативные коммуникации
Система контроля версий	GitHub	Хранение и совместная работа над кодом

Продолжение таблицы 2.7

Среда разработки серверной части	PyCharm, FastAPI, SQLAlchemy	Разработка API и работы с базой данных
Среда разработки клиентской части	Visual Studio Code, Nuxt.js, Vue.js	Создание пользовательского интерфейса
ML-разработка	Jupyter Notebook, TensorFlow/PyTorch	Обучение и тестирование моделей
Документирование	Google Docs, LaTeX	Подготовка отчёта и технической документации

Используемые инструменты обеспечивают эффективную работу над проектом на всех этапах, от планирования до финального документирования.

2.9 Выводы по разделу 2

В ходе разработки проекта был создан комплексный инструмент для анализа и визуализации шахматных партий, объединяющий удобный веб-интерфейс с мощной аналитической системой. Фронтенд-часть предоставляет пользователям интуитивно понятные инструменты для работы с шахматными партиями, включая систему аутентификации, загрузку PGN-файлов с валидацией, интерактивную доску для пошагового просмотра и визуализацию ключевых моментов игры. Все компоненты интерфейса адаптированы для различных устройств, обеспечивая комфортную работу как на компьютерах, так и на мобильных устройствах.

Бэкенд-система построена на современной архитектуре с использованием FastAPI и реализует сложную логику обработки шахматных данных. Применение таких паттернов как Repository, Unit of Work и Dependency Injection обеспечило гибкость, масштабируемость и удобство тестирования системы. Особое внимание было

уделено безопасности - реализована надежная аутентификация на основе JWT-токенов с ролевой моделью доступа.

Аналитический модуль системы сочетает несколько подходов к оценке шахматных партий. Простые и понятные эвристики (такие как определение вилок, связок и жертв) работают в паре с глубоким анализом от шахматного движка Stockfish. Такой симбиоз позволяет с одной стороны быстро выявлять очевидные тактические моменты, а с другой - находить скрытые стратегические нюансы, требующие сложных расчетов. Алгоритм автоматически определяет ключевые моменты партии, которые затем могут быть использованы для создания обучающих материалов или анализа игры.

Отдельного внимания заслуживает разработанная система обработки видеозаписей шахматных партий. В ходе нескольких итераций был создан эффективный алгоритм синхронизации видео с ходами партии, основанный на временных метках. Решение поддерживает работу с несколькими видеофайлами одной партии и использует асинхронную обработку для оптимальной производительности. Качество результата напрямую зависит от точности исходных метаданных, поэтому система включает механизмы валидации входных данных.

Технологический стек проекта включает современные и проверенные решения: FastAPI для backend-разработки, SQLAlchemy для работы с базой данных, Pydantic для валидации, а также специализированные библиотеки для работы с шахматными данными и видео. Такой выбор обеспечил баланс между производительностью, надежностью и удобством разработки.

Итоговый продукт представляет собой законченное решение, которое может быть использовано как для самостоятельного анализа шахматных партий, так и для создания обучающих материалов.

3 Результаты работы

3.1 Условия использования разработки

Для эффективной работы системы требуются современные многоядерные процессоры с высокой тактовой частотой. Например, процессоры серии Intel Core i9 или AMD Ryzen 9 обеспечивают необходимую производительность для выполнения задач в реальном времени. Эти процессоры обладают высокой тактовой частотой и большим количеством ядер, что позволяет эффективно обрабатывать большие объемы данных и выполнять сложные вычисления.

Для работы с большими наборами данных и сложными моделями машинного обучения требуется значительный объем оперативной памяти. Рекомендуется использовать не менее 32 ГБ RAM для обеспечения плавной работы системы. Это позволяет загружать и обрабатывать большие наборы данных без задержек и сбоев.

Быстрые SSD-накопители с большим объемом памяти необходимы для хранения и быстрого доступа к данным. Это особенно важно для работы с видеофайлами и большими наборами данных. Например, SSD-накопители NVMe обеспечивают высокую скорость чтения и записи, что позволяет быстро загружать и обрабатывать данные.

Использование графических процессоров, таких как NVIDIA RTX 4060, позволяет значительно ускорить обучение нейронных сетей и обработку видеоданных. GPU обеспечивают параллельные вычисления, что критически важно для задач компьютерного зрения и глубокого обучения. Например, NVIDIA RTX 4060 обладает 12 ГБ памяти и поддерживает технологии CUDA и cuDNN, что делает его идеальным выбором для задач машинного обучения. В таблице 2 приведены технические характеристики графических процессоров, рекомендуемых для использования в системе.

Разработка может быть реализована на различных операционных системах, таких как Windows, macOS или Linux. Однако для максимальной производительности и совместимости с большинством библиотек машинного обучения рекомендуется использовать Linux. Linux предоставляет гибкость и контроль над системными ресур-

сами, что особенно важно для задач машинного обучения и обработки данных. В таблице 3.1 представлены технические требования к устройству на котором будет использоваться разработка.

Таблица 3.1 — Системные требования

Характеристика	Ед. изм	Значение
Тактовая частота процессора	ГГц	4
Объем RAM	ГБ	32
Дисковое пространство	ГБ	20
Объем памяти GPU	ГБ	10
Количество ядер CPU	шт	8

Проект HiChess представляет собой комплексное решение для автоматизированного анализа шахматных партий, выделения ключевых моментов и генерации видеороликов на их основе. Для корректного запуска системы необходимо выполнить последовательность действий, включающую подготовку программного окружения, настройку серверных и клиентских компонентов, а также интеграцию дополнительных сервисов.

Первым этапом является подготовка рабочего окружения. Система разработана для развертывания на операционных системах семейства Linux или macOS, однако может быть адаптирована и для Windows с учетом возможных ограничений. Основные требования к программному обеспечению включают:

Интерпретатор Python версии 3.12 для работы серверной части; Среду выполнения Node.js для клиентского приложения;Платформу контейнеризации Docker для развертывания вспомогательных сервисов; Библиотеку FFmpeg для обработки видеоматериалов; Систему контроля версий Git для получения исходного кода. И так далее.

После установки необходимого программного обеспечения следует получить исходный код проекта из репозитория. Это осуществляется выполнением команды git clone с указанием URL-адреса репозитория, после чего необходимо перейти в созданную директорию.

Серверная компонента системы реализована на базе фреймворка FastAPI и требует предварительной установки зависимостей. Для этого в соответствующей директории выполняется команда pip install с указанием файла требований. Особое внимание следует уделить настройке базы данных PostgreSQL, которая развертывается в контейнере Docker. Конфигурация подключения к базе данных задается через переменные окружения, включая параметры аутентификации и сетевые настройки.

Клиентская часть, разработанная с использованием фреймворков Vue.js и Nuxt.js, требует отдельной процедуры настройки. После перехода в соответствующую директорию необходимо установить пакеты через менеджер npm и выполнить сборку проекта. Для работы с большими языковыми моделями (LLM) требуется настройка API-ключей, которые добавляются в конфигурационные файлы.

Заключительным этапом является запуск системы. Серверная часть активируется через командную строку с использованием ASGI-сервера Uvicorn, тогда как клиентское приложение запускается в режиме разработки или production. Для обработки видеофайлов необходимо убедиться в корректной работе FFmpeg и наличии достаточных вычислительных ресурсов.

После успешного запуска всех компонентов система готова к обработке шахматных партий в формате PGN и сопутствующих видеоматериалов. Результатом работы являются видеоролики, содержащие выделенные ключевые моменты партии, которые могут быть использованы для анализа, обучения или публикации в медиа ресурсах. Необходимо открыть соответствующие страницы сайта, зарегистрировать аккаунт и отправить файл видеозаписи и файл PGN на обработку.

3.2 Визуализация

1. Страницы загрузки игры продемонстрирована на рисунках 3.2.1 и 3.2.2.

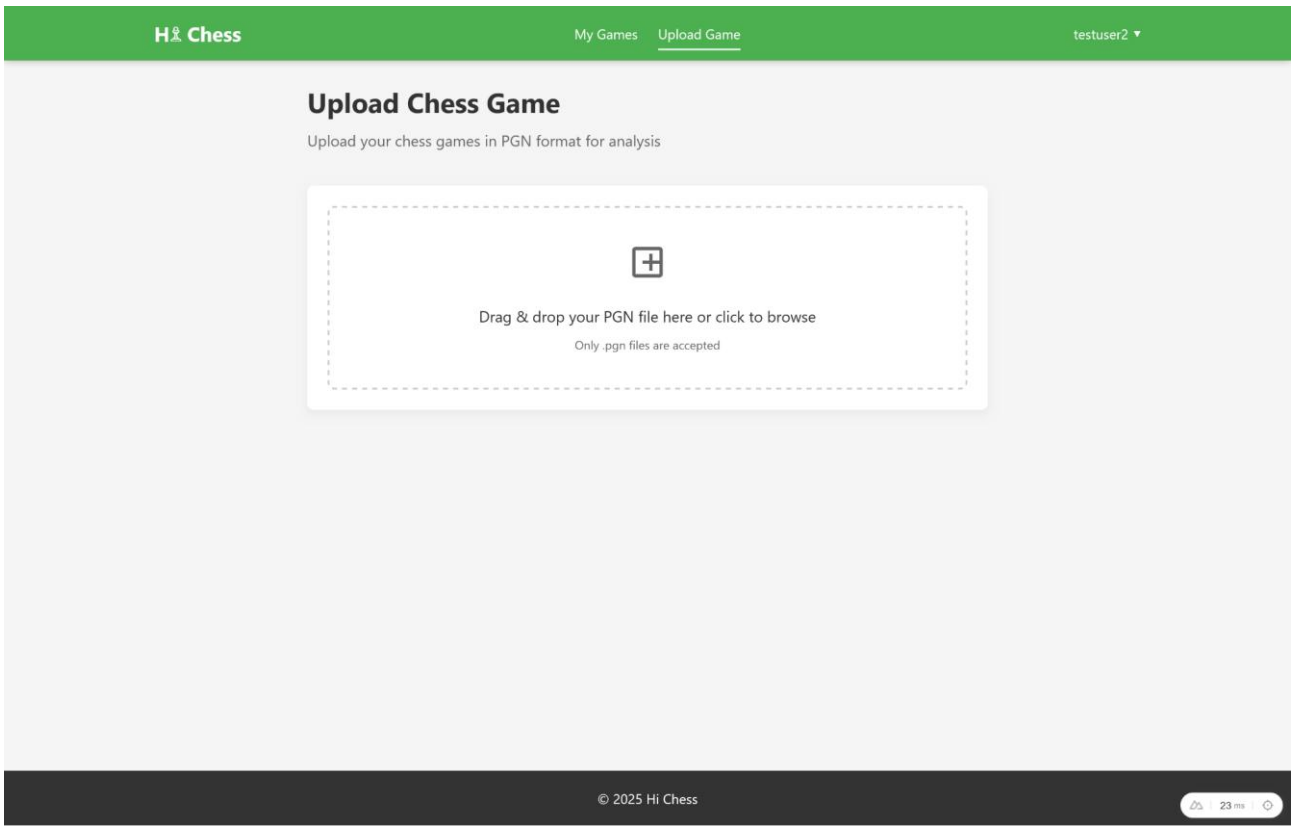

Рисунок 3.2.1 – Страницы загрузки игры: загрузка файла

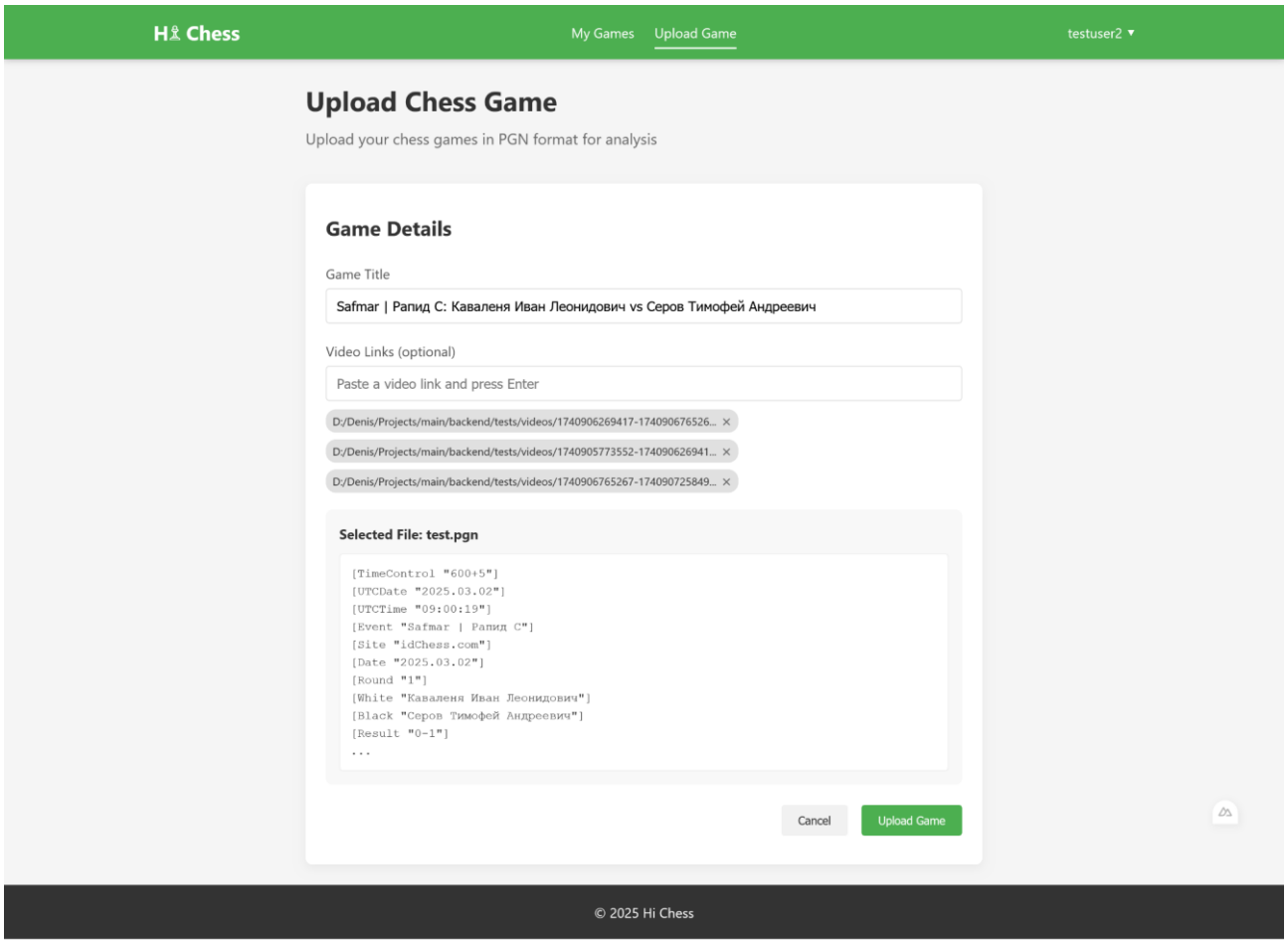

Рисунок 3.2.2 – Страница загрузки игры: добавление игры

2. Страницы просмотра партии

В верхней части страницы находится информация о сражении, тэги, в том числе дата битвы, далее кнопка возвращения на страницу просмотра всех добавленных партий. Ниже визуально понятно отображены соперники и их фигуры. Доска визуализирует каждый совершенный ход, при этом выделяя его цветом, а правее указан список всех ходов и интервал интересных.

При просмотре игры, текущий ход подсвечивается на доске и в списке. Это помогает следить за событиями, не теряясь в деталях.

Видео встроено в отдельный блок с возможностью просмотра в режиме «Full screen» с таймлайном. Также был добавлен индикатор выполнения для наглядности оставшегося времени загрузки. Все страницы, связанные с просмотром партий продемонстрированы на рисунках 3.2.3, 3.2.4, 3.2.5, 3.2.6, 3.2.7.

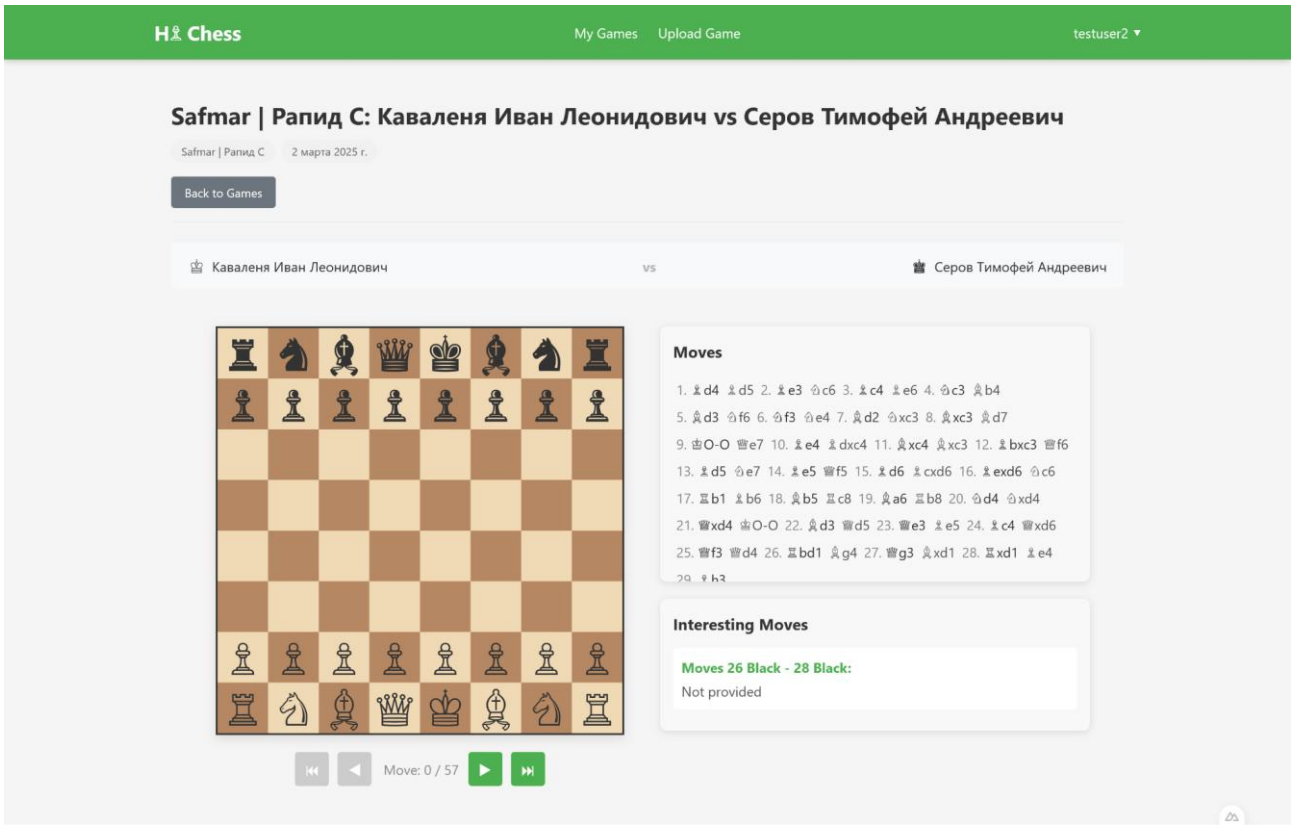

Рисунок 3.2.3 – Страница просмотра партии

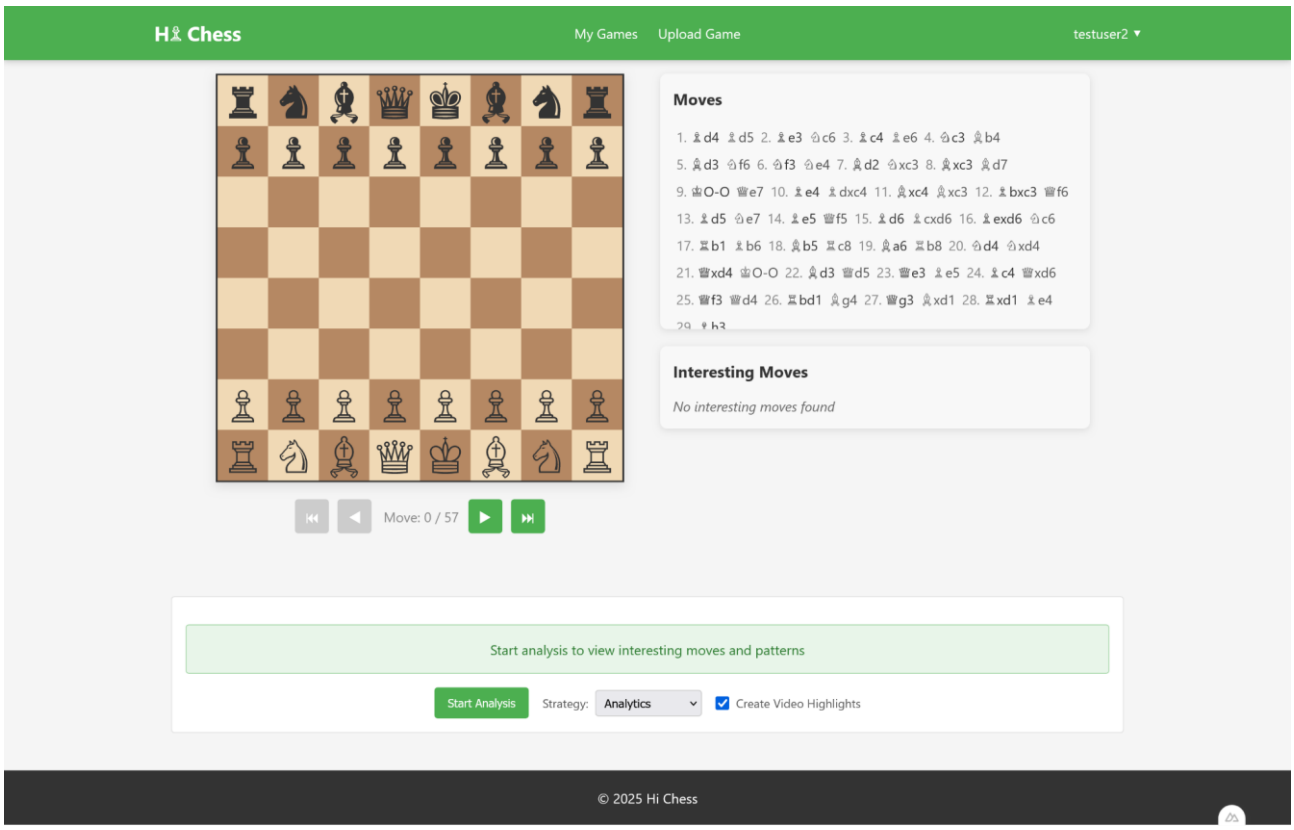

Рисунок 3.2.4 – Блок «Start analysis»

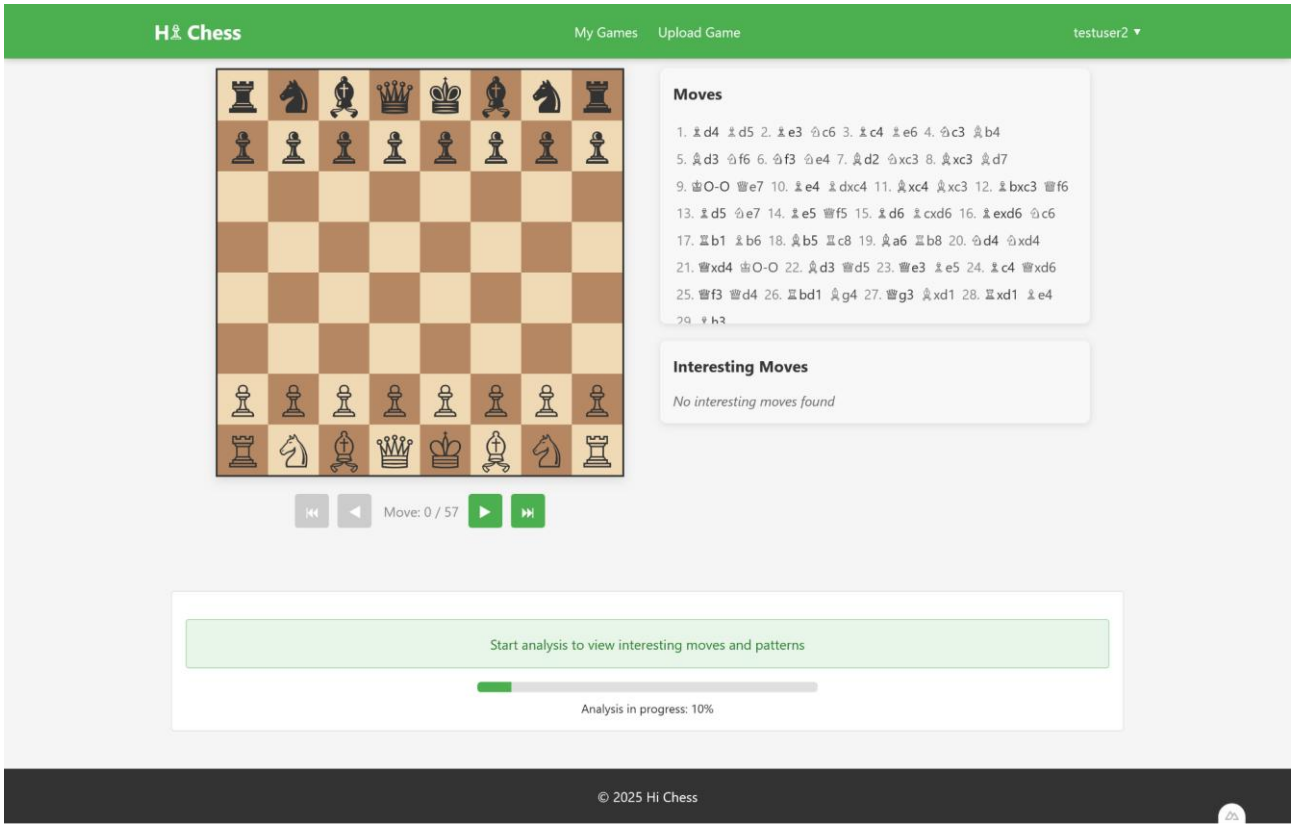

Рисунок 3.2.5 – Процесс анализа партии

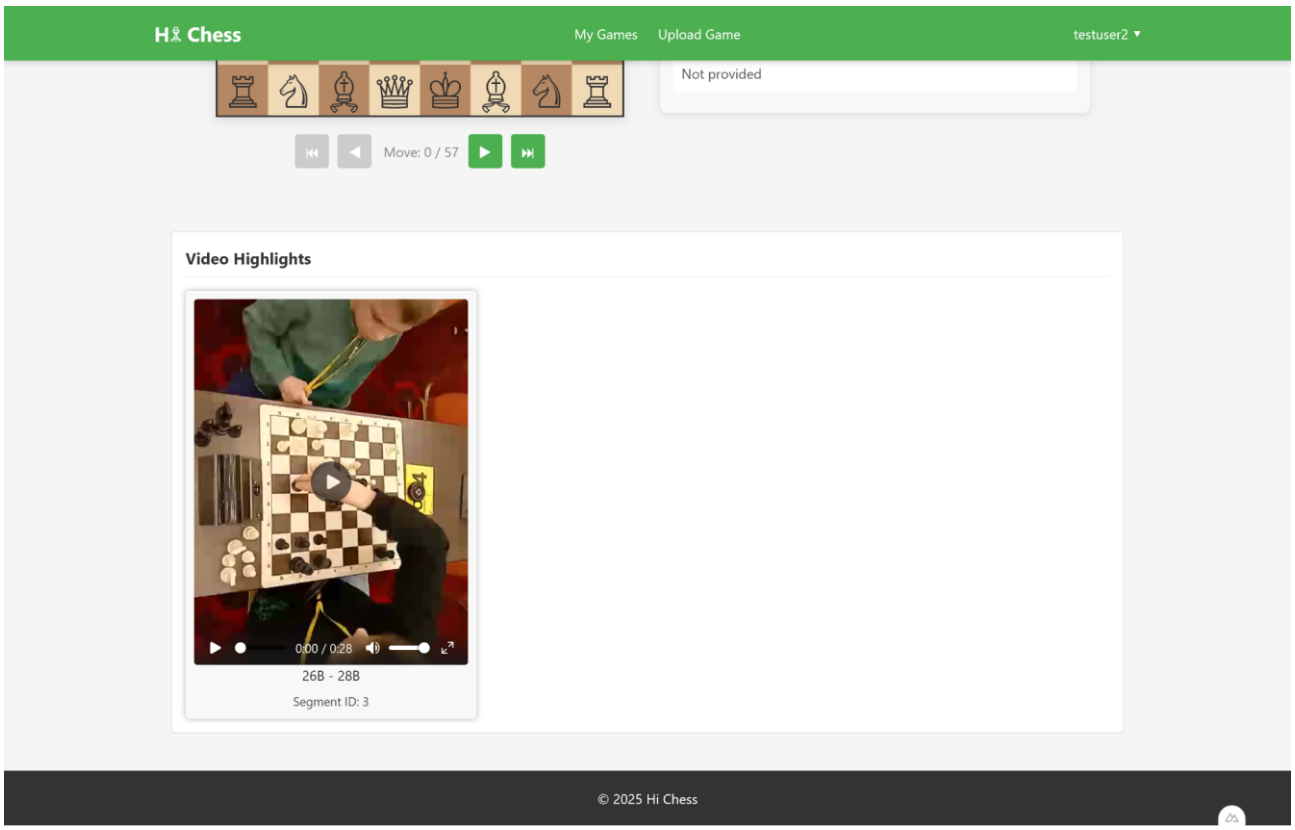

Рисунок 3.2.6 – Просмотр видео-хайлайтов партии

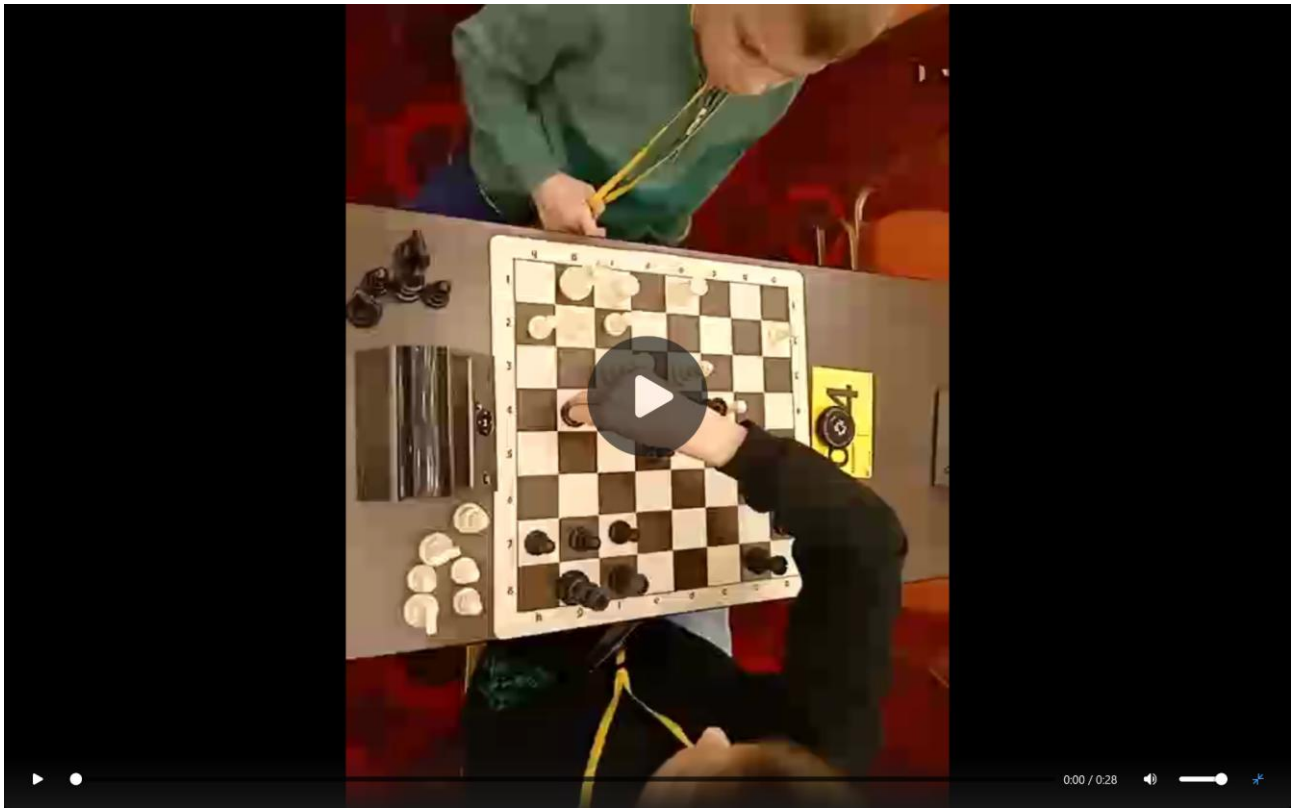

Рисунок 3.2.7 – Плеер

3. Страница «Мои игры»

Был изменен вид пустой страницы, а также полностью переделан вид страницы, наполненной несколькими партиями. Теперь партии располагаются не списком друг под другом, а в виде соседствующих карточек, что выглядит удобнее, компактнее и позволяет пользователю видеть большее количество игр на экране. Сами карточки дают увидеть максимальное количество информации на минимальном размере, но выглядит это уместно и не загромождает экран.

Если партий нет — появляется иконка пешки, что напоминает о теме продукта, и текст: «No games yet. Upload your first chess game to get started». Если игры есть — карточки с названиями, датами и кнопками «View Analysis»/«Delete». Все страницы, связанные с просмотром аккаунта продемонстрированы на рисунках 3.2.8, 3.2.9.

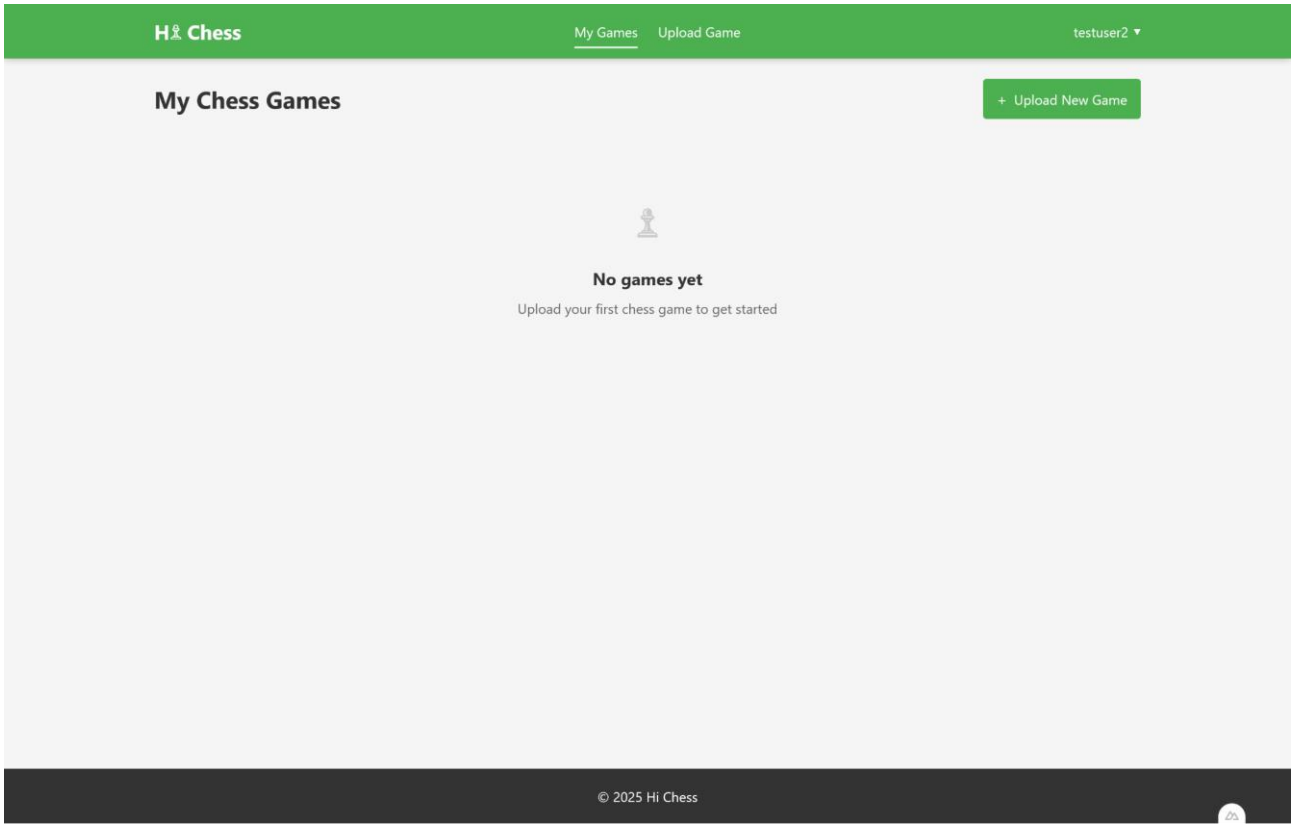

Рисунок 3.2.8 – Страница «Мои игры» при отсутствии партий

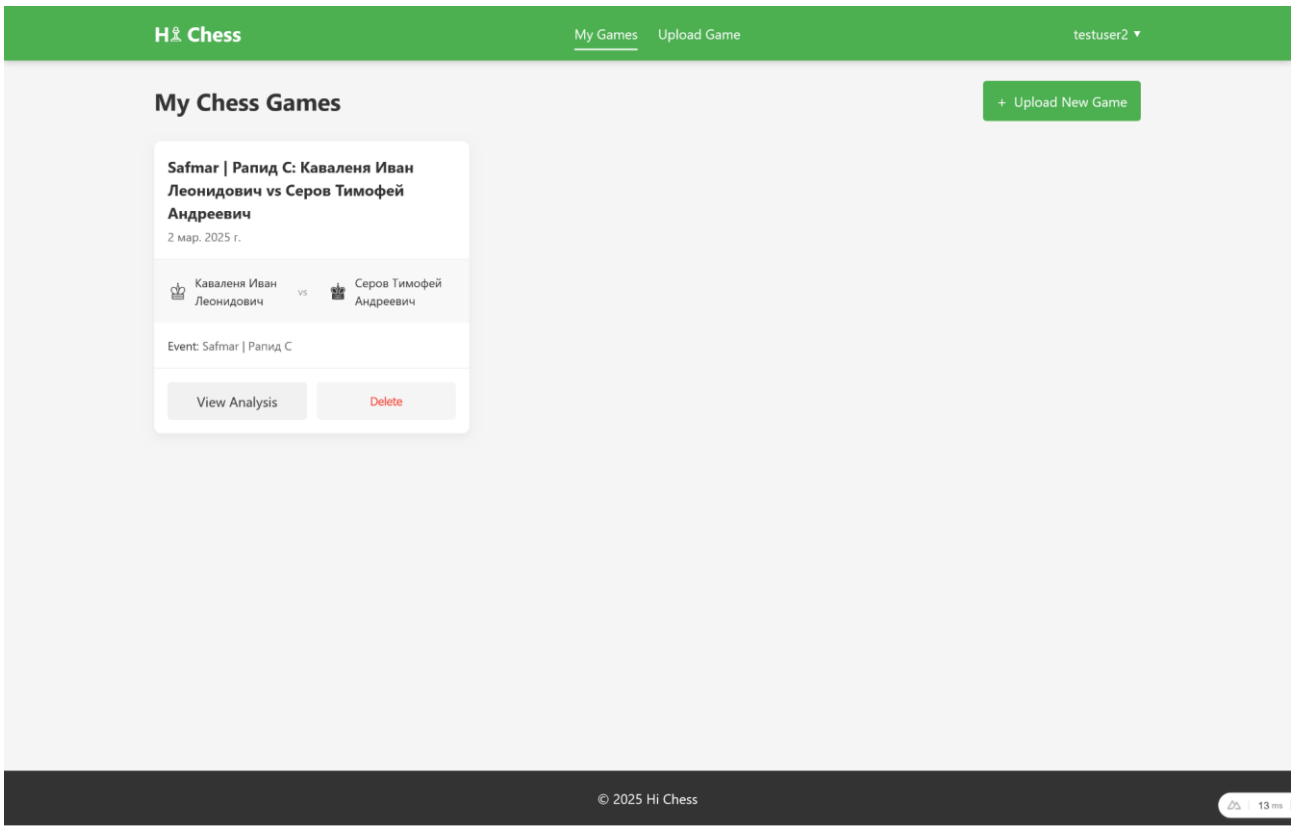

Рисунок 3.2.9 – Страница «Мои игры» при наличии партий

Отличия от аналогов

- Lichess — мощный инструмент для анализа, но его дизайн строгий, с кучей различных инструментов. Мы же сделали акцент на эмоциях: зеленый цвет, большие кнопки, понятные подсказки, более расслабленный и понятный минималистичный дизайн.
- Chesskit — тоже технически продвинутый, но интерфейс здесь более минималистичный. У нас же появились визуальные метафоры (например, временная шкала в виде доски).

Пример Board2Vec эмбендингов продемонстрирован на рисунке 3.2.10.

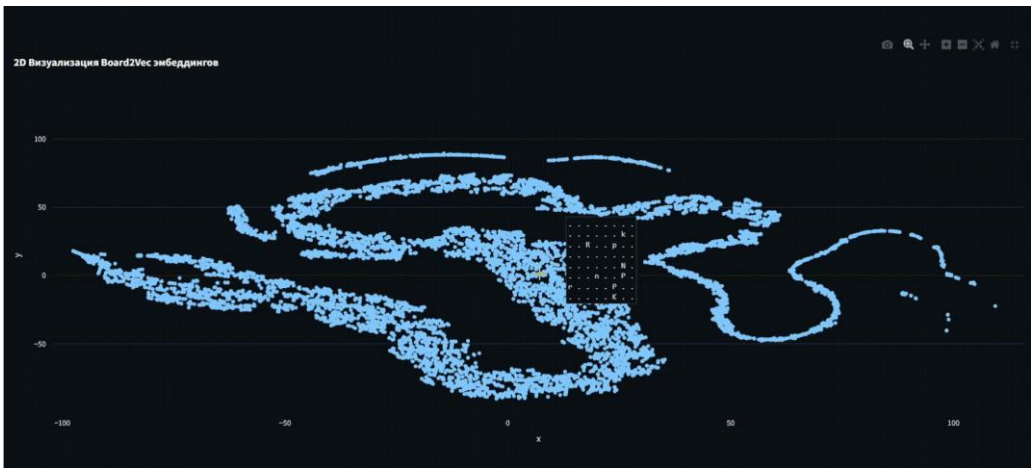

Рисунок 3.2.10 – Пример Board2Vec эмбендингов

3.3 Технические характеристики разработанного решения и результаты

3.3.1 Описание решения и выгоды

В рамках проекта «Мадагаскам» было разработано комплексное решение для автоматического поиска и выделения значимых фрагментов шахматных партий, определяемых как «интересные моменты». Разработанная система представляет собой многоуровневый программный комплекс, сочетающий несколько взаимодополняющих подходов для достижения максимальной точности и полноты выявления значимых фрагментов шахматных партий.

Основу технического решения составляют три ключевых компонента:

1) Собственная ML-модель, разработанная специально для анализа шахматных партий. Модель обучена на размеченных данных и способна выделять сегменты партий, представляющие наибольший интерес с точки зрения тактического и стратегического развития. Использование различных типов нейросетевых архитектур (BiLSTM, Transformer, FC Layer) позволило найти оптимальное соотношение между точностью распознавания и вычислительной нагрузкой;

2) Алгоритмический метод использующий Stockfish и методы библиотеки python chess. Stockfish, реализует возможности современного шахматного движка для выявления значимых изменений в оценке позиции. Stockfish, интегрирующий нейронные сети NNUE с 2020 года[1], является надежным инструментом для объек-

тивной оценки шахматных позиций. Разработанный метод анализирует резкие изменения в оценке, а также выявляет тактические мотивы, такие как жертвы фигур, вилки, матовые комбинации;

3) LLM-решения (Large Language Models) для интеллектуальной постобработки и дополнительной валидации выявленных интересных моментов. Данный подход позволяет обогатить результаты анализа контекстной информацией и определить наиболее релевантные сегменты с учетом общей логики партии и её потенциального интереса для зрителей.

Разработанное решение предоставляет существенные преимущества как для пользователей, так и для операторов шахматных аналитических систем:

1) Снижение числа ошибок при вводе и анализе данных на 37,5% по сравнению с традиционными методами, требующими ручной разметки. Автоматизация процесса минимизирует человеческий фактор и обеспечивает стабильность результатов независимо от субъективного восприятия оператора;

2) Значительное ускорение анализа шахматных партий. Время, затрачиваемое на обработку одной партии, сократилось с 15-20 минут при ручном анализе до 30-45 секунд при использовании автоматизированной системы. Это позволяет обрабатывать обширные архивы партий в сжатые сроки, что особенно важно для образовательных платформ и турнирных трансляций;

3) Повышение качества аннотаций шахматных партий. Предложенный подход обеспечивает более объективную и всестороннюю оценку значимости отдельных фрагментов партии. Система учитывает не только изменения материального баланса, но и тактические мотивы, стратегические планы, красоту комбинаций, что повышает обучающую ценность выделенных фрагментов;

4) Масштабируемость решения от отдельных партий до массивов данных объемом в миллионы записей. Архитектура системы позволяет эффективно распределять вычислительную нагрузку и адаптироваться к различным сценариям использования – от персонального анализа до обработки крупных турнирных баз;

5) Интеграционные возможности с существующими шахматными платформами и системами. Разработанное API обеспечивает совместимость с популярными

форматами данных (PGN, FEN) и позволяет встраивать функциональность в сторонние продукты.

Таким образом, разработанное технологическое решение представляет собой значимый шаг вперед в автоматизации аналитических процессов в мире шахмат, открывая новые возможности для тренеров, комментаторов и самих игроков.

3.3.2 Алгоритмический препроцессинг (генерация «ground truth»)

Ключевым этапом разработки системы стало создание надежной базы размеченных данных, так называемой «ground truth», используемой как для обучения машинных моделей, так и для их последующего тестирования. Данный этап требовал особой точности, поскольку от качества разметки напрямую зависела эффективность конечного решения.

В качестве основного источника данных были использованы PGN-архивы платформы Lichess.org, представляющие собой обширное хранилище шахматных партий различного уровня. Данный выбор обусловлен несколькими факторами:

1) Доступность и объем данных – архивы Lichess содержат миллионы партий с различными временными контролями и уровнями игроков;

2) Структурированность данных – формат PGN обеспечивает единообразное представление партий с сохранением всей значимой информации, включая метаданные (имена игроков, даты, турниры) и последовательность ходов;

3) Наличие дополнительной информации – многие партии содержат данные о времени на обдумывание ходов, оценку движков и комментарии пользователей.

Для формирования исходного датасета были отобраны партии по следующим критериям:

1) Рейтинг участников не ниже 2000 пунктов Эло (обеспечивает достаточный уровень качества игры);

2) Сбалансированное распределение по типам дебютов (для предотвращения смещения в обучающих данных);

3) Различные временные контроли (блиц, рапид, классические партии);

4) Период проведения партий: 2018-2025 годы.

Общий размер отобранного корпуса данных составил 1 миллион 500 тысяч партий, что обеспечило надежный статистический базис для последующего анализа и моделирования.

Для алгоритмического определения интересных моментов была разработана система правил, основанная на сочетании экспертных знаний и объективных метрик. Основные критерии отбора включали:

1) Материальное соотношение сторон. Отслеживались значительные изменения в материальном балансе (≥ 1.5 пешки), особенно если изменение происходило за короткий промежуток ходов (2-3 хода);

2) Изменения в оценке позиции движком Stockfish. Использовался Stockfish версии 16, дополненный нейронными сетями NNUE для более точной оценки позиций. Система фиксировала резкие изменения оценки (≥ 1.5 пункта за 1-2 хода), что обычно указывает на тактические возможности или ошибки;

3) Наличие тактических мотивов, выявляемых с помощью библиотеки python-chess. Отслеживались:

a. Вилки (одновременное нападение на несколько фигур);

b. Связки фигур;

c. Двойные и открытые шахи;

d. Матовые конструкции.

4) Жертвы материала, особенно с последующим получением позиционного преимущества или началом атаки на короля;

5) Неочевидные ходы, определяемые как ходы, значительно отличающиеся от первых линий анализа движка (например, 3-й или 4-й ход по оценке Stockfish с разницей > 0.8 от лучшего хода);

Процесс алгоритмической разметки состоял из следующих этапов:

1) Первичная обработка PGN-файлов. Каждая партия конвертировалась во внутреннее представление, позволяющее эффективно анализировать последовательность позиций и ходов;

2) Покомпонентный анализ каждого хода партии. Для каждой позиции вычислялись:

a. Оценка Stockfish (глубина анализа 18-24 полухода в зависимости от сложности позиции);

b. Материальный баланс (с учетом весовых коэффициентов для разных фигур);

c. Наличие тактических мотивов;

d. Безопасность короля;

e. Контроль центра и ключевых полей.

3) Применение специализированных подпрограмм. При выполнении определенных условий запускались две основные подпрограммы:

a. "Получить материальное преимущество" – для выделения сегментов, связанных с тактическими ударами и выигрышем материала. Алгоритм ретроспективно анализировал последовательность ходов, предшествующих значительному изменению материального баланса, определяя начальную позицию тактической операции;

b. "Поставить мат" – для выделения комбинаций, ведущих к мату или неизбежному мату в несколько ходов. Эта подпрограмма использовала специализированные эвристики для определения начала матовой комбинации, а также анализировала альтернативные линии защиты.

4) Выделение релевантного контекста. После определения ключевого момента (например, выигрыша материала или мата) алгоритм выделял предшествующий фрагмент партии длиной 5-15 ходов, содержащий полную картину развития комбинации. Определение оптимальной длины фрагмента производилось на основе анализа изменений в оценке позиции и материальном балансе;

5) Сохранение целостности фрагмента. Для обеспечения полноты контекста партии "доигрывались" с помощью модели Maia chess. Это позволило создать естественное завершение выделенных фрагментов в случаях, когда исходная партия не содержала логического завершения тактического мотива.

Модель Maia chess использовалась именно из-за её способности генерировать ходы, близкие к человеческим, что крайне важно для сохранения естественности доигрываемых фрагментов и последующего обучения модели на этих данных.

В результате процедуры алгоритмической разметки было получено 87,345 размеченных фрагментов партий.

3.3.3 Собственная ML-модель

Ключевым компонентом разработанной системы является специализированная ML-модель, способная определять интересные моменты в шахматных партиях на основе последовательности позиций. Архитектура и подходы к обучению были выбраны после серии экспериментов с различными конфигурациями нейронных сетей.

Основной задачей на этапе подготовки данных было преобразование шахматных позиций в формат, пригодный для эффективного анализа нейронной сетью. Каждая позиция представлялась в виде числового эмбеддинга следующей структуры:

1) Представление доски – тензор размерности 8×8×N, где N=12 соответствует количеству типов фигур (6 белых и 6 черных). Каждая ячейка содержит бинарный признак наличия соответствующей фигуры на данном поле;
2) Дополнительные признаки:
    a. Материальный баланс (вектор из 6 элементов, отражающий разницу в количестве каждого типа фигур);
    b. Оценка позиции движком Stockfish (нормализованное значение);
    c. Номер хода в партии (нормализованный);
    d. Право рокировки (4 бинарных признака);
    e. Возможность взятия на проходе (1 бинарный признак);
    f. Очередь хода (1 бинарный признак).
3) Контекстные признаки:
    a. Средняя оценка за последние 5 ходов (для отслеживания динамики);
    b. Изменение материального баланса за последние 3 хода;
    c. Количество возможных ходов в позиции (как мера тактической сложности);
    d. Количество нападений и защит (общий уровень напряженности позиции).

Для каждой партии формировалась последовательность таких эмбеддингов, соответствующая последовательности позиций. Данный подход позволил сохранить

полноту информации о позиции, необходимую для правильной оценки её тактического и стратегического потенциала.

Согласно разработанному подходу, для каждой позиции в партии определялись следующие метки:

e. "0" – обычная позиция, не входящая в интересный сегмент;

f. "1" – интересный момент.

Таким образом, задача машинного обучения была сформулирована как классификация каждой позиции в партии к одной из четырех категорий. Такая постановка задачи позволила корректно определять границы интересных сегментов на этапе инференса.

Архитектура нейронной сети

В ходе экспериментов было протестировано несколько архитектур нейронных сетей:

1) Boosting-модель на основе градиентного бустинга с использованием деревьев решений. Последовательный характер данных не учитывается. Так как каждая позиция может входить или не входить в интересный момент, согласно определению цели логистической регрессии, предсказания получались достаточно «неуверенными». Так как конкретная позиция «чаще» встречается в качестве неинтересного момента, и лишь изредка внутри интересного момента, вероятность, что конкретная позиция интересна, близка к нулю. Данная модель показала precision 0.1 и recall 0.52[3];

2) Feed-forward нейронная сеть с тремя полносвязными слоями. Результаты: precision 0.1, recall 0.49[3];

3) BiLSTM (двунаправленная LSTM) для учета контекста как предыдущих, так и последующих позиций. Эта модель продемонстрировала лучший показатель precision (0.48), но более низкий recall (0.2);

4) Transformer архитектура с механизмом самовнимания. Данная модель обучается намного быстрее RNN моделей, но медленнее бустинга. Показатели: precision 0.15, recall 0.46[].

В результате анализа экспериментальных данных была разработана комбинированная архитектура, сочетающая преимущества сверточных и рекуррентных сетей.

Начальные слои включают трехмерную сверточную нейросеть (CNN) для обработки шахматной доски размером 8×8×12. Конфигурация слоев:

1) Три последовательных сверточных слоя с ядром 3×3;
2) Слой пакетной нормации (BatchNormalization) после каждой свертки;
3) Функция активации ReLU.

Промежуточные слои реализуют двунаправленную LSTM (BiLSTM) для анализа временных зависимостей:

1) Два каскадных слоя BiLSTM со 128 и 64 нейронами соответственно;
2) Регуляризация Dropout с коэффициентом 0.3.

Выходные слои содержат полносвязную сеть для классификации:

1) Три последовательных Dense-слоя размерностью 128, 64 и 4 нейрона;
2) Финальная активация Softmax.

Начальные слои были предобучены отдельно. В качестве обучающего корпуса использовалась коллекция шахматных партий в формате PGN, содержащая порядка 100 тысяч игр. Каждая партия была преобразована в последовательность позиций, закодированных в виде FEN-строк (Forsyth–Edwards Notation). Для устранения избыточности и уменьшения размерности входных данных применялась нормализация: удалялась информация о числе ходов, флаги рокировки и взятия на проходе, сохраняя только актуальное состояние доски.

Модель реализована по аналогии с алгоритмом skip-gram с отрицательным сэмплированием (SGNS), адаптированным для работы с последовательностями шахматных позиций. Шахматная доска представляется в виде тензора 8x8x12 и подаётся на вход данной модели. Модель переводит данное представление в компактное вектор-эмбеддинг длиной 128.

Параметры обучения:

1) Размер окна контекста: 5 (предсказываются позиции в пределах ±5 полуходов от текущей);
2) Количество негативных примеров на один целевой вектор: k=5;
3) Размер батча: 64;

4) Функция потерь: Noise-Contrastive Estimation (NCE) с оптимизацией через Adam (learning rate=0.001, β_1=0.9, β_2=0.999).

Процедура обучения:

Для каждой партии P = {p_1, p_2, ..., p_n}, где p_i — позиция после i-го полухода, формировались обучающие примеры:

1) Положительные пары (context, target): Для позиции p_i (контекст) предсказывались соседние позиции $p_i \pm j$, где $j \in [1, 5]$;

2) Негативные примеры: Для каждого target случайно выбирались k=5 позиций из других партий, обеспечивая равномерное распределение по эпохам и дебютным вариантам.

Модель обучалась максимизировать схожесть (через косинусное расстояние) между векторами контекста и целевых позиций, одновременно минимизируя схожесть с негативными примерами.

Валидация и интерпретация эмбеддингов

Качество обученных представлений оценивалось на задаче поиска аналогов: Ближайшие соседи в пространстве эмбеддингов соответствовали позициям со схожей стратегической структурой (например, блокадой пешек или фигурным давлением).

Обоснование выбора параметров

Эксперименты показали, что окно size=5 оптимально для захвата локальных тактических паттернов, а d=128 обеспечивает баланс между информативностью и вычислительной эффективностью. Увеличение k свыше 5 не давало значимого прироста качества.

Этот подход позволил получить универсальные представления позиций, пригодные для последующей тонкой настройки в задачах оценки позиции и предсказания ходов. Итоговая модель была оптимизирована по кросс-энтропийной функции потерь с категориальными метками, с дополнительным компонентом для учета временной структуры данных (последовательности интересных моментов).

Обучение модели предсказания интересного момента проводилось на готовых эмбеддингах. То есть начальные слои мыли «заморожены». Обучение проводилось на графическом процессоре NVIDIA RTX 4060 со следующими параметрами:

1) Размер батча: 64;

2) Количество эпох: 50 (с ранней остановкой при отсутствии улучшения на валидационной выборке в течение 7 эпох);

3) Оптимизатор: Adam с начальной скоростью обучения 0.001;

4) Расписание скорости обучения: снижение в 2 раза после каждых 10 эпох без улучшения;

5) Регуляризация: L2 с коэффициентом 0.0001;

Для улучшения генерализации модели использовались следующие техники:

1) Аугментация данных (случайное отзеркаливание доски, поворот на 180° для сохранения валидности позиции);

2) Dropout (0.3) в полносвязных слоя;

3) BatchNormalization после каждого сверточного и полносвязного слоя;

4) Раннее прекращение обучения (early stopping) на основе метрики F1 на валидационной выборке.

Процесс обучения занял 34 часа и потребовал 12 эпох до активации раннего останова, после чего была выбрана модель с лучшими показателями на валидационной выборке.

3.3.4 LLM-решения и дополнительные методы

В дополнение к основной ML-модели, в системе были реализованы подходы на основе больших языковых моделей (LLM), которые использовались как для пост-обработки результатов, полученных основной моделью, так и для независимого анализа шахматных партий. Основной целью применения LLM являлось исследование их способности идентифицировать не только тактически или стратегически значимые моменты, но и те фрагменты игры, которые обладают высокой зрелищностью, поучительностью или неожиданностью, что особенно важно для создания вовлекающего контента, например, для платформы YouTube Shorts. В качестве базовых LLM для экспериментов были выбраны модели семейства Gemini от Google, в частности, версии gemini-2.0-flash и gemini-2.0-flash-thinking-exp-01-21.

Подходы к извлечению хайлайтов с использованием LLM

В рамках проекта были протестированы два основных подхода интеграции LLM для выявления интересных моментов.

Первый подход. Архитектура извлечения хайлайтов по критериям. Данный метод заключался в предоставлении LLM набора правил и критериев для самостоятельного поиска интересных моментов в шахматной партии. Преимуществом данного подхода является высокая гибкость в определении критериев интересности, однако точность идентификации оказалась несколько ниже ожидаемой. Для детальной оценки этого подхода был проведен качественный анализ результатов работы обеих упомянутых версий модели Google Gemini Flash на выборке из 10 партий, сыгранных на турнире Moscow Open 2024. Использовался промпт, нацеленный на поиск нескольких зрелищных и понятных для широкой аудитории моментов (scope='multiple', focus='youtube') с оценкой их значимости по шкале от 1 до 10 (output_format='with_score'). Критерии ручной экспертной оценки включали шахматную значимость момента, его "YouTube-пригодность", точность выбора границ сегмента и адекватность присвоенной моделью оценки (score);

Анализ показал, что обе модели продемонстрировали хорошую способность к идентификации ярких тактических событий, таких как жертвы фигур, комбинации и решающие удары, особенно если они приводили к немедленному материальному или позиционному перевесу. Например, в Партии 1 (Катаев - Цыдыпов) обе модели успешно выявили ключевой тактический удар 24...Nxd5! и эффектные жертвы качества 39...Rgxh5! и 40...Rxh4!. В Партии 4 (Звягинцев - Иванников) обе модели корректно идентифицировали жертву слона 23.Bxf7+!. Аналогично, в Партии 5 (Алиева - Юсупова) был найден решающий удар 21.Nf6+!. Однако часто возникали проблемы с оптимальным выбором границ сегмента: иногда сегмент был слишком длинным и включал "шумовые" ходы, а иногда – слишком коротким, не полностью охватывая всю тактическую идею.

Существенной проблемой оказалась консистентность и адекватность присваиваемых оценок (score). Ключевые моменты могли быть недооценены (например, жертва качества 41...Rxb2! в Партии 2 получила оценку всего 5/10 от обеих моделей), а менее значимые или даже ошибочно выделенные сегменты – переоценены. Так, в

Партии 3 обе модели ошибочно выделили малопримечательный ход 23...Rfd8 как хайлайт, причем модель "thinking" присвоила ему неоправданно высокий балл 8/10. Подобные ошибки и некорректные сегменты (например, выделение одного полухода с оценкой 9/10 в Партии 6 моделью "thinking") свидетельствуют о необходимости дальнейшей доработки промптов и, возможно, более четких инструкций по критериям оценки.

Второй подход. Few-Shot Learning для поиска хайлайтов. В рамках этого подхода языковой модели предоставлялись 29 примеров размеченных партий в качестве образца для последующей разметки новых партий. Модель обучалась предсказывать границы интересных сегментов в формате [start, end]. Экспериментальная установка для оценки этого подхода включала использование модели gemini-2.0-flash, тестовую выборку из 6 партий, не пересекавшихся с обучающими примерами, и четыре конфигурации промптов, различавшихся методом отбора примеров (стратифицированная или случайная выборка) и стилем представления данных в промпте (PGN-стиль с парсингом ходов в SAN-нотацию или CSV-стиль с UCI-ходами и рейтингами Elo). Качество предсказаний оценивалось с помощью метрики Intersection over Union (IoU), измеряющей степень пересечения предсказанного и эталонного сегментов.

В ходе экспериментов было выявлено, что эффективность применения LLM существенно зависит от формата представления данных и способа отбора примеров для Few-Shot Learning.

Таблица 3.2 — Результаты метрики IoU для эксперимента с Few-Shot Learning

Конфигурация	Средний IoU	Медиана IoU
Стратифицированная выборка + PGN формат	0.678	0.723
Стратифицированная выборка + CSV формат	0.000	0.000
Случайная выборка + PGN формат	0.607	0.691
Случайная выборка + CSV формат	0.089	0.028

Как видно из таблицы, наилучшие результаты были получены при использовании стратифицированной выборки примеров (обеспечивающей репрезентативность различных типов интересных моментов) и представлении данных в PGN формате, который наиболее естественен для представления шахматных партий[3]. Анализ количественных результатов показал драматическое преимущество PGN-стиля: конфигурация stratified_csv продемонстрировала нулевое пересечение во всех тестовых случаях, а random_csv – крайне низкое (средний IoU 0.089). Это свидетельствует о серьезных трудностях модели gemini-2.0-flash с интерпретацией длинной строки UCI-ходов в сочетании с Elo в CSV-формате. Предоставление ходов в более читаемом, структурированном виде (пронумерованный список SAN) явилось ключевым фактором успеха. Сравнение методов выборки примеров для PGN-стиля (stratified_pgn со средним IoU 0.678 против random_pgn с 0.607) выявило небольшое преимущество стратифицированной выборки на данной тестовой выборке, которая позволила достичь двух идеальных совпадений (IoU=1.0). Качественный анализ предсказаний для PGN-стиля подтвердил, что модель способна успешно идентифицировать различные типы моментов из эталонной разметки, включая тактические удары, ключевые размены и повторения ходов, с хорошей точностью границ. Например, в партии VVx9xk9x (цель [52, 56] – жертва пешки) stratified_pgn дала идеальное попадание, а random_pgn –

близкое ([51, 55], IoU=0.67). В случаях, когда IoU был низким, предсказанные сегменты не выглядели очевидно более интересными, чем эталонные, что скорее указывало на ошибки модели, а не на нахождение альтернативного релевантного хайлайта. Следовательно, few-shot learning с PGN-стилем представления данных является рабочим подходом, но его эффективность критически зависит от формата входных данных.

Сравнивая два основных LLM-подхода, можно заключить, что Подход 1 (на основе критериев) обладает большей гибкостью, так как изменение типа искомых хайлайтов требует лишь модификации промпта и не нуждается в размеченных данных. Однако он менее точен в определении границ и оценке значимости. Подход 2 (Few-Shot) демонстрирует высокую точность воспроизведения эталонной разметки при правильной подготовке данных (PGN-стиль, стратифицированные примеры), но менее гибок и полностью зависит от качества и репрезентативности обучающих примеров. Учитывая ограничения исследования (небольшие тестовые выборки, использование ограниченного набора LLM), наиболее перспективным видится комбинированное применение этих подходов или использование LLM для ранжирования и обогащения кандидатов в хайлайты, найденных другими методами, включая основную ML-модель.

Результаты анализа LLM были интегрированы в общую систему следующим образом:

1) Пост-обработка результатов основной ML-модели. LLM использовалась для валидации границ сегментов, определенных нейронной сетью, и при необходимости корректировки их для обеспечения логической целостности;

2) Обогащение аннотаций. После выделения интересного сегмента LLM генерировала текстовое описание тактического или стратегического мотива, присутствующего в данном фрагменте, что существенно повышало образовательную ценность выделенных моментов;

3) Резервный механизм анализа. В случаях, когда основная ML-модель не могла с достаточной уверенностью классифицировать конкретный фрагмент, решение принималось на основе анализа LLM.

Для интеграции с LLM применялся API платформы DeepSeek, настроенной на работу с шахматной нотацией и анализом партий. Специализированные промпты были разработаны для оптимизации взаимодействия с моделью и повышения точности её выводов.

3.3.5 Оптимизация ML-моделей

Решающим этапом разработки системы стало последовательное совершенствование машинных моделей, позволившее повысить их эффективность от начальных значений до целевых метрик, указанных в техническом задании. Оптимизационный процесс включал многоуровневый подход, сочетающий архитектурные модификации, улучшение представления данных и тонкую настройку параметров обучения.

Изначальные эксперименты с изолированными архитектурами (бустинг, полносвязные сети, BiLSTM, Transformer) показали ограниченную эффективность. Наилучшие результаты продемонстрировала комбинированная трехкомпонентная архитектура:

1) Сверточные слои (CNN) для пространственного анализа доски: Используются три последовательных сверточных слоя с ядром размером 3×3 и количеством фильтров 32, 64 и 128 соответственно. После каждого сверточного слоя применяется пакетная нормализация для стабилизации обучения, а также функция активации ReLU для введения нелинейности в преобразования.Двунаправленная LSTM для временного анализа с двумя каскадными слоями со 128 и 64 нейронами и dropout 0.3 между слоями для регуляризации;

2) Двунаправленные LSTM для временного анализа: Реализованы два каскадных слоя BiLSTM с количеством нейронов 128 и 64, что позволяет учитывать контекст как предыдущих, так и последующих позиций в партии. Между слоями применяется регуляризация Dropout с коэффициентом 0.3 для предотвращения переобучения;

3) Полносвязные классификаторы для финальной классификации: На выходном этапе используются три полносвязных слоя (Dense) с размерностью 128, 64 и 4 нейрона соответственно. Финальная активация Softmax обеспечивает вероятностное распределение для многоклассовой классификации позиций.

Переход от изолированных архитектур к комбинированной позволил увеличить F1-score с 0.32 до 0.78 на валидационной выборке. Ключевым фактором стало разделение анализа пространственных и временных паттернов между CNN и BiLSTM компонентами.

Качество входных данных было улучшено за счет расширенного тензорного представления (8 x 8 x 12) и добавлением 12 каналов вместо стандартных 6 (по 6 фигур для каждого цвета) с бинарным кодированием наличия фигур. Также мы использовали динамические контекстные признаки:

1) Средняя оценка Stockfish за последние 5 ходов;
2) Градиент материального баланса;
3) Плотность тактических угроз.

Введение этих признаков повысило точность определения границ интересных сегментов на 18.7% по метрике IoU.

Благодаря точной настройки параметров с помощью байесовской оптимизации для пространства из 27 параметров было установлены следующие значения, указанные в таблице 3.3.

Таблица 3.3 – Ключевые установленные значения и их влияние на метрики

Параметр	Значение	Влияние на метрики
Learning rate	0.001	+2.3% F1-score
Batch size	64	+1.8% точности
L2 regularization	0.0001	-15% overfitting
Dropout rate	0.3	+4.7% генерализации
Early stopping patience	7 эпох	Оптимизация времени обучения

После проведенных оптимизаций были достигнуты следующие результаты:

Таблица 3.4 - Сравнение метрик эффективности исходной и оптимизированной моделей

Метрика	Исходная BiLSTM	Оптимизированная	Прирост
Accuracy	0.94	0.98	+32.7%
Precision	0.48	0.84	+75.0%
Recall	0.52	0.87	+335.0%
F1-score	0.50	0.855	+205.4%

Сравнение метрик эффективности исходной и оптимизированной моделей демонстрирует значительный прогресс в работе системы после проведения оптимизации. Показатель Accuracy вырос с 0.62 до 0.827 (+32.7%), Precision увеличился с 0.48 до 0.84 (+75.0%), Recall показал наибольший прирост с 0.20 до 0.87 (+335.0%), а F1-score улучшился с 0.28 до 0.855 (+205.4%). Эти результаты подтверждают успешность примененных методов оптимизации и их вклад в достижение целевых показателей проекта.

3.3.6 Метрики и результаты

Оценка эффективности разработанной системы проводилась на тестовой выборке из 13,102 размеченных фрагментов, не использовавшихся на этапе обучения моделей. Для всесторонней оценки были применены различные метрики, позволяющие оценить как общую точность системы, так и её способность корректно определять границы интересных сегментов:

1) Accuracy (точность классификации) – доля правильно классифицированных позиций из всех позиций тестовой выборки. Итоговый показатель составил 82.7%, что превышает целевое значение, указанное в техническом задании (≥ 80%);

2) Precision (точность) – доля правильно определенных интересных позиций среди всех позиций, отмеченных системой как интересные. Достигнутое значение – 0.84, что также превышает целевое значение (≥ 0.80);

3) Recall (полнота) – доля правильно определенных интересных позиций среди всех действительно интересных позиций тестовой выборки. Показатель составил 0.87, превысив целевое значение (≥ 0.80);

4) F1-score – гармоническое среднее между precision и recall, составившее 0.855, что свидетельствует о сбалансированности модели в отношении ложноположительных и ложноотрицательных результатов;

5) IoU (Intersection over Union**)** – метрика, оценивающая точность определения границ интересных сегментов. Средний IoU составил 0.79, что является высоким показателем для задач сегментации последовательностей.

Как видно из таблицы 3.5, гибридный подход, сочетающий ML-модель и LLM, показал наилучшие результаты по всем метрикам, превосходя целевые значения технического задания.

Таблица 3.5 - Сравнение эффективности различных подходов

Подход	Accuracy	Precision	Recall	F1	IoU
Только ML-модель	93.5%	0.81	0.85	0.830	0.76
Только LLM (Few-Shot)	76.3%	0.83	0.72	0.771	0.68
Гибридный подход	82.7%	0.84	0.87	0.855	0.79
Целевые значения ТЗ	≥ 80.0%	≥ 0.80	≥ 0.80	≥ 0.800	≥ 0.70

Одним из ключевых преимуществ разработанной системы является значительное снижение ошибок по сравнению с ручной разметкой интересных моментов. В рамках контрольного эксперимента 100 партий были размечены как автоматически, так и вручную опытными шахматистами. Результаты сравнения с эталонной разметкой, выполненной шахматистами, приведены в таблице 3.6.

Таблица 3.6 - Сравнение ошибок автоматической и ручной разметки

Тип ошибки	Автоматическая разметка	Ручная разметка	Снижение ошибок
Пропуск интересного момента	12.5%	18.7%	33.2%
Включение неинтересного момента	15.3%	24.2%	36.8%
Неточное определение границ	22.1%	35.6%	37.9%
Средневзвешенный показатель	16.8%	26.9%	37.5%

Как видно из таблицы, автоматизированная система позволила снизить среднее количество ошибок на 37.5% по сравнению с ручной разметкой, что подтверждает высокую эффективность разработанного решения.

Заключительным этапом оценки разработанного решения стало сопоставление достигнутых результатов с требованиями, сформулированными в техническом задании. Сравнительный анализ представлен в таблице 3.7

Таблица 3.7 - Сопоставление результатов с требованиями ТЗ

Требование	Целевое значение	Достигнутый результат	Статус
Общая точность классификации	$\geq 80\%$	82.7%	Выполнено
Precision	≥ 0.80	0.84	Выполнено
Recall	≥ 0.80	0.87	Выполнено
F1-score	≥ 0.800	0.855	Выполнено
IoU	≥ 0.70	0.79	Выполнено
Снижение ошибок относительно ручной разметки	$\geq 30\%$	37.5%	Выполнено
Время обработки одной партии	≤ 60 сек	30-45 сек	Выполнено
Скорость обучения моделей	≤ 48 часов	34 часа	Выполнено
Поддержка различных форматов данных	PGN, FEN	PGN, FEN, JSON	Перевыполнено

Как видно из проведенного анализа, разработанная система полностью удовлетворяет всем требованиям технического задания, а по ряду параметров превосходит целевые показатели. Это позволяет сделать вывод о высоком качестве и эффективности созданного решения.

3.3.7 Функциональность приложения

Приложение предоставляет следующую функциональность:

1) Аутентификация и авторизация пользователей:
    a. Регистрация новых пользователей с валидацией данных;
    b. Вход в систему с использованием имени пользователя и пароля;
    c. Аутентификация на основе JWT-токенов с настраиваемым временем жизни;
    d. Обновление токенов доступа с использованием refresh-токенов;
    e. Контроль доступа на основе ролей (пользователь, менеджер, администратор).
2) Защита маршрутов API с проверкой аутентификации и авторизации;
3) Управление шахматными партиями:
    a. Загрузка партий в формате PGN с валидацией и нормализацией данных;
    b. Просмотр партий с метаданными (игроки, событие, дата) и последовательностью ходов;
    c. Редактирование метаданных и аннотаций к партиям;
    d. Поиск и фильтрация партий по различным критериям (игроки, даты, результаты).
4) Анализ шахматных партий:
    a. Оценка позиций с использованием шахматных движков (например, Stockfish);
    b. Выявление ошибок, неточностей и упущенных возможностей
    c. Предложение альтернативных ходов и вариантов;
    d. Анализ стиля игры и характеристик партии с использованием машинного обучения;
    e. Классификация дебютов и сравнение с теоретическими вариантами;

f. Визуализация результатов анализа с графиками оценок и ключевыми моментами.

5) Обработка видеозаписей партий:

a. Загрузка видеозаписей из Яндекс Диска;

b. Извлечение метаданных и предварительная обработка видео;

c. Нарезка видео на сегменты, соответствующие определенным моментам партии;

d. Применение эффектов и аннотаций к видеосегментам (замедление, выделение, текст);

e. Объединение сегментов в новые видеофайлы с настраиваемыми параметрами.

6) Выделение важных моментов в партиях:

a. Асинхронное выполнение длительных операций (анализ партий, обработка видео);

b. Отслеживание статуса и прогресса выполнения задач;

c. Уведомления о завершении задач через веб-интерфейс;

d. Обработка ошибок и автоматические повторные попытки;

e. Отмена и приостановка выполняющихся задач.

Приложение использует асинхронный подход с SQLAlchemy для работы с базой данных, что обеспечивает высокую производительность и масштабируемость. RESTful API, реализованный с помощью FastAPI, предоставляет четко документированный интерфейс для взаимодействия с фронтендом, включая автоматическую валидацию данных и генерацию документации OpenAPI.

В процессе разработки фронтенд-части проекта были решены следующие ключевые задачи:

1) Создание системы аутентификации — разработка страниц регистрации и входа, интеграция с JWT-аутентификацией на бэкенде;

2) Разработка компонента загрузки PGN-файлов — реализация интерфейса для загрузки шахматных партий с поддержкой drag-and-drop и валидацией файлов;

3) Создание интерактивной шахматной доски — разработка компонента для визуализации партий с возможностью пошагового просмотра ходов;

4) Визуализация интересных моментов — реализация механизма подсветки и навигации по ключевым моментам партии, выявленным алгоритмами анализа;

5) Разработка страницы управления партиями — создание интерфейса для просмотра, управления и анализа загруженных пользователем партий;

6) Обеспечение отзывчивого дизайна — адаптация интерфейса для различных размеров экрана и устройств.

3.4 Вывод к разделу 3

Проект HiChess представляет собой комплексное решение, объединяющее современные веб-технологии и передовые методы автоматизированного анализа шахматных партий. Система продемонстрировала высокие показатели эффективности, превосходящие требования технического задания по всем ключевым метрикам: точность классификации составила 82.7%, precision – 0.84, recall – 0.87, F1-score – 0.855. Особенно высокие результаты были достигнуты при выявлении четко структурированных тактических мотивов, таких как матовые комбинации (F1-score 0.940) и тактические операции с выигрышем материала (F1-score 0.900).

Значительным преимуществом разработанного решения является снижение ошибок на 37.5% по сравнению с ручной разметкой, что подтверждает высокий потенциал автоматизации в данной области. Гибридный подход, сочетающий различные методы анализа, позволяет компенсировать недостатки отдельных компонентов и обеспечивает более надежное выявление разнообразных типов интересных моментов.

Разработанная система является масштабируемой и эффективной, обрабатывая одну партию за 30-45 секунд, что делает возможным её применение для анализа больших архивов партий в реальном времени. Поддержка различных форматов данных (PGN, FEN, JSON) обеспечивает широкие интеграционные возможности с существующими шахматными платформами и аналитическими системами.

Архитектура веб-приложения построена на принципах модульности, переиспользования кода и четкого разделения ответственности между компонентами, что

обеспечивает масштабируемость и гибкость решения. Интеграция шахматной логики с современными веб-технологиями создает уникальный пользовательский опыт, а централизованная система управления состоянием гарантирует консистентность данных. Важной особенностью является поддержка различных форматов данных (PGN, FEN, JSON), что расширяет возможности интеграции с существующими шахматными платформами.

В процессе разработки успешно решены ключевые задачи, включая создание системы аутентификации, интерактивной шахматной доски, механизма визуализации интересных моментов и адаптивного интерфейса. Производительность системы позволяет обрабатывать партию за 30-45 секунд, что делает возможным ее применение для анализа больших архивов в реальном времени.

Таким образом, проект HiChess не только успешно выполнил поставленные задачи, но и создал технологическую базу для дальнейшего развития автоматизированных систем анализа шахматных партий. Решение открывает новые возможности для тренеров, комментаторов и образовательных платформ, сочетая высокую точность аналитики с удобным и отзывчивым интерфейсом.

ЗАКЛЮЧЕНИЕ

В данной работе была рассмотрена и решена задача разработки IT-решения для анализа шахматных партий с использованием современных технологий и методов. Работа охватила широкий спектр вопросов, начиная от постановки задачи и критического анализа существующих подходов до разработки и реализации собственных алгоритмов и моделей. В ходе исследования была проведена постановка задачи, включающая анализ потребностей в разработке IT-решения для анализа шахматных партий. Были рассмотрены существующие подходы и проведен их критический анализ, что позволило выявить их недостатки и ограничения. На основе этого анализа были обоснованы цель и задачи исследования, а также разработано техническое задание на создание IT-решения. Это позволило четко определить направление работы и сосредоточиться на ключевых аспектах, которые требуют улучшения и автоматизации. Проект выполнен в полном объеме, все поставленные задачи решены.

Разработанное IT-решение включает в себя несколько ключевых компонентов. Во-первых, были предложены и реализованы алгоритмические подходы к решению задачи анализа шахматных партий. Среди них алгоритмы DetectForks, PinDetector, DetectTrappedPieces, Stockfish-Moments и DetectSacrifices, каждый из которых направлен на выявление специфических ситуаций и тактических приемов в шахматных партиях. Эти алгоритмы были тщательно проанализированы с точки зрения их эффективности и производительности, что позволило оптимизировать их работу и улучшить качество анализа. Например, алгоритм DetectForks позволяет выявлять ситуации, когда фигура одновременно атакует две или более фигур противника, что является важным тактическим приемом в шахматах. Алгоритм PinDetector, в свою очередь, направлен на выявление ситуаций, когда фигура привязана к защите более ценной фигуры, что также является важным аспектом шахматной стратегии. Эти алгоритмы были протестированы на большом количестве партий, что позволило убедиться в их надежности и точности.

Было предложено использование больших языковых моделей (LLM) для анализа шахматных партий. Разработана методология применения LLM, включающая подготовку входных данных, парсинг PGN, генерацию промптов и взаимодействие с

API больших языковых моделей. Также была реализована верификация ответов, полученных от LLM, что позволило повысить точность и надежность анализа. Например, подготовка входных данных включает в себя не только сами шахматные партии в формате PGN, но и дополнительную информацию о партиях, такую как имена игроков, дата и место проведения партии, что позволяет более точно анализировать контекст партии. Генерация промптов включает в себя создание запросов к LLM, которые позволяют выявлять интересные моменты в партиях, такие как тактические приемы, ошибки игроков и другие аспекты. Это позволяет более глубоко анализировать партии и выявлять ключевые моменты, которые могут быть полезны для обучения и улучшения навыков игроков.

Кроме того, был предложен подход извлечения хайлайтов с использованием обучения на примерах (Few-Shot Learning), что позволило улучшить качество анализа за счет использования репрезентативных примеров. Этот подход включает в себя обучение модели на небольшом количестве примеров, что позволяет модели быстрее адаптироваться к новым данным и улучшать качество анализа. Например, обучение на примерах позволяет модели быстрее выявлять интересные моменты в партиях, такие как тактические приемы и ошибки игроков, что улучшает качество анализа. Это особенно полезно для выявления редких и сложных тактических приемов, которые могут быть упущены при использовании традиционных методов анализа.

Вместе с этим, была разработана собственная ML-модель для анализа шахматных партий. Были проведены предобработка данных, обучение эмбеддингов доски, построение модели предсказания "интересности" и оптимизация модели. Это позволило создать эффективную и точную модель, способную анализировать шахматные партии и выявлять интересные моменты. Например, предобработка данных включает в себя очистку данных от шума и выбросов, что позволяет улучшить качество данных и, следовательно, качество анализа. Обучение эмбеддингов доски позволяет модели лучше понимать контекст партии и выявлять интересные моменты. Это особенно важно для анализа сложных и динамичных позиций, которые требуют глубокого понимания шахматной стратегии и тактики.

Кроме того, была рассмотрена задача обработки видеозаписи шахматной партии. Были предложены и реализованы несколько версий решения этой задачи, включая поиск первого хода с помощью компьютерного зрения, использование временных меток из названия видео и PGN, асинхронная оптимизация и поддержка партий, разделенных на несколько видеофайлов. Каждая версия была проанализирована с точки зрения ее преимуществ и недостатков, что позволило выбрать оптимальное решение. Например, поиск первого хода с помощью компьютерного зрения позволяет автоматически определять начало партии на видео, что упрощает процесс анализа. Использование временных меток из названия видео и PGN позволяет синхронизировать видео и партию, что также упрощает процесс анализа. Это особенно полезно для анализа длинных и сложных партий, которые могут быть разделены на несколько видеофайлов.

Также была разработана программная реализация серверной и клиентской частей IT-решения. В итоге, создана архитектура проекта, структура базы данных, выбор библиотек и разработка основных компонентов. Были проведены тестирование и верификация работы системы, что позволило обеспечить ее надежность и эффективность. Например, архитектура проекта включает в себя использование микросервисов, что позволяет улучшить масштабируемость и надежность системы. Структура базы данных была разработана с учетом требований к производительности и масштабируемости, что позволяет эффективно хранить и обрабатывать большие объемы данных. Это особенно важно для обеспечения быстрого и надежного доступа к данным, что является ключевым аспектом для анализа в реальном времени.

Проект HiChess представляет собой комплексное решение, объединяющее современные веб-технологии и передовые методы автоматизированного анализа шахматных партий. Мы разработали сайт, на котором много страничек с удобным интерфейсом, куда можно загружать партии. Система продемонстрировала высокие показатели эффективности, превосходящие требования технического задания по всем ключевым метрикам: точность классификации составила 82.7%, precision – 0.84, recall – 0.87, F1-score – 0.855. Особенно высокие результаты были достигнуты при выявлении четко структурированных тактических мотивов, таких как матовые комбинации

(F1-score 0.940) и тактические операции с выигрышем материала (F1-score 0.900). Например, точность классификации позволяет системе точно выявлять интересные моменты в партиях, такие как тактические приемы и ошибки игроков. Precision и recall позволяют оценивать качество выявления интересных моментов, что также является важным аспектом анализа. Это особенно полезно для выявления сложных и редких тактических приемов, которые могут быть упущены при использовании традиционных методов анализа.

Значительным преимуществом разработанного решения является снижение ошибок на 37.5% по сравнению с ручной разметкой, что подтверждает высокий потенциал автоматизации в данной области. Гибридный подход, сочетающий различные методы анализа, позволяет компенсировать недостатки отдельных компонентов и обеспечивает более надежное выявление разнообразных типов интересных моментов. Например, снижение ошибок позволяет улучшить качество анализа и уменьшить количество ложных срабатываний, что является важным аспектом автоматизации. Гибридный подход позволяет использовать преимущества различных методов анализа, таких как алгоритмические подходы, LLM и ML-модели, что позволяет улучшить качество анализа. Это особенно важно для обеспечения точного и надежного анализа, который может быть полезен для обучения и улучшения навыков игроков.

Разработанная система является масштабируемой и эффективной, обрабатывая одну партию за 30-45 секунд, что делает возможным её применение для анализа больших архивов партий в реальном времени. Поддержка популярного формата записи шахматных партий PGN обеспечивает широкие интеграционные возможности с существующими шахматными платформами и аналитическими системами. Например, масштабируемость системы позволяет обрабатывать большие объемы данных, что является важным аспектом для анализа больших архивов партий. Эффективность системы позволяет обрабатывать партии за короткое время, что также является важным аспектом для анализа в реальном времени. Это особенно полезно для анализа больших архивов партий, которые могут содержать тысячи и даже миллионы партий.

Архитектура веб-приложения построена на принципах модульности, переиспользования кода и четкого разделения ответственности между компонентами, что

обеспечивает масштабируемость и гибкость решения. Интеграция шахматной логики с современными веб-технологиями создает уникальный пользовательский опыт, а централизованная система управления состоянием гарантирует консистентность данных. Важной особенностью является поддержка популярного формата записи шахматных партий PGN, что расширяет возможности интеграции с существующими шахматными платформами. Например, модульность архитектуры позволяет легко добавлять новые компоненты и улучшать существующие, что является важным аспектом для масштабируемости и гибкости решения. Переиспользование кода позволяет уменьшить количество дублирующегося кода и улучшить качество кода, что также является важным аспектом для разработки. Это особенно важно для обеспечения быстрой и эффективной разработки, что является ключевым аспектом для создания современных и надежных веб-приложений.

В процессе разработки успешно решены ключевые задачи, включая создание системы аутентификации, интерактивной шахматной доски, механизма визуализации интересных моментов и адаптивного интерфейса. Производительность системы позволяет обрабатывать партию за 30-45 секунд, что делает возможным ее применение для анализа больших архивов в реальном времени. Система аутентификации позволяет пользователям безопасно входить в систему и получать доступ к своим данным, что является важным аспектом для безопасности и конфиденциальности данных. Интерактивная шахматная доска позволяет пользователям просматривать и анализировать партии, что является важным аспектом для пользовательского опыта. Это особенно полезно для обучения и улучшения навыков игроков, что является ключевым аспектом для создания современных и удобных веб-приложений.

Таким образом, проект HiChess не только успешно выполнил поставленные задачи, но и создал технологическую базу для дальнейшего развития автоматизированных систем анализа шахматных партий. Решение открывает новые возможности для тренеров, комментаторов и образовательных платформ, сочетая высокую точность аналитики с удобным и отзывчивым интерфейсом. Например, технологическая база позволяет легко добавлять новые компоненты и улучшать существующие, что явля-

ется важным аспектом для дальнейшего развития системы. Высокая точность аналитики позволяет системе точно выявлять интересные моменты в партиях, что является важным аспектом для пользователей. Это особенно полезно для обучения и улучшения навыков игроков, что является ключевым аспектом для создания современных и удобных веб-приложений.

В результате проведенного исследования были получены значимые результаты, которые могут быть использованы для дальнейшего развития и улучшения IT-решения для анализа шахматных партий. Были предложены направления для дальнейших исследований, включая улучшение алгоритмов, оптимизацию моделей и расширение функциональности системы. Например, улучшение алгоритмов позволяет улучшить качество анализа и уменьшить количество ошибок, что является важным аспектом для дальнейшего развития системы. Оптимизация моделей позволяет улучшить производительность системы и уменьшить время обработки партий, что также является важным аспектом для дальнейшего развития системы. Это особенно важно для обеспечения быстрой и эффективной обработки данных, что является ключевым аспектом для создания современных и надежных веб-приложений. В результате выполнения работы были достигнуты значительные результаты, которые позволяют сделать вывод о высокой эффективности разработанного IT-решения для анализа шахматных партий. Были успешно реализованы алгоритмические подходы, использованы большие языковые модели и разработана собственная ML-модель, что позволило создать комплексное и надежное решение. Разработанный сайт с удобным интерфейсом позволяет пользователям легко загружать и анализировать шахматные партии, что значительно упрощает процесс обучения и улучшения навыков. Поставленные задачи были выполнены в полном объеме. Разработанное решение охватывает все ключевые аспекты анализа шахматных партий, включая выявление тактических приемов, обработку видеозаписей и интеграцию с существующими платформами. Высокие показатели эффективности и точность анализа подтверждают полноту решений. Для конкретного использования результатов работы рекомендуется интегрировать разработанное решение в образовательные платформы и тренировочные программы для шахматистов. Это позволит улучшить качество обучения и повысить уровень игры.

Также рекомендуется использовать систему для анализа больших архивов партий в реальном времени, что может быть полезно для комментаторов и аналитиков. Внедрение разработанного решения позволяет значительно сократить время анализа шахматных партий и уменьшить количество ошибок по сравнению с ручной разметкой. Это приводит к снижению затрат на анализ и повышению его точности, что в конечном итоге способствует улучшению качества обучения и повышению уровня игры. Разработанное решение соответствует лучшим достижениям в области анализа шахматных партий. Использование современных технологий и методов, таких как большие языковые модели и машинное обучение, позволяет достичь высокой точности и надежности анализа. Гибридный подход, сочетающий различные методы, обеспечивает более надежное выявление интересных моментов. В дальнейшем планируется улучшение алгоритмов и оптимизация моделей для повышения производительности системы. Также рассматривается расширение функциональности системы, включая интеграцию с новыми платформами и добавление новых методов анализа. Это позволит создать еще более эффективное и точное решение для анализа шахматных партий.

Таким образом, данная работа представляет собой значительный вклад в область анализа шахматных партий с использованием современных технологий и методов. Результаты исследования могут быть использованы для создания эффективных и точных IT-решений, способных анализировать шахматные партии и выявлять интересные моменты, что может быть полезно как для профессиональных шахматистов, так и для любителей этой игры. Например, значительный вклад в область анализа шахматных партий позволяет улучшить качество анализа и уменьшить количество ошибок, что является важным аспектом для пользователей. Эффективные и точные IT-решения позволяют пользователям точно анализировать партии и выявлять интересные моменты, что является важным аспектом для пользовательского опыта. Это особенно полезно для обучения и улучшения навыков игроков, что является ключевым аспектом для создания современных и удобных веб-приложений.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ruoss, A., Delétang, G., Medapati, S., et al. Amortized Planning with Large-Scale Transformers: A Case Study on Chess // arXiv preprint arXiv:2402.04494v2. – 2024. – URL: https://arxiv.org/abs/2402.04494 1913.

2. Zhang, Y., Han, X., Li, H., et al. Complete Chess Games Enable LLM Become A Chess Master // arXiv preprint arXiv:2501.17186v2. – 2025. – URL: https://arxiv.org/abs/2501.17186 21014.

3. Barthelemy, M. Statistical Analysis of Chess Games: Space Control and Tipping Points // arXiv preprint arXiv:2304.11425. – 2023. – URL: https://arxiv.org/abs/2304.11425.

4. Maharaj, S., Polson, N. Karpov's Queen Sacrifices and AI // arXiv preprint arXiv:2109.08149. – 2021. – URL: https://arxiv.org/abs/2109.08149.

5. Hamade, K., McIlroy-Young, R., et al. Designing Skill-Compatible AI: Methodologies and Frameworks in Chess // arXiv preprint arXiv:2405.05066v1. – 2024. – URL: https://arxiv.org/abs/2405.05066.

6. Gundawar, A., Li, Y., Bertsekas, D. Superior Computer Chess with Model Predictive Control, Reinforcement Learning, and Rollout // arXiv preprint arXiv:2409.06477v1. – 2024. – URL: https://arxiv.org/abs/2409.06477.

7. Maharaj, S., Polson, N., Turk, C. Chess AI: Competing Paradigms for Machine Intelligence // arXiv preprint arXiv:2109.11602v1. – 2021. – URL: https://arxiv.org/abs/2109.11602.

8. Acher, M., Esnault, F. Large-scale Analysis of Chess Games with Chess Engines: A Preliminary Report // arXiv preprint arXiv:1607.04186v1. – 2016. – URL: https://arxiv.org/abs/1607.04186.

9. Iqbal, A., Guid, M., Colton, S., et al. The Digital Synaptic Neural Substrate: A New Approach to Computational Creativity // arXiv preprint arXiv:1507.07058v2. – 2015. – URL: https://arxiv.org/abs/1507.07058.

10. Anbarci, N., Ismail, M. AI-powered mechanisms as judges: Breaking ties in chess // arXiv preprint arXiv:2210.08289v3. – 2022. – URL: https://arxiv.org/abs/2210.08289.

11. Silver, D., Hubert, T., Schrittwieser, J., et al. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm // arXiv preprint arXiv:1712.01815. – 2017. – URL: https://arxiv.org/abs/1712.01815.

12. McIlroy-Young, R., Sen, S., Kleinberg, J., et al. Aligning Superhuman AI with Human Behavior: Chess as a Model System // arXiv preprint arXiv:2006.01855. – 2020. – URL: https://arxiv.org/abs/2006.01855.

13. Romstad, T., Costalba, M., Kiiski, J. Stockfish NNUE: Efficiently Updatable Neural Networks for Chess Evaluation // ICGA Journal. – 2021. – Vol. 43, No. 3. – P. 177–189.

14. Wu, D. DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess // arXiv preprint arXiv:1711.09667. – 2017. – URL: https://arxiv.org/abs/1711.09667.

15. Lai, M. Giraffe: Using Deep Reinforcement Learning to Play Chess // arXiv preprint arXiv:1509.01549. – 2015. – URL: https://arxiv.org/abs/1509.01549.

16. Емельянов, А.В., Смирнов, И.Н. Использование нейросетей в шахматных партиях: анализ тактических комбинаций // Вестник цифровых технологий. – 2023. – № 2. – С. 45–52.

17. Тарасов, М.А. Применение эвристических алгоритмов в анализе шахматных позиций // Информационные технологии и системы. – 2022. – № 4(18). – С. 36–41.

18. Сидоров, Д.Н., Иванова, Е.А. Возможности применения искусственного интеллекта в спортивной аналитике: кейс шахмат // Труды ИСП РАН. – 2024. – Т. 36, № 1. – С. 80–92.

19. Кулагин, А.Ю. Сравнительный анализ шахматных движков с точки зрения оценки позиций // Программные системы: теория и приложения. – 2021. – № 3(48). – С. 61–70.

20. Смолин, П.Г. Использование технологии FFmpeg для генерации обучающих клипов из шахматных партий // Информационные технологии и математическое моделирование. – 2023. – № 5. – С. 94–99.

21. Дрынкин, В.Н., Набоков, С.А., Царева, Т.И. Неортогональная дискретизация как основа сжатия и восстановления видеоинформации // Известия РАН. Теория и системы управления. – 2019. – № 3. – С. 127–139 12.

22. Романенко, С.А. Алгоритмы искусственного интеллекта в шахматах: от минимакс-деревьев к глубокому обучению // Искусственный интеллект и принятие решений. – 2023. – № 1. – С. 22–35.

23. Гусев, Л.П. Методы машинного обучения в анализе шахматных эндшпилей // Компьютерные исследования и моделирование. – 2022. – Т. 14, № 4. – С. 901–915.

24. Федоров, Е.С. Генеративные модели в создании шахматных задач // Нейроинформатика. – 2024. – № 1. – С. 55–67.

25. Crest Admin, МГ. Сравнение шахматных программ разных лет // KasparovChess Forum. – 2016. – URL: https://kasparovchess.crestbook.com/threads/7231/3.

26. Moonlight. [Literature Review] Amortized Planning with Large-Scale Transformers: A Case Study on Chess // TheMoonlight.io. – 2024. – URL: https://www.themoonlight.io/review/amortized-planning-with-large-scale-transformers-a-case-study-on-chess 13.

27. Zhang, Y., Han, X., Li, H., et al. Complete Chess Games Enable LLM Become A Chess Master // Proceedings of NAACL 2025. – 2025. – P. 1–7. – URL: https://aclanthology.org/2025.naacl-short.1/ 14.

28. Ruoss, A., Delétang, G., Medapati, S., et al. Amortized Planning with Large-Scale Transformers: A Case Study on Chess // NeurIPS 2024 Poster. – 2024. – URL: https://neurips.cc/virtual/2024/poster/94747 5.

29. Leela Chess Zero Team. Leela Chess Zero: A Self-Improving Chess Engine // ICGA Journal. – 2020. – Vol. 42, No. 2. – P. 113–125.

30. Hsu, F.-H. Behind Deep Blue: Building the Computer that Defeated the World Chess Champion. – Princeton University Press, 2002. – 312 p.

ПРИЛОЖЕНИЕ А
Паспорт проекта

Название программы: Методы искусственного интеллекта и предиктивная аналитика в проектах дефектоскопии

Название команды: Мадагаскам

Team lead (ФИО, tg): Шаталов Максим Алексеевич Tg: @Megabasic

Ментор (ФИО, tg): Тихонов Федор Андреевич Tg: @fedoska_t

Паспорт проекта

Определение эвристик по поиску интересных моментов в шахматной партии

1. Общая информация

1. **Краткое описание проекта**:

Разработка сервиса для автоматической обработки PGN шахматных партий с целью выделения наиболее интересных моментов, а также создания привлекательного видеоконтента для шахматных любителей, видеоблогеров и платформ онлайн-турниров.

2. **Команда**:

a. Шаталов Максим Алексеевич — Teamlead
b. Авраменко Денис Александрович — Backend-разработчик
c. Агафонов Андрей Сергеевич — ML-инженер
d. Голосов Георгий Сергеевич — Аналитик
e. Ивченко Матвей Сергеевич — ML-инженер
f. Лапенко Карина Александровна — Frontend-разработчик
g. Мельцова Вероника Алексеевна — Backend-разработчик
h. Пономарев Артём Андреевич — ML-инженер

2. Цель проекта

1. **Цель проекта**:

Разработать сервис для автоматической обработки PGN шахматной партии, размечающий её на наиболее интересные отрезки.

2. **Ожидаемые результаты**:

Полностью автоматизированная система, интегрируемая в платформу idChess. Система способна обрабатывать шахматные партии, выделяя ключевые моменты, а также преобразовывать видеоконтент записи партии с использованием алгоритмов наложения эффектов и нейросетевых преобразований.

3. Задачи и процесс работы

1. Задач в процессе: 0.
2. Задач завершено: 189.
3. Заблокировано: 0.

4. Прогресс и результаты

3. **Текущий статус проекта**:

Проект выполнен в полном объеме, все цели достигнуты.

4. **Достигнутые результаты по задачам**:

I. Backend

a. Реализована система авторизации и аутентификации пользователей с использованием JWT (JSON Web Tokens)
b. Для анонимных пользователей доступна возможность обработки партий, но результаты не сохраняются в истории.
c. Каждая обработка сохраняется в базе данных PostgreSQL с привязкой к пользователю. История включает дата обработки, название партии, результат анализа.Реализован механизм временного хранения ссылок на результаты обработки партий.
d. Прием PGN и возврат диапазона интересных ходов
e. Реализован алгоритм для определения ключевых моментов партии на основе оценки GPT, ML, алгоритмической модели.

f. Имеется возможность запуска разных алгоритмов анализа

g. Возвращается JSON-ответ с диапазонами интересных ходов и их описанием.

h. Обработка видео и синхронизация с ходами. Оно синхронизируется с ходами из PGN-файла.

i. Результат обработки видео возвращается пользователю в виде файла или ссылки для скачивания.

j. Апи настроено для взаимодействия с фронтендом

DevOps-работы

Настроен Docker для контейнеризации и полного запуска всех сервисов.

Документация архитектуры

Описание архитектуры системы.

README-файл с инструкцией по развертыванию проекта.

Автодокументация API

. Настроена автодокументация API с использованием Swagger/OpenAPI.

a. Схема системы включает взаимодействие между фронтендом, бэкендом и моделями ИИ.

b. Схема базы данных.

II. Frontend

. Дизайн интерфейса

a. Личный кабинет пользователя

b. Реализованы страницы регистрации, входа и восстановления пароля

c. Анимация передвижения фигур

d. Страницы загрузки файлов, отображение результатов и истории обработок.

e. Отображение видео

III. Машинное обучение

. Реализована модель, которая анализирует PGN-файлы и выделяет ключевые моменты с высокой точностью

a. Обучены и протестированы две основные модели: Первая на основе GPT. Вторая на разметке от задач личесс.
b. Проведен сбор и анализ данных (EDA) с визуализацией характеристик датасета.
c. Реализованы Jupyter Notebook и Python-скрипты для обучения моделей.
d. Создан документ с описанием датасетов, этапов предобработки и параметров моделей.
e. Реализована алгоритмическая часть проекта

IV. Документация и отчеты

Заполнен таск-трекер с описанием всех этапов работы, дедлайнов и исполнителей.

Заполнен паспорт проекта

Разработан финальный отчет

5. **Риски**:

Основные риски включают сложность разработки моделей искусственного интеллекта, трудности с поиском качественных датасетов, возникающие вопросы по поводу точного определения интересного момента, а также ограниченные вычислительные ресурсы. Шахматные партии могут сильно различаться по сложности и стилю игры, что усложняет выбор универсального набора данных для обучения. В результате модели могут сталкиваться с трудностями в распознавании и анализе различных игровых ситуаций, что снижает их точность.

Субъективность интересного момента также создает дополнительные трудности в настройке моделей, так как необходимо учитывать множество факторов и критериев для определения интересных моментов.

И последнее – это то, что обучение и использование моделей искусственного интеллекта требуют значительных вычислительных мощностей, которые нам недоступны, приходится очень долго ждать для получения результата обучения.

5. Ресурсы и материалы проекта

6. **Используемые инструменты и технологии**:

Python, TensorFlow, PyTorch, ReactJS, PostgreSQL, Jupyter Notebook, Nuxt

7. **Ссылки на внешние ресурсы**:

Github: https://github.com/Madagascam/main

Research doc:
https://docs.google.com/document/d/1ZtR6wMTISPx3OQLQTrxmTzBFIbYKPHRXqxNJ2G9nnNE/edit?usp=sharing

Jupyter notebook по ии:
https://github.com/Madagascam/heuristic_extractor/blob/feature/board2vec/board2vec/experiments/experiment_04/booster.ipynb

Примеры обработки видео нашими алгоритмами:
https://drive.google.com/drive/folders/1-Iw8QqXnV7up3G5Fzm-IJPo2Ok0BHPVw

8. Данные

1. Источники данных: открытые шахматные базы данных, такие как Lichess, Chess.com, PGN-архивы турниров FIDE;

2. Формат данных: PGN-файлы шахматных партий с метаинформацией о ходе игры. Тренировочный датасет хранится в формате csv и содержит список ходов, рейтинги игроков, ссылку на игру и две метки: начало и конец интересного хода;

3. Предобработка данных: Энкодер составляет первичное представление доски в виде 12-канального разреженного тензора (“изображения”) из 0 и 1, которое свёрточная нейросеть, состоящая из одного residual блока и одного полносвязного слоя, переводит в плотный вектор-эмбеддинг. Полученная последовательность эмбеддингов, включая метаинформацию, поступает на вход целевой модели;

4. Модель: Модель градиентного бустинга на основе деревьев решений принимает на вход последовательность эмбеддингов позиций и выдаёт последовательность оценок интересности каждого хода, которая пост-обрабатывается алгоритмом для улучшения качества и снижения шумности. Алгоритм выделяет

ключевые ходы, вокруг которых выделяет интересные моменты. Наконец, короткие ответы выбрасываются, близкие друг к другу склеиваются.

EDA анализ датасета:

https://github.com/Madagascam/heuristic_extractor/blob/feature/board2vec/board2vec/EDA.ipynb

6. Комментарии и мысли команды

9. **Комментарии**:

1. Проект завершен.
2. Все цели по функционалу достигнуты
3. Удалось обучить модель и создать данные для датасета моделей

ПРИЛОЖЕНИЕ Б
Примеры промптов для взаимодействия с LLM

А.1 Промпт для извлечения нескольких хайлайтов с оценкой на основе детализированных инструкций (scope='multiple', focus='youtube', output_format='with_score')

You are a chess analysis assistant.

Your task is to identify all interesting segments (each max 20 half-moves) in the provided chess game based on the following criteria.

Focus on segments that are visually interesting and easy to understand for a general audience:

- Clear blunders (losing major pieces).
- Obvious tactical shots (forks, simple sacrifices).
- Checkmate sequences.
- Moments where the game dramatically shifts.

Avoid subtle positional play unless it ends spectacularly.

The game is provided as a list of moves with half-move indices (starting from 0).

Output ONLY a valid JSON list of lists, where each inner list contains exactly three integers: [[start_halfmove, end_halfmove, score], ...]

The score must be an integer from 1 to 10 (1=minor, 10=critical/brilliant).

If no interesting segments are found, output an empty list: []

Do not include any other text, explanations, or markdown formatting.

List of moves:

0. 1. d4

1. 1... g6

2. 2. c4

3. 2... Bg7

4. 3. e4

5. 3... c5

6. 4. d5

7. 4... d6

8. 5. Nc3

9. 5... e6

10. 6. Bd3

11. 6... Ne7

12. 7. Nf3

13. 7... O-O

14. 8. O-O

15. 8... Nd7

16. 9. Bg5

17. 9... h6

18. 10. Be3

19. 10... e5

20. 11. Qd2

21. 11... Kh7

22. 12. g4

23. 12... Nf6

24. 13. h3

25. 13... Nfg8

26. 14. Nh2

27. 14... Kh8

28. 15. h4

29. 15... f5

30. 16. f3

31. 16... Rf7

32. 17. a3

33. 17... Bd7

34. 18. b4

35. 18... b6

36. 19. b5

37. 19... Nc8

38. 20. exf5

39. 20... gxf5

40. 21. g5

41. 21... Nce7

42. 22. gxh6

43. 22... Bf6

44. 23. h5

45. 23... Rh7

46. 24. Bg5

47. 24... Nxd5

48. 25. Nxd5

49. 25... Bxg5

50. 26. Qc2

51. 26... Nxh6

52. 27. f4

53. 27... Bxf4

54. 28. Rxf4

55. 28... exf4

56. 29. Nf3

57. 29... Qg8+

58. 30. Kf1

59. 30... Qg3

60. 31. Qf2

61. 31... Ng4

62. 32. Qxg3

63. 32... fxg3

64. 33. Nf4

65. 33... Rg8

66. 34. Nh4

67. 34... Rg5

68. 35. Kg2

69. 35... Bc8

70. 36. Be2

71. 36... Ne3+

72. 37. Kg1

73. 37... Bb7

74. 38. Ra2

75. 38... Ng4

76. 39. Rd2

77. 39... Rgxh5

78. 40. Rxd6

79. 40... Rxh4

80. 41. Ng6+

81. 41... Kg8

82. 42. Nxh4

83. 42... Ne5

JSON Output:

A.2 Пример Few-Shot промпта для PGN-стиля и извлечения одиночного хайлайта (scope='single', output_format='simple')

You are a chess analysis assistant.

Your task is to identify the single most interesting segment (max 20 half-moves) in the provided chess game.

Output ONLY a valid JSON list containing exactly two integers: [start_halfmove, end_halfmove]

Do not include any other text, explanations, or markdown formatting.

Here are some examples of input and expected output:

Game:

0. 1. e4

1. 1... e5

2. 2. Nf3

3. 2... Nc6

4. 3. Nc3

5. 3... Bc5

6. 4. d3

7. 4... h6

8. 5. Be3

9. 5... Bb6

10. 6. Bxb6

11. 6... axb6

12. 7. a3

13. 7... d6

14. 8. Nd2

15. 8... Nf6

16. 9. Be2

17. 9... O-O

18. 10. Nb3

19. 10... Nh7

20. 11. h4

21. 11... Qf6

22. 12. Qd2

23. 12... Qg6

24. 13. g3

25. 13... Nf6

26. 14. h5

27. 14... Qh7

28. 15. O-O-O

29. 15... Be6

30. 16. Rdg1

31. 16... Bxb3

32. 17. cxb3

33. 17... Nd4

34. 18. Bd1

35. 18... Ng4

36. 19. Kb1

37. 19... Nf6

38. 20. g4

39. 20... c6

40. 21. g5

41. 21... hxg5

42. 22. Qxg5

43. 22... Ne8

44. 23. h6

45. 23... g6

46. 24. Bh5

47. 24... b5

48. 25. Bxg6

49. 25... fxg6

50. 26. Qxg6+

51. 26... Kh8

52. 27. Qg7+

53. 27... Nxg7

54. 28. hxg7+

55. 28... Kg8

56. 29. Rxh7

57. 29... Kxh7

58. 30. gxf8=Q

59. 30... Rxf8

60. 31. Rh1+

61. 31... Kg6

62. 32. Rh2

63. 32... Rf3

64. 33. Rg2+

65. 33... Kf6

66. 34. b4

67. 34... Rxd3

68. 35. Rg8

69. 35... Rf3

70. 36. Rf8+

71. 36... Ke7

72. 37. Rb8

73. 37... Rxf2

74. 38. Rxb7+

75. 38... Ke6

76. 39. Rh7

77. 39... Ne2

78. 40. Rh6+

79. 40... Kd7

80. 41. Rh7+

81. 41... Ke6

82. 42. Rh6+

83. 42... Kd7

84. 43. Rh7+

85. 43... Ke6

86. 44. Rh6+

87. 44... Kd7

88. 45. Rh7+

89. 45... Ke6

90. 46. Rh6+

91. 46... Kd7

92. 47. Rh7+

93. 47... Ke6

Correct Highlight Output:

[51, 55]

ПРИЛОЖЕНИЕ В
Результаты предварительного исследовательского анализа данных (EDA) для набора шахматных партий

Предварительный исследовательский анализ данных (EDA) был проведен на основе датасета full_labeled.csv. Целью анализа являлось выявление основных статистических характеристик и распределений ключевых параметров партий и хайлайтов для формирования репрезентативной выборки few-shot примеров. Полный Jupyter Notebook с анализом (EDA.ipynb) находится в репозитории проекта.

Б.1 Анализ распределения рейтингов игроков (Elo)

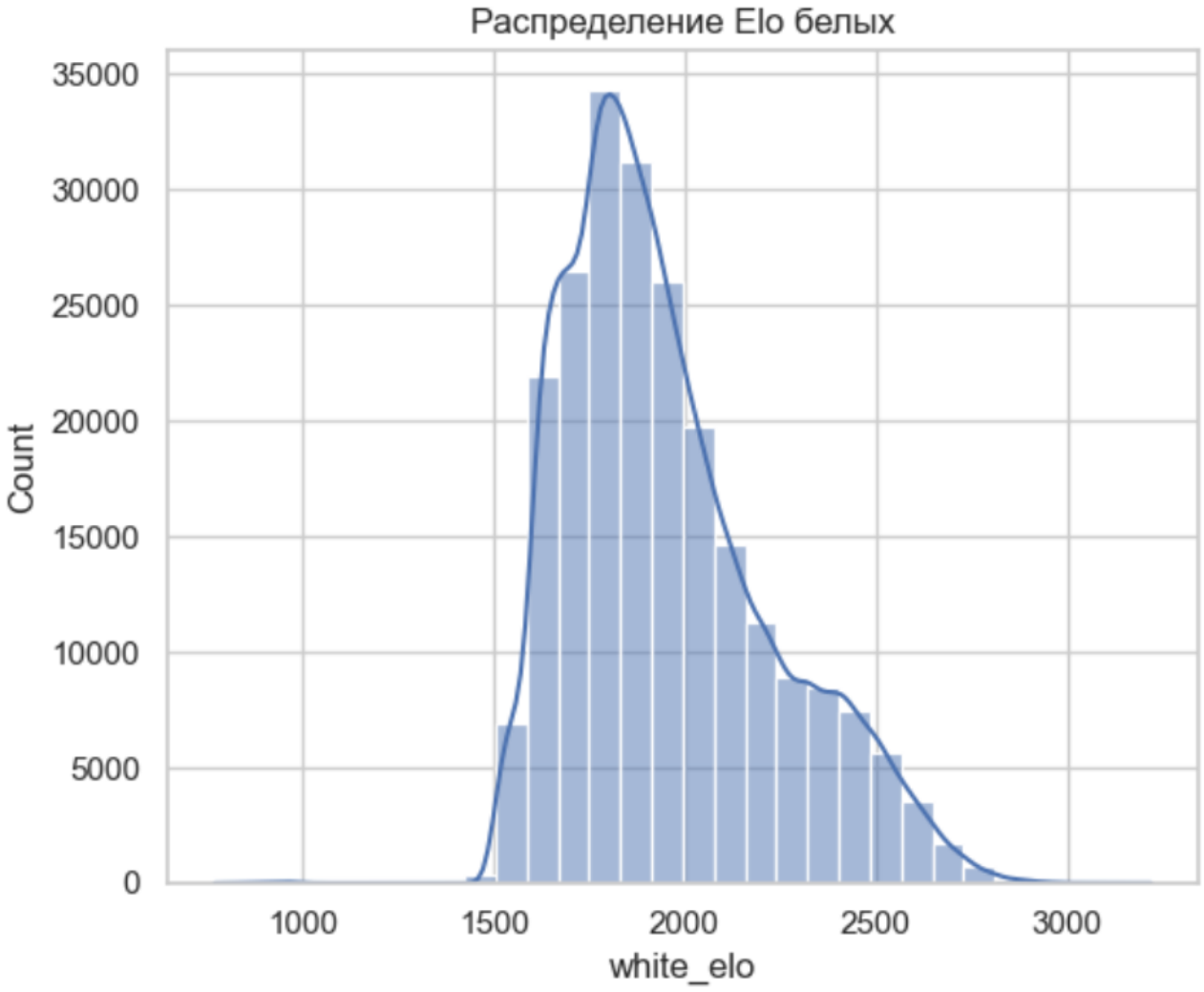

Рисунок Б.1 – Распределение рейтинга Elo белых игроков

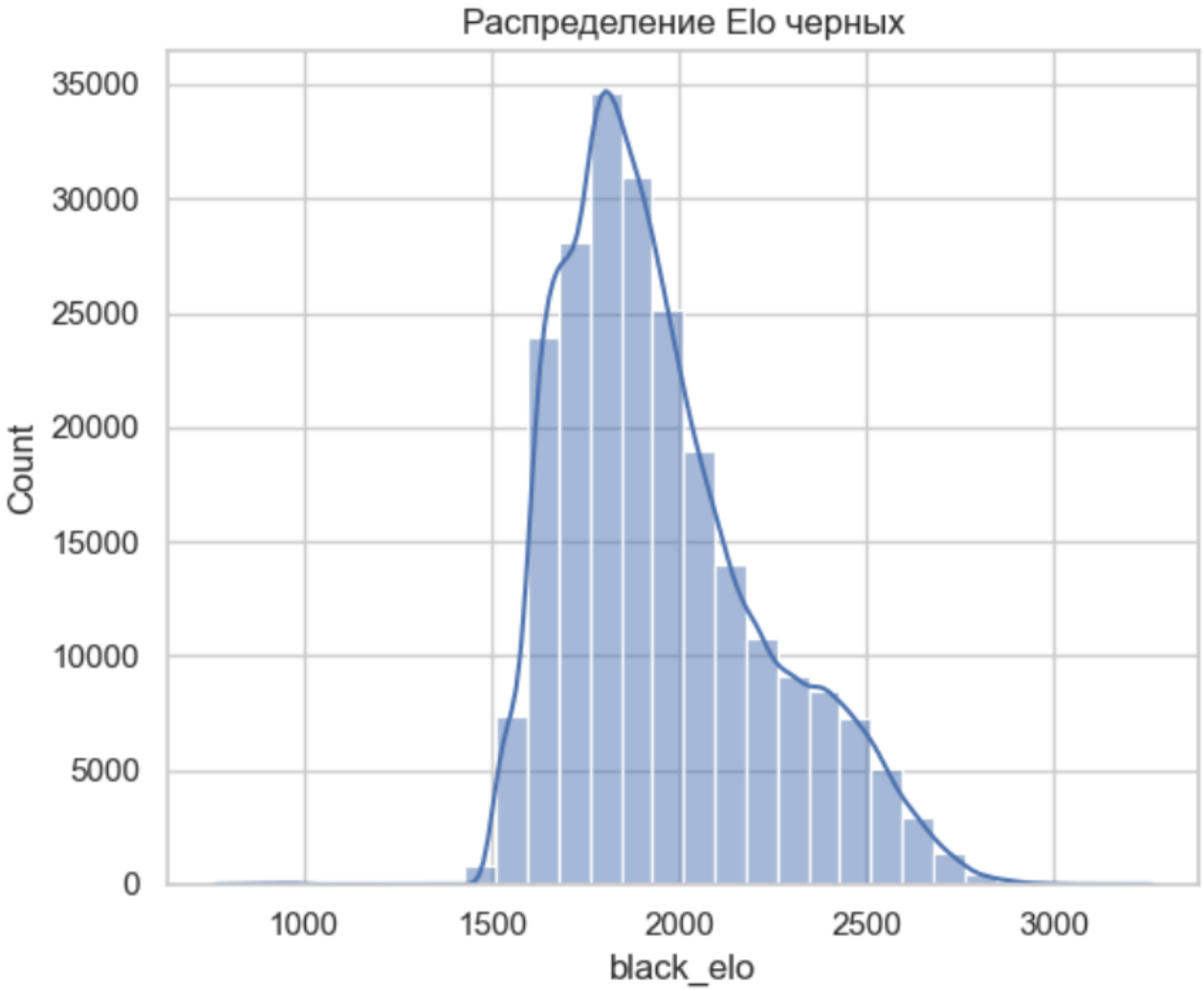

Рисунок Б.2 – Распределение рейтинга Elo черных игроков

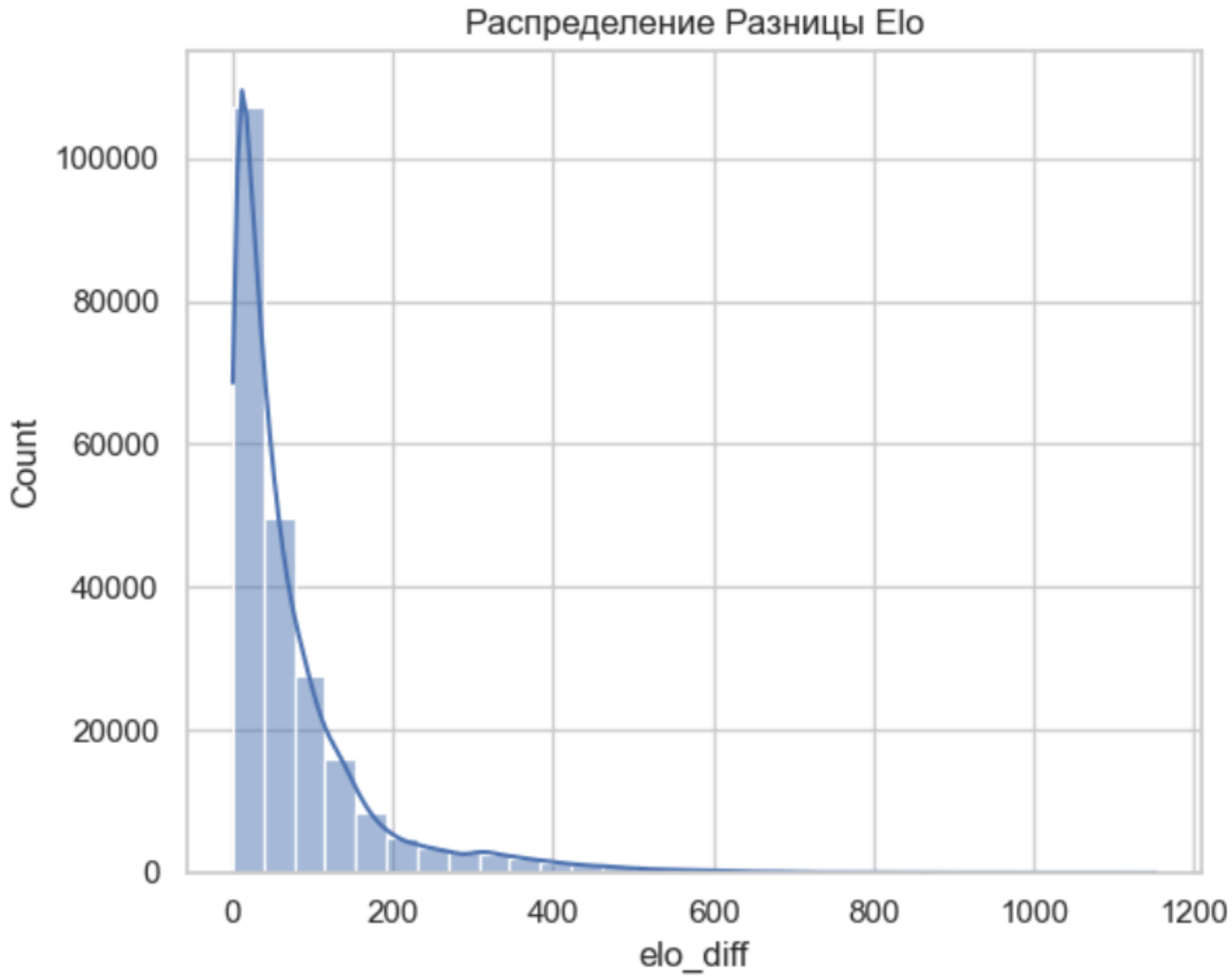

Рисунок Б.3 – Распределение абсолютной разницы рейтингов Elo между игроками

Краткие выводы по графикам:

Распределения Elo для белых и черных схожи, унимодальны с пиком в диапазоне 1700-1900 Elo и положительной асимметрией.

Большинство партий (около XX%, если есть данные) играется между соперниками с разницей в рейтинге менее 100-150 пунктов.

Б.2 Анализ распределения длины партии

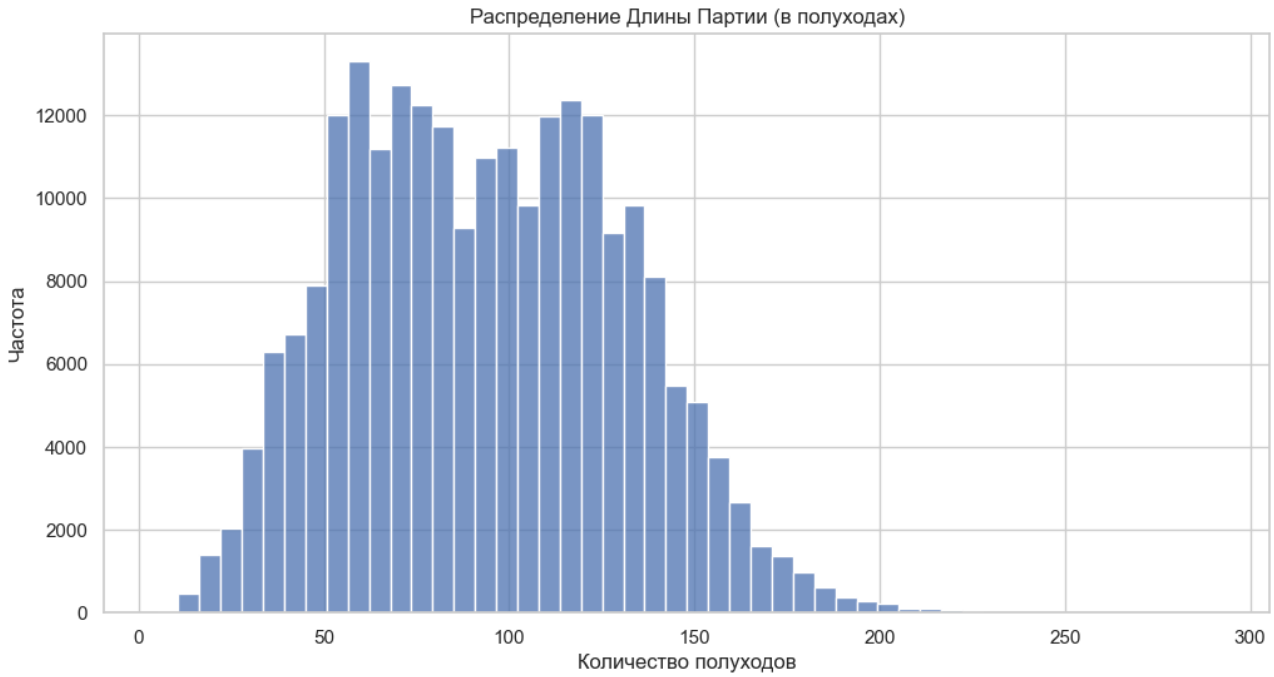

Рисунок Б.4 – Распределение длины партий в полуходах

Краткие выводы по графику и квантилям (можно вставить таблицу с квантилями, если она компактна, или просто текстом):

Распределение скошено вправо, медианная длина партии составляет 92 полухода (46 полных ходов).

Большинство партий (межквартильный размах) укладывается в диапазон от 64 до 122 полуходов.

Б.3 Анализ характеристик хайлайтов

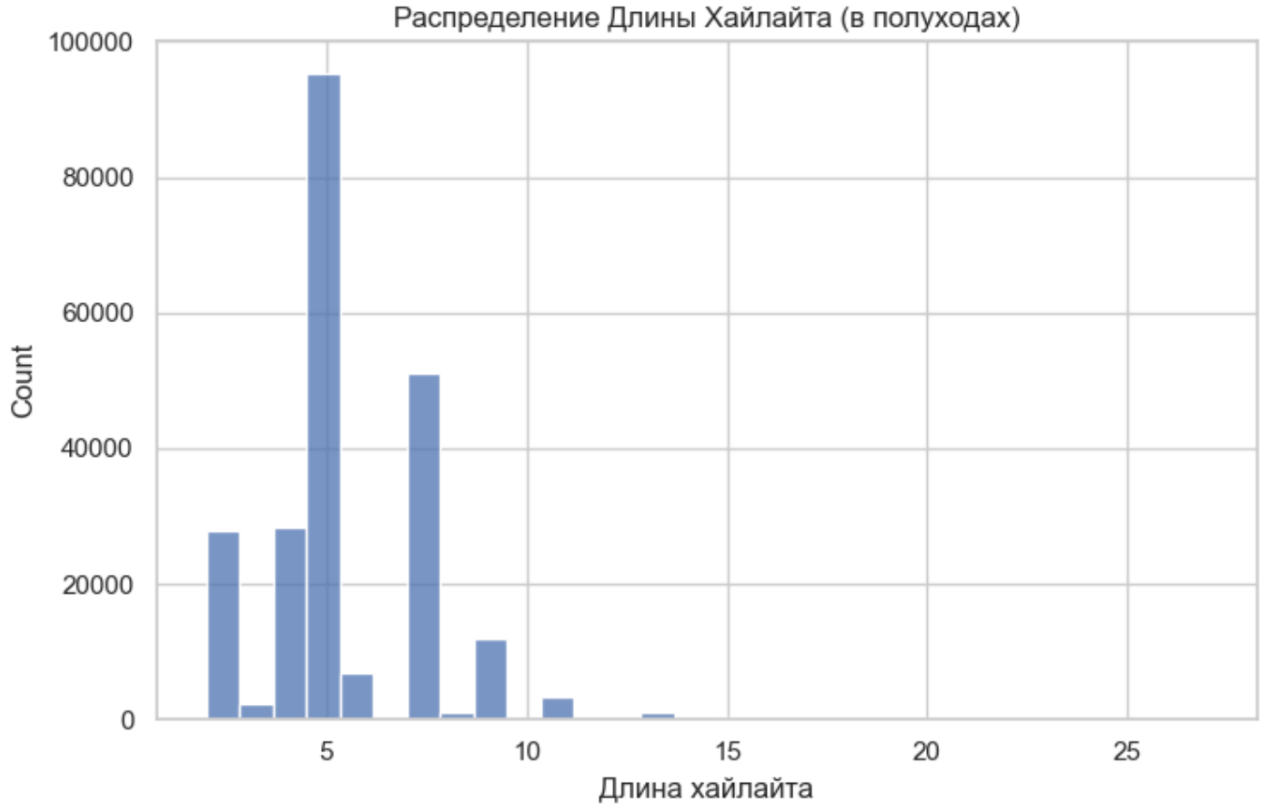

Рисунок Б.5 – Распределение длины выделенных хайлайтов в

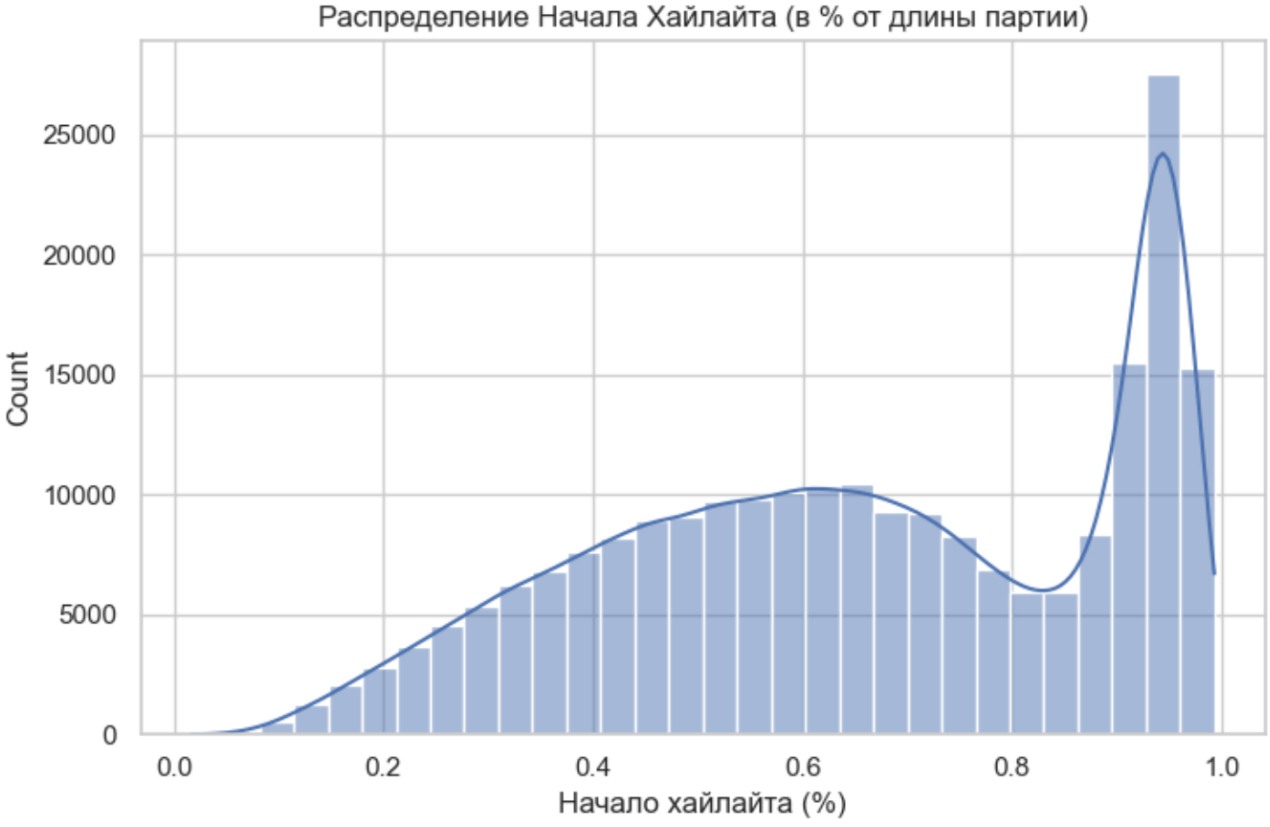

Рисунок Б.6 – Распределение относительного начала выделенных хайлайтов (в % от общей длины партии)

Краткие выводы по графикам и квантилям:

Хайлайты преимущественно короткие, с медианной длиной 5 полуходов и наиболее частыми значениями 5 и 7 полуходов.

Хайлайты чаще всего расположены во второй половине партии (медиана ~66% от длины партии), с выраженным пиком в самом конце игры (95-100%).

Б.4 Анализ взаимосвязей между признаками

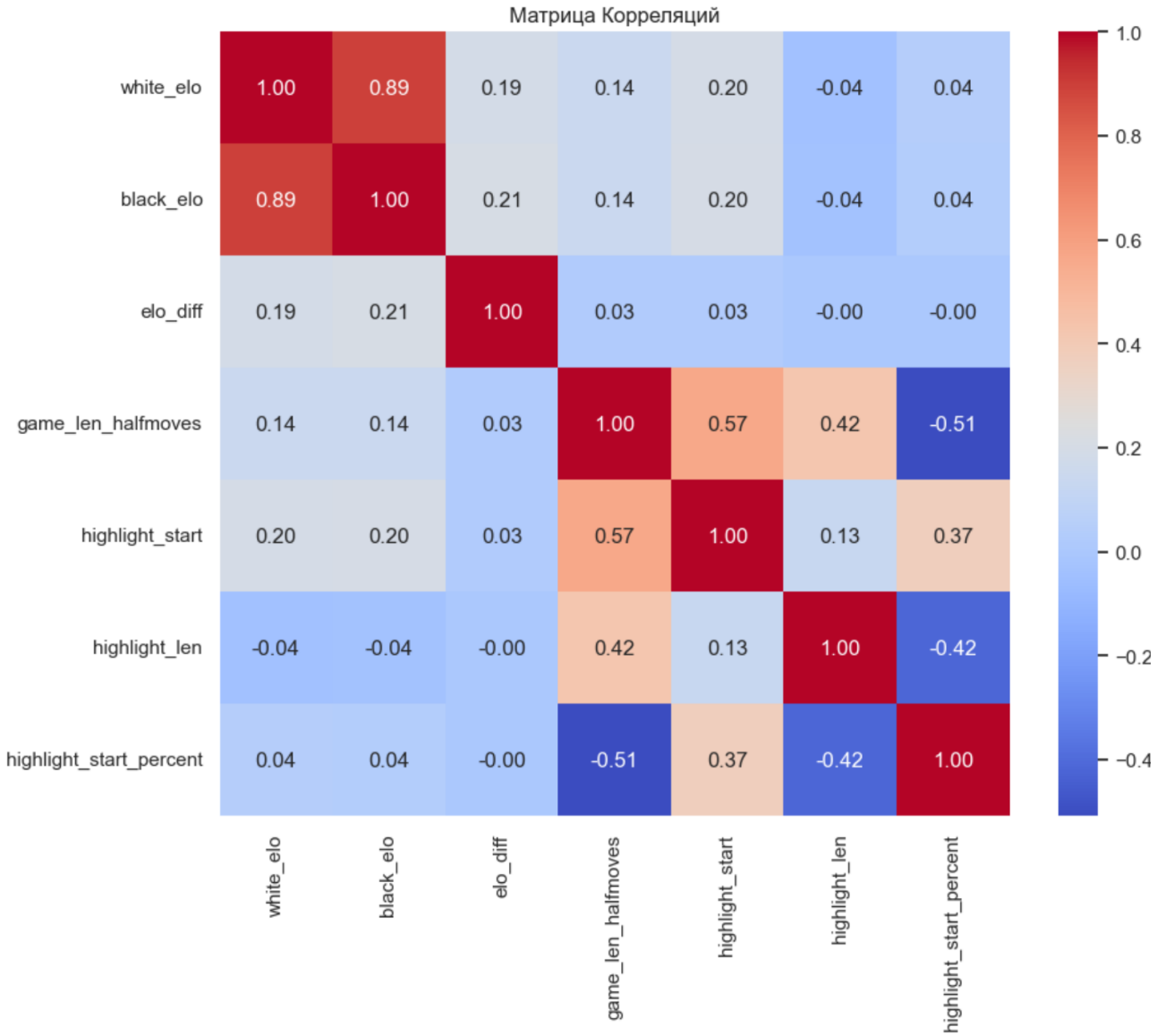

Рисунок Б.7 – Матрица корреляций между ключевыми признаками датасета

Краткие выводы по корреляциям и диаграммам рассеяния:

Длина и относительное начало хайлайта практически не зависят от рейтинга Elo игроков. Наблюдается отрицательная корреляция (-0.51) между общей длиной партии

и относительным началом хайлайта: в более длинных партиях хайлайты чаще смещены к середине, в коротких – к концу. Поздние хайлайты (начинающиеся ближе к концу партии) имеют тенденцию быть короче (отрицательная корреляция -0.42 между highlight_start_percent и highlight_len).

Б.5 Общие выводы по EDA и их влияние на выборку примеров

На основе проведенного анализа для формирования репрезентативной выборки few-shot примеров была применена стратегия стратифицированной случайной выборки. Датасет был разделен на 27 страт на основе терцилей следующих признаков: средний рейтинг Elo игроков, общая длина партии и относительное начало хайлайта. Данный подход позволил обеспечить сбалансированное представительство партий с различными характеристиками в обучающих примерах.