Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления
Кафедра интеллектуальных информационных технологий

К защите допустить:

Заведующий кафедрой ИИТ

_____ Д. В. Шункевич

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему:

ПЛАТФОРМА ДЛЯ ИНТЕРАКТИВНОГО ФОРМИРОВАНИЯ ЗАПРОСОВ К ЯЗЫКОВЫМ И ГЕНЕРАТИВНЫМ НЕЙРОСЕТЯМ

БГУИР ДП 1-40 03 01 054 ПЗ

Студент	Е. С. Самохвал
Руководитель	Ю. Б. Крапивин
Консультанты:	
от кафедры ИИТ	Ю. Б. Крапивин
по экономической части	Т. Л. Слюсарь
Нормоконтролёр	С. А. Самодумкин
Рецензент	

Минск 2025

Решением рабочей комиссии
допущен(а) к защите дипломного проекта

Председатель рабочей комиссии

_____ Д. В. Шункевич
(Подпись) (Инициалы и Фамилия)

____ _____ 2025 г.

Учреждение образования
Белорусский государственный университет
информатики и радиоэлектроники

Кафедра интеллектуальных информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой ИИТ
_____ Д. В. Шункевич
____ _____ 2025 г.

ЗАДАНИЕ
на дипломный проект

Обучающемуся Самохвал Евгений Сергеевич
(фамилия, собственное имя, отчество (если таковое имеется))

Курс 4 Учебная группа 121703

Специальность 1-40 03 01 Искусственный интеллект

Тема дипломного проекта Платформа для интерактивного формирования запросов к языковым и генеративным нейросетям
(наименование темы)

Утверждена руководителем УВО БГУИР №371-с от 10 февраля 2025 г.

Исходные данные к дипломному проекту: Язык разработки Python; Язык разработки TypeScript; Фрэймворк Vue; Фрэймворк FastAPI;FusionBrain API;DeepSeek API;база данных PostgreSQL

Перечень подлежащих разработке вопросов или краткое содержание расчетно-пояснительной записки:
Введение
1 Анализ подходов и технологий к разработке платформы для интерактивного формирования запросов к языковым и генеративным нейросетям
2 Проектирование модели платформы для интерактивного формирования запросов к языковым и генеративным нейросетям
3 Реализация платформы для интерактивного формирования запросов к языковым и генеративным нейросетям
4 Технико-экономическое обоснование разработки платформы для интерактивного формирования запросов к языковым и генеративным нейросетям
Заключение

Перечень графического материала (с точным указанием обязательных чертежей и графиков):
Диаграмма вариантов использования платформы — формат А1, лист 1.
Алгоритм оценки качества запроса — формат А1, лист 1.
Интерфейс платформы — формат А1, лист 1.
Диаграмма базы данных — формат А1, лист 1.
Диаграмма структуры запроса — формат А1, лист 1.
Диаграмма структуры платформы — формат А1, лист 1.

Консультанты по дипломному проекту (с указанием разделов, по которым они консультируют):
Раздел 1–3 — консультант от кафедры Ю.Б. Крапивин
Раздел 4 — консультант по экономической части Т.Л. Слюсарь

Примерный календарный график выполнения дипломного проекта

№ п/п	Наименование этапа дипломного проекта	Объем этапа, %	Срок выполнения этапа
1	Обзор литературных источников по теме, изучение проблемной области	10	10.02 – 14.03
3	Определение требований к реализации	10	14.03 – 10.04
4	Проектирование модели подсистемы	25	10.04 – 30.04
5	Реализация подсистемы	30	23.04 – 20.05
6	Экономическое обоснование принятого решения, определение экономической эффективности внедрения полученных результатов	10	01.05 – 20.05
7	Оформление расчетно-пояснительной записки	10	30.03 – 29.05
8	Оформление графического материала	5	01.05 – 29.05

Дата выдачи задания 10 февраля 2025 г.
Срок сдачи студентом законченного дипломного проекта 31 мая 2025 г.

Руководитель дипломного проекта _____ Ю. Б. Крапивин
(подпись) (инициалы, фамилия)

Подпись обучающегося _____

Дата ____ _____ 2025 г.

РЕФЕРАТ

ПЛАТФОРМА ДЛЯ ИНТЕРАКТИВНОГО ФОРМИРОВАНИЯ ЗАПРОСОВ К ЯЗЫКОВЫМ И ГЕНЕРАТИВНЫМ НЕЙРОСЕТЯМ: дипломный проект/ Е. С. Самохвал. – Минск : БГУИР, 2025. – п.з. – 79 с., чертежей (плакатов) – 6 л. формата А1.

Дипломный проект выполнен на 6 листах А1 с пояснительной запиской на 79 страницах, без приложений справочного или информационного характера. Пояснительная записка включает 4 раздела, 27 рисунков, 3 таблицы, 9 формул, 29 литературных источников.

Целью дипломного проекта является создание веб-платформы, обеспечивающей интерактивное формирование, структурирование и оптимизацию запросов к языковым и генеративным нейросетям с возможностью последующей интеграции в корпоративные и образовательные процессы.

Предметом исследования является архитектура, методы оптимизации и интерфейсы систем интерактивного формирования запросов к LLM и генеративным моделям.

Данный программный модуль позволяет создавать качественные, структурированные и адаптируемые запросы к нейросетям с учетом контекста, цели и формата ответа.

В первом разделе пояснительной записки проведен анализ существующих платформ и методик оптимизации запросов, сформулированы критерии качества и предложен сравнительный обзор.

Во втором разделе спроектирована модель системы, включая архитектуру платформы, логические и физические модели, информационные потоки и безопасность.

В третьем разделе реализованы клиентская часть на Vue 3, серверная логика на FastAPI, а также интеграция с API FusionBrain и DeepSeek, проведена валидация и тестирование.

В четвертом разделе приведено технико-экономическое обоснование разрабатываемого программного продукта, включая расчет затрат и оценку эффективности (NPV, ROI, PI).

Результатом дипломного проектирования является программа, обеспечивающая пользовательски-ориентированный интерфейс и высокое качество взаимодействия с языковыми моделями, реализованная по принципам модульности и безопасности.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ

ACID – Atomicity, Consistency, Isolation, Durability (атомарность, согласованность, изоляция, долговечность);

AI – Artificial Intelligence (искусственный интеллект);

APE – Automatic Prompt Engineer (автоматический инженер запросов);

API – Application Programming Interface (программный интерфейс приложений);

ASGI – Asynchronous Server Gateway Interface (асинхронный шлюзовый интерфейс сервера);

AWS – Amazon Web Services (облачная платформа Amazon);

CDN – Content Delivery Network (сеть доставки контента);

CI – Continuous Integration (непрерывная интеграция);

CI/CD – Continuous Integration / Continuous Delivery (непрерывная интеграция / непрерывная доставка);

CPU – Central Processing Unit (центральный процессор);

CUDA – Compute Unified Device Architecture (универсальная архитектура вычислений наGPU отNVIDIA);

DB – DataBase (база данных);

DevOps – Development & Operations (комплекс практик для объединения процессов разработки и эксплуатации);

DoS – Denial of Service (отказ в обслуживании);

FaaS – Functions as a Service (функции как сервис);

GPT – Generative Pre-trained Transformer (генеративно-дополнительно обученная модель-трансформер);

GPU – Graphics Processing Unit (графический процессор);

HTML – HyperText Markup Language (язык гипертекстовой разметки);

HTTP – HyperText Transfer Protocol (протокол передачи гипертекста);

HTTPS – HyperText Transfer Protocol Secure (защищённый протокол передачи гипертекста);

IEEE – Institute of Electrical and Electronics Engineers (Институт инженеров по электротехнике и электронике);

ISO – International Organization for Standardization (Международная организация по стандартизации);

IT – Information Technology (информационные технологии);

JWT – JSON Web Token (JSON-веб-токен);

LLaMA – Large Language Model Meta AI (семейство больших языковых моделей от Meta AI);

LLM – Large Language Model (большая языковая модель);

ML – Machine Learning (машинное обучение);

NLP – Natural Language Processing (обработка естественного языка);

ORM – Object-Relational Mapping (объектно-реляционное отображение);

OWASP – Open Web Application Security Project (проект по веб-безопасности);

RAM – Random Access Memory (оперативная память);

RBAC – Role-Based Access Control (управление доступом на основе ролей);

REST – Representational State Transfer (архитектурный стиль взаимодействия клиент-сервер);

RL – Reinforcement Learning (обучение с подкреплением);

SPA – Single Page Application (одностраничное веб-приложение);

SQL – Structured Query Language (язык структурированных запросов);

SSH – Secure Shell (безопасная оболочка удалённого доступа);

SSL – Secure Sockets Layer (уровень защищённых сокетов);

T5 – Text-to-Text Transfer Transformer (модель для преобразования текста);

UI – User Interface (пользовательский интерфейс);

VM – Virtual Machine (виртуальная машина);

VRAM – Video Random Access Memory (видеопамять).

БД – база данных;

ГОСТ – Государственный стандарт;

ООО – Общество с ограниченной ответственностью;

ОЗУ – оперативная память;

ОС – операционная система;

ПК – персональный компьютер;

СУБД – система управления базами данных.

СТБ – Стандарт Республики Беларусь;

СУОТ – система управления охраной труда;

ЗАО – Закрытое акционерное общество;

ТНПА – технические нормативно-правовые акты.

ВВЕДЕНИЕ

За последние пять лет область *prompt-engineering* и интерактивных систем поддержки работы с крупными языковыми и генеративными нейронными сетями (LLM / GDM) продемонстрировала существенные достижения. С выходом моделей GPT-3.5 / GPT-4, Claude 2, LLaMA 2, DeepSeek-LLM и аналогов стали доступны инструменты, обеспечивающие качество генерации текста и изображений, сопоставимое с экспертным уровнем. Появление автоматических оптимизаторов запросов (AISEO Prompt Enhancer, PromptPerfect), визуальных конструкторов для Midjourney, Stable Diffusion и гибридных диалоговых ассистентов (OctiAI) вместе с методиками chain-of-thought, self-consistency и few-shot-обучения позволило резко повысить точность и стабильность выводов моделей при решении прикладных задач анализа текста, креативного письма и генерации мультимедиа. Актуальность разработки платформ, упрощающих формирование и оптимизацию запросов, обусловлена постоянным ростом числа конечных пользователей, не обладающих глубокими знаниями в области ИИ, одновременно предъявляющих повышенные требования к надёжности и повторяемости результатов.

Цель дипломного проектирования — разработка веб-платформы, обеспечивающей *интерактивное формирование, структурирование и оптимизацию запросов* к языковым и генеративным нейросетям с возможностью последующей интеграции в корпоративные и образовательные процессы.

В основу проектирования положены следующие принципы:

– *Модульность и микросервисная архитектура* — разделение подсистем (клиент Vue 3, сервер FastAPI) с независимым масштабированием;

– *Непрерывная интеграция/доставка (CI/CD)* и контроль качества кода средствами GitHub Actions и SonarQube;

– *Эвристики prompt-engineering* — использование структурных делимитеров, chain-of-thought-шаблонов, автоматического рефрейзинга и скоринговых метрик качества запроса;

– *Пользовательско-центричный дизайн* — адаптивный интерфейс drag-and-drop, поддержка многоязычия и доступность WCAG 2.1 AA;

– *Информационная безопасность и конфиденциальность* — OWASP Top-10 hardening, RBAC, шифрование данных в покое и транзите.

Структура пояснительной записки отражает последовательное решение поставленных задач:

1 Анализ подходов и технологий. Выполняется сравнительный анализ современных методик улучшения детализированности и структу-

ры запросов, рассматриваются существующие платформы, проводится классификация архитектурных вариантов и выбор целевых инструментов разработки. *Задача раздела — обоснование технологической и методической базы проекта.*

2 Проектирование модели платформы. Определяются целевая аудитория, функциональные требования, информационные потоки и архитектура; формализуются алгоритмы оптимизации запросов, схема БД PostgreSQL + pgvector и модели безопасности. *Задача раздела — разработка логической и физической модели системы.*

3 Реализация программных компонентов. Описываются детали клиентского приложения (Vue 3 + Pinia), серверного API (FastAPI), ML-сервисов, интеграции с FusionBrain и DeepSeek, а также методика оценки качества сформированных запросов. *Задача раздела — демонстрация работоспособности и экспериментальная валидация.*

4 Технико-экономическое обоснование. Рассчитываются затраты на разработку и эксплуатацию, экономический эффект от внедрения и показатели эффективности проекта (NPV, ROI, PI). *Задача раздела — подтверждение экономической целесообразности.*

Заключение. Подводятся итоги исследования, формулируются выводы и направления дальнейшего развития платформы.

Дипломный проект выполнен *самостоятельно*, проверен системой «Антиплагиат»; процент оригинальности соответствует нормам кафедры ИИТ. Все заимствования оформлены ссылками на публикации, приведённые в «Списке использованных источников».

1 АНАЛИЗ ПОДХОДОВ И ТЕХНОЛОГИЙ К РАЗРАБОТКЕ ПЛАТФОРМЫ ДЛЯ ИНТЕРАКТИВНОГО ФОРМИРОВАНИЯ ЗАПРОСОВ К ЯЗЫКОВЫМ И ГЕНЕРАТИВНЫМ НЕЙРОСЕТЯМ

Платформы для интерактивного формирования запросов к языковым и генеративным нейросетям помогают пользователям составлять эффективные и подробные инструкции для моделей. Крупные языковые модели (Large Language Models, `LLM`) чувствительны к тому, как сформулирован запрос: четкость, контекст и структура запроса напрямую влияют на качество ответа. `Prompt engineering` – это практика улучшения запросов для получения более релевантных и точных результатов от нейросетей[1].В данном исследовании рассматриваются подходы к улучшению детализации и структуры запросов, анализируются существующие системы для генерации подсказок, обсуждаются архитектурные варианты построения таких платформ и возможные модели (NLP и ML) для автоматического улучшения запросов.

1.1 Методы улучшения детализации и структуры запросов

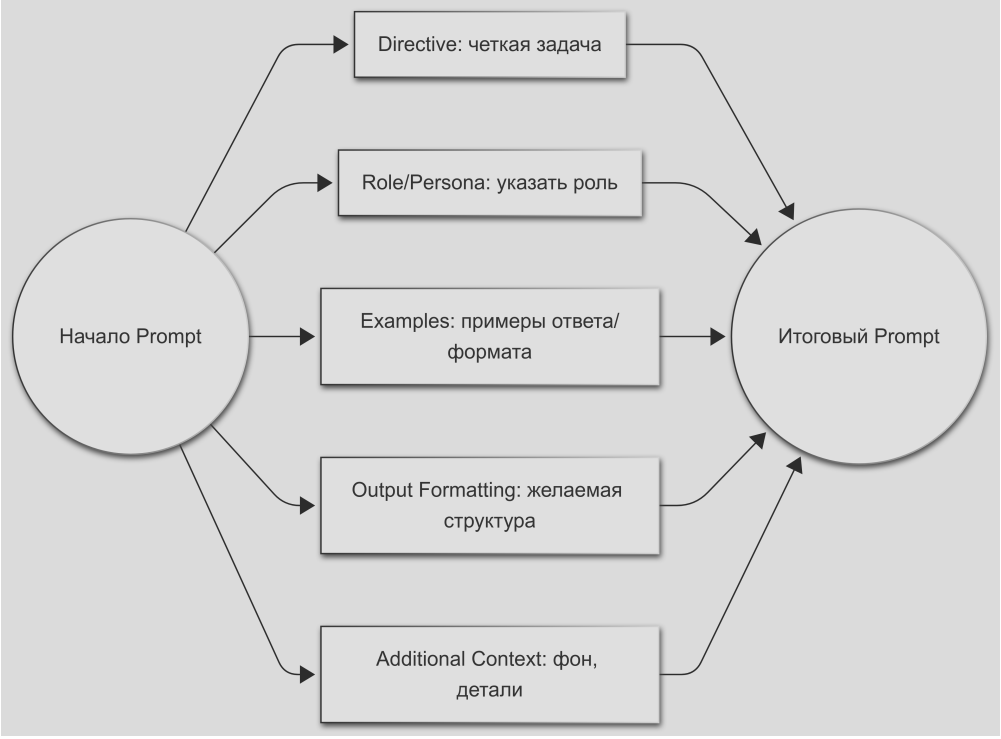

Рисунок 1.1 – Аспекты качественного запроса

Качественный запрос обычно содержит несколько ключевых элементов, обеспечивающих модели необходимый контекст[2]; они представлены на рисунке 1.1. Во-первых, четкая директива (The Directive) – основная инструкция, указывающая модели, что нужно сделать. Без ясной директивы модель может дать расплывчатый или не относящийся к делу ответ[2]. Поэтому запрос должен содержать конкретный глагол действия и формулировать задачу напрямую (например, "составь список из пяти книг по теме... вместо общего "расскажи что-нибудь интересное").

Во-вторых, полезно приводить образец (Examples) выполнения задачи, особенно для сложных или специфичных запросов. Примеры в запросах демонстрируют желаемый формат или стиль ответа, помогая модели понять ожидаемый результат [2]. Например, прежде чем попросить модель переводить предложения, можно показать один-два примера исходного текста и правильного перевода – тогда модель с большей вероятностью продолжит в заданном формате[2].

В-третьих, можно задать модели роль или персону (Role/Persona). Указывая, что "Ты – эксперт-историк"или "Представь себя профессиональным переводчиком мы настраиваем тон и стиль ответа под нужный нам контекст [2]. Такой явный контекст задаёт модели рамки: например, ответ "от лица врача"будет более формальным и содержательным в медицинской тематике. Чёткое определение роли помогает повысить релевантность и точность ответа в требуемом стиле.

Кроме того, стоит явно указывать желаемый формат вывода (Output Formatting) – например, запросить ответ в виде маркированного списка, таблицы или конкретной структуры. Наконец, полезно добавить дополнительную информацию и контекст (Additional Information): любой фон, ограничения или детали, которые могут помочь модели. Чем более конкретно очерчена задача и условия, тем более целенаправленным будет ответ модели. Интерактивные генераторы подсказок часто предлагают заполнить такие поля, как роль, цель, тон ответа, длина ответа, дополнительные детали, чтобы ничего не упустить при формулировании запроса[3].

Оптимизация структуры запроса. Помимо содержания, важно и то, как структурирован запрос. Рекомендуется явно разделять разные части запроса – контекст, сам вопрос, ограничения – либо визуальными разделителями, либо форматированием. Практика показывает, что использование разграничителей (делимитеров), таких как тэги XML, тройные кавычки ... или угловые скобки для секций, повышает ясность структуры и понимание модели[1]. Например, можно оформить запрос с секциями `<context>...</context>|`, `<user_request>...</user_request>`, `<constraints>...</constraints>`, что явно разделит вводные данные, собственно вопрос пользователя и особые требования[1].

Хорошо структурированный ввод помогает модели не перепутать раз-

личную информацию и соблюдать заданные рамки при генерации ответа. Исследования по инжинирингу запросов отмечают, что чёткие границы между разными типами данных в запросе значительно улучшают понимание и точность ответов модели[1].

Другой приём для сложных задач – это так называемый *Chain-of-Thought* (цепочка мыслей). В запрос явно закладывается пошаговое рассуждение: мы просим модель сначала размышлять или вывести промежуточные шаги, прежде чем дать итоговый ответ. Такой *Chain-of-Thought prompting* заставляет модель разбить сложную задачу на более простые части и решать их последовательно[1]. Исследования показывают, что без структуры модели путаются в многошаговых рассуждениях, тогда как запрос в формате "Шаг 1: ...; Шаг 2: ...; Вывод: ..."даёт более логичные результаты[1]. Например, для задачи решения головоломки или математической задачи, можно явно попросить: "Реши задачу пошагово: сначала проанализируй условия, затем вычисли промежуточные значения, и в конце дай ответ". Такая структура заставляет нейросеть имитировать процесс размышления человека и выдавать более обоснованные ответы.

Уточнение и детализация. Один из принципов хорошего запроса – максимальная конкретность. Чем конкретнее и однозначнее сформулирован запрос, тем меньше шансов на размытый ответ. Практические советы включают: задавать конкретные вопросы, избегать жаргона и двусмысленностей, явно прописывать, что именно нужно на выходе[1]. Например, вместо расплывчатого "Расскажи про экономику лучше спросить: "Дай краткий обзор современной экономики США, упомянув ВВП, уровень безработицы и основные вызовы, в 2–3 абзацах". Здесь чётко указана тема, аспекты, которые надо осветить, и даже желаемый объём ответа. Такой подход задаёт модели понятную задачу и формат ответа[3].

Если требуется определённый стиль или тон, стоит это явно оговорить. Например: "Ответь дружелюбным тоном и простыми словами, понятными ребёнку 10 лет". Или: "Предоставь ответ формально, в академическом стиле, с цитированием источников". Эти уточнения направляют модель на нужный стиль изложения. В интерактивных генераторах подсказок подобные параметры часто выносятся как отдельные настройки (tone/style, reading level), чтобы пользователь не забыл их указать[3].

Работа с длинными запросами и контекстом. Отдельная задача – как эффективно обрабатывать очень объёмные вводы или контекст. Современные LLM имеют ограничение на размер контекста (количество токенов), поэтому при превышении лимитов приходится искать обходные пути. Один из подходов – разбиение длинного ввода на части. Если у нас длинный документ, который нужно проанализировать, можно разбить его на несколько секций и отправлять модели последовательно, объединяя затем результаты. Разработчики отмечают, что в таком случае полезно между вызовами

модели передавать краткие резюме предыдущих частей, чтобы сохранять контекст [4]. Проще говоря, можно организовать итеративное суммирование: сначала модель резюмирует часть текста, затем этот резюме включается при обработке следующей части и так далее.

В ответе OpenAI сообщества отмечено: "Нужно разбить документ на секции и объединить полученные резюме. Если предыдущий контекст важен для последующих секций, попробуйте добавлять сокращённый контекст (например, заголовки или краткие выводы предыдущих секций) в новый запрос"[4]. Такой подход позволяет обойти ограничение по токенам, жертвуя деталями, но сохраняя ключевую информацию.

Когда используется длинный контекст, есть и другой неожиданный трюк. Документация Anthropic (модель Claude) рекомендует размещать объёмные данные в начале запроса, а сам вопрос – ближе к концу [5]. В тестах с моделью Claude было показано, что если длинный текст (20k+ токенов) поместить перед инструкциями и вопросом, качество ответа заметно возрастает (в некоторых случаях на 30% для сложных мультидокументных запросов) [5]. Иными словами, модель лучше усваивает длинный контекст, если он дан сначала, а уже затем – конкретный запрос пользователя. Этот приём улучшает точность ответа, особенно когда нужно учесть сразу несколько больших фрагментов информации. Помимо порядка, Anthropic предлагает использовать специальные разметки, например, заключать важные цитаты из контекста в теги `<quotes>` и просить модель опираться именно на эти цитаты при ответе [5]. Такой структурированный подход помогает модели сфокусировать внимание на релевантной информации даже при очень большом объёме данных. Наконец, улучшение запроса – это зачастую итеративный процесс. Рекомендуется протестировать запрос, посмотреть на ответ модели и при необходимости уточнить или переформулировать запрос. 90% успешного инжиниринга запросов – это эксперименты и доработки, и лишь 10% – само изначальное написание запроса [6]. Можно применить рефлексию модели: например, попросить модель сначала проверить свой ответ на соответствие запросу или сгенерировать несколько вариантов ответа и выбрать наиболее консистентный. Такой приём называется self-consistency: модель генерирует несколько решений и затем путем сравнения выбирается наиболее частый или подходящий – это повышает надёжность результата при сложных вопросах. Также модель можно попросить объяснить, почему её ответ правильный, или найти ошибки – это помогает в критических задачах снизить количество фактических ошибок[1]. Эти техники относятся уже к продвинутым стратегиям (reflective prompting), где модель используется не только для выдачи ответа, но и для самопроверки или улучшения своего же вывода[7]. Подводя итог, чтобы получить от нейросети детальный и точный ответ, нужно максимально конкретизировать запрос, структурировать его на понятные ча-

сти (роль, задача, формат ответа, контекст, примеры) и при необходимости разбивать сложные задания на последовательные этапы. Внимательное отношение к формулировке запроса способно значительно повысить качество результатов модели[1]. Ниже рассмотрим, как эти принципы реализованы в существующих инструментах и платформах для генерации и оптимизации запросов.

1.2 Обзор существующих систем для генерации запросов

С ростом популярности LLM появилось множество сервисов, помогающих пользователям создавать или улучшать запросы. Рассмотрим несколько систем, их возможности, преимущества и ограничения:

1 Prompt Generator for ChatGPT (CopilotWorks) – веб-инструмент для быстрого составления продуманных запросов к ChatGPT и аналогичным моделям. Пользователь заполняет готовые поля: выбирает роль (эксперт, учитель, переводчик и т.д.), формулирует цель запроса, задаёт желаемый тон или формат ответа (например, совет, объяснение, краткое резюме), ограничивает объём ответа (количество абзацев или слов) и добавляет детали или контекст[3]. На основе этих параметров генератор собирает готовый текст запроса, который можно скопировать и отправить в ChatGPT[3]. Инструмент фактически реализует лучшие практики создания запросов – например, предлагает указать роль модели и чётко сформулировать цель, добавлять контекст для детализации[3]. Преимущества: простота и скорость – даже неопытный пользователь за несколько кликов получит структурированный запрос. Благодаря выпадающим спискам и подсказкам снижается вероятность забыть важный параметр (тон, формат и проч.). Такой подход обеспечивает четкость и полноту запроса, что повышает качество ответа модели. Недостатки: шаблонность – система опирается на предустановленные категории ролей, тонов и т.д., что может ограничивать гибкость. Если задача нестандартна и выходит за рамки предусмотренных опций, пользователю придётся вручную дорабатывать сгенерированный запрос. Кроме того, генератор сам по себе не гарантирует идеальный результат – он лишь структурирует запрос, но не использует ИИ для его оптимизации. Тем не менее, как отправная точка для инжиниринга запросов этот инструмент очень полезен.

2 AISEO Prompt Enhancer – автоматизированный инструмент улучшения запросов, доступный как плагин для ChatGPT. Он был разработан компанией AISEO (известной по SEO-инструментам) и позволяет автоматически перефразировать и обогащать введённый пользователем запрос. Активируется плагин ключевым словом: если начать запрос со слова “AISEO:”, подключается модуль, который перепишет последующий текст запроса более эффективным образом [7], [8]. По сути, AISEO Prompt Enhancer вы-

ступает как надстройка над ChatGPT: пользователь набрасывает черновой запрос, а плагин трансформирует его в оптимизированный, "отполированный"запрос, предназначенный для получения наилучшего ответа от модели. Преимущества: удобство – не нужно вручную вспоминать правила хорошего запроса, достаточно в свободной форме описать, что требуется, а плагин сам отформатирует и уточнит запрос. Это снижает порог входа для новых пользователей, незнакомых с тонкостями составления вопросов к ИИ[8]. Улучшенные запросы получаются более подробными и релевантными задаче, за счёт чего повышается качество диалога с ChatGPT. Недостатки: плагин доступен только в платной версии ChatGPT с поддержкой плагинов, что ограничивает его аудиторию. Кроме того, универсальность – плагин не обучен под конкретную доменную область, он улучшает формулировку общим образом, поэтому при очень специализированных запросах результат может быть неидеальным. Также есть ограниченная прозрачность: пользователь видит только финальный изменённый запрос, но не всегда понятно, какие правки внёс сервис (может затруднить обучение пользователя самостоятельному составлению запросов). Тем не менее, AISEO Prompt Enhancer – эффективный способ "доверить AI улучшение запросов для AI что особенно полезно для быстрого создания запросов.

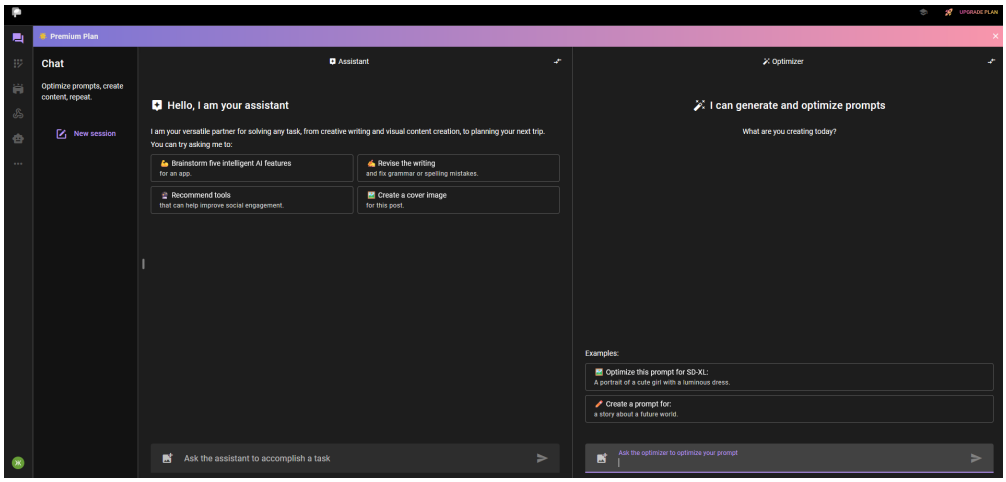

Рисунок 1.2 – Веб интерфейс PromptPerfect

3 PromptPerfect – мощный AI-инструмент для автоматической оптимизации текстовых и визуальных запросов под различные модели. Он позиционируется как "всё-в-одном"генератор и оптимизатор запросов, поддерживающий популярные LLM (GPT-3.5/4, Claude) и генеративные модели изображений (Midjourney, DALL-E и др.) [9].Пользователь вводит исходный (сырой) запрос, выбирает целевую модель – а PromptPerfect генерирует улучшенный вариант, часто более подробный и точный. Как отмечается в обзоре, этот сервис "превращает сырые английские предложения в запросы, от которых LLM краснеют то есть существенно обогащает и уточняет

их. Преимущества: PromptPerfect использует продвинутые техники инжиниринга запросов под капотом. Он может автоматически добавить недостающие детали, уточнить контекст, перевести запрос на другой язык и т.п. [9]. Отличительной чертой является поддержка мультиязычных запросов – инструмент способен принимать запросы на разных языках и выдавать оптимизированный prompt также на любом из поддерживаемых языков. Также сервис предлагает дополнительные функции: few-shot prompting (автоматически подставляет примеры в запрос), reverse `prompt engineering` для изображений (по загруженной картинке пытается сгенерировать текстовый prompt, который мог бы её описать), а также сравнение результатов разных моделей для одного и того же запроса. Наконец, у PromptPerfect есть веб-интерфейс представлен на рисунке 1.2. и API, что позволяет интегрировать его в рабочие процессы. Недостатки: отмечают не самую удобную UI[9]. – освоить все возможности может быть непросто из-за обилия настроек (крутая кривая обучения для новичков). Кроме того, хотя базовый тариф бесплатный, для активного профессионального использования может потребоваться подписка (платный план). Ещё один нюанс – автоматическая оптимизация не всегда попадает точно в потребности пользователя, иногда может "переформулировать слишком сильно изменив оттенки смысла. Тем не менее, выигрыш в качестве часто ощутим: инструмент действительно "добавляет детализации, повышая качество выходов ИИ"[9]. В тестовом обзоре PromptPerfect получил высокие оценки за функциональность (4.7/5) и производительность (4/5), при общем рейтинге 3.9/5 – снижают впечатление лишь нюансы интерфейса и поддержки. Это один из самых продвинутых на сегодня оптимизаторов запросов, позволяющий быстро улучшать запросы для разных моделей.

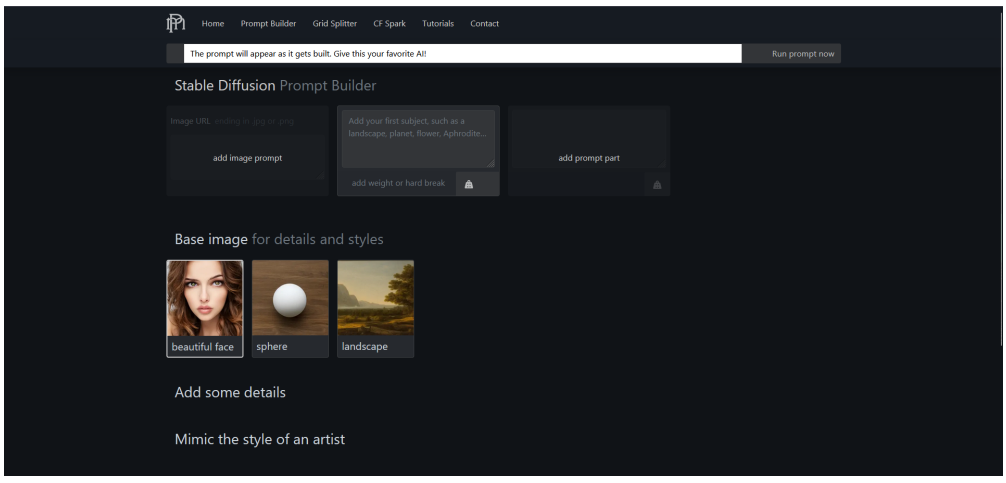

Рисунок 1.3 – Веб интерфейс PromptMania

4 Midjourney Prompt Generators – отдельная категория инструментов, помогающих генерировать описания для нейросетей, создающих изображения (в частности, Midjourney). Визуальные генеративные модели

очень чувствительны к нюансам запроса: нужный стиль, детальность описания, перечисление художественных приемов. Сформулировать такой запрос сложно, поэтому появились prompt builders – интерактивные онлайн-формы, где пользователь выбирает настройки, а на выходе получает готовую строку для Midjourney[10]. Пример – PromptoMANIA, бесплатный конструктор запросов для Midjourney и Stable Diffusion. Интерфейс предлагает многочисленные опции представленные на рисунке 1.3: тип сцены, художественный стиль (живопись маслом, фотореализм, акварель и т.д.), имя художника или референс (например, "в стиле Ван Гога"), параметры съемки (ракурс, объектив), качество и аспекты изображения. Преимущества: такие генераторы позволяют пользователю, не знакомому с терминологией и "секретными словами"Midjourney, легко подобрать эффектные описания. Они обычно имеют списки популярных ключевых слов: стили, жанры, освещение, цветовые схемы – которые можно комбинировать. Это гарантирует, что запрос будет содержать слова, понятные модели, и задействует известные стилистические паттерны. В результате картинки получаются более соответствующими замыслу. Кроме того, многие генераторы (PromptoMANIA, PromptHero и др.) обновляются под новые версии Midjourney[10],[11]. учитывая лучшие практики для последней модели. Недостатки: основной минус – ограниченная креативность, поскольку пользователь выбирает из предложенных вариантов. Не все задумки может выразить готовый интерфейс, иногда требуется вручную дописать специфические детали. Также перегруженность опциями может сбивать с толку новичков – десятки выпадающих списков с художественными стилями способны запутать. Некоторые инструменты генерируют очень длинные запросы со множеством тегов – это не всегда лучше, иногда лишние теги могут внести шум. Тем не менее, для старта в работе с Midjourney они незаменимы: снижают порог входа и учат, какие слова влияют на результат. Отдельно стоит упомянуть генераторы, которые сами используют ИИ: например, сервис HowToLeverageAI предлагает сгенерировать 10 уникальных стилей одним кликом, комбинируя случайные варианты стиля для вдохновения. Это помогает, когда хочется экспериментировать с разными художественными направлениями.

5 Stable Diffusion Prompt Generators – аналогично предыдущему, но ориентированы на модели Stable Diffusion. Поскольку SD – открытая модель, экосистема генераторов разнообразна: от простых веб-форм до открытого кода. Например, Neural Frames Stable Diffusion Prompt Generator предоставляет поле ввода для описания идеи и затем автоматически дополняет его, улучшая и уточняя для лучшего результата[12]. Заявлено, что он использует NLP-алгоритмы для подбора ключевых слов по введённому описанию – то есть пользователь пишет простое описание ("горный пейзаж на закате"), а система добавляет детали ("HDR, 4K, dramatic lighting,

by Greg Rutkowski, trending on ArtStation..."и т.п.)[13]. Преимущества: экономит время и знания – не нужно самому перебирать сотни тегов, ИИ подскажет, какие ключевые слова усилят изображение. Кроме того, такие генераторы могут обучаться на больших массивах существующих запросов. В частности, открыт репозиторий модели, дообученной на 74 тысячах текстовых описаний для Stable Diffusion[14]. Эта модель способна превратить краткий запрос пользователя в развернутый запрос, оптимизированный для SD[14]. Такие подходы, подкрепленные данными, зачастую генерируют запросы не хуже экспертов: учитываются популярные стили, корректно расставляются весы важности слов (например, скобками или символом : в SD). Недостатки: автоматические улучшители для SD могут, подобно телефонной игре, искажать исходный замысел. Пользователь рискует получить красивый, но уже не тот кадр, что он хотел, если полностью полагается на генератор. Поэтому часто советуют использовать их как подсказку: сгенерированный запрос – основа, которую можно затем вручную отредактировать. Кроме того, некоторые инструменты (как Neural Frames) работают онлайн, и качество генерации ключевых слов может варьироваться. Однако, даже частичное дополнение запроса правильными стилевыми тегами сильно повышает качество итогового изображения[13]. В целом, генераторы запросов для Stable Diffusion демократизируют использование этой сложной модели, позволяя получать красочные результаты без глубокого погружения в сообщество художников AI.

6 OctiAI Prompt Generator – относительно новая платформа (позиционируется как инструмент 2025 года), использующая комбинацию автозаполнения и ИИ для создания "идеальных запросов". OctiAI интересен своим подходом: он интерактивно задает уточняющие вопросы о вашем запросе и может автоматически заполнять ответы на них, опираясь на уже введённые данные[15]. Фактически, это полуавтоматический ассистент: пользователь описывает свой wish (желание/идею) в общем виде, далее OctiAI может сам сформулировать наводящие вопросы и сам же на них ответить, уточняя детали проекта. Сообщается, что после нескольких нажатий Enter пользователь получает персонализированный запрос, на "90% точно отражающий его задумку сгенерированный буквально за секунды[15]. Преимущества: OctiAI реализует принцип диалога при создании запроса – как опытный инженер по запросам, он расспрашивает о подробностях (а при авто-режиме сам логически додумывает их). Это экономит время: по сути, ИИ экстраполирует из начального замысла полный подробный сценарий. В примерах на сайте показано, как из краткой фразы "Хочу создать видеошаблон для рекламы моего AI-приложения в TikTok"OctiAI генерирует развернутый запрос с учетом всех особенностей платформы (динамичный темп, визуальные hooks, упоминание трендов, призыв к действию и т.д.)[15]. То есть он добавляет экспертные знания о домене (TikTok-маркетинг) пря-

мо в запрос. Для пользователя, не обладающего такими сведениями, это огромный плюс: результат – словно консультация специалиста. Недостатки: пока о нем известно не слишком много; очевидно, для такой глубокой работы OctiAI сам опирается на мощные языковые модели (возможно GPT-4), что может означать платный доступ или ограничения. Кроме того, автоматическое "додумывание"деталей за пользователя – палка о двух концах: если предположения ИИ ошибочны, итоговый запрос может увести не туда. Система заявляет 90% точности, но 10% расхождения могут быть критичными, особенно в творческих задачах. И, конечно, как и другие коммерческие сервисы, OctiAI – закрытая платформа, где не полностью ясно, как генерируется результат. Тем не менее, подход с автозаполнением и диалоговым уточнением представляет интересное направление: интерактивное соавторство с ИИ при создании запроса.

Помимо перечисленных, существуют и другие решения. Например, популярное расширение AIPRM для браузера предоставляет библиотеку готовых запрос-шаблонов для ChatGPT на все случаи жизни – от маркетинга до программирования. Оно позволяет в один клик подставить сложный многошаговый запрос с переменными, экономя время. Также сообщества вроде FlowGPT обмениваются успешными запросами. Однако эти варианты меньше про автоматическую генерацию, а скорее про шаблоны и обмен опытом. В контексте же интерактивных платформ, перечисленные 5–6 систем демонстрируют разнообразные подходы: от простых форм для структурирования (Prompt Generator), через плагины и оптимизаторы на базе ИИ (AISEO, PromptPerfect), до специализированных визуальных конструкторов (для Midjourney/SD) и гибридных ассистентов (OctiAI). Каждый инструмент имеет свои сильные стороны – либо простоту, либо интеллектуальность, либо узкую заточку под конкретную модель – а пользователю и разработчику платформы важно понимать эти особенности при выборе решения.

1.3 Архитектурные варианты платформы

При разработке платформы для интерактивного формирования запросов важен выбор архитектуры приложения. Рассмотрим основные варианты – монолитная, микросервисная, клиент-серверная – и их плюсы/-минусы применительно к нашей задаче описанные на рисунке 1.4.

Монолитная архитектура. Монолитом называют цельное приложение, где все компоненты (интерфейс, логика, обработка данных, модель ИИ) развернуты как единое целое. Такой подход традиционно проще на начальных этапах: вся логика сосредоточена в одном кодовом базе, изменения вносятся централизованно. Преимущества: высокая скорость разработки и простота деплоя – достаточно собрать и развернуть один исполняемый

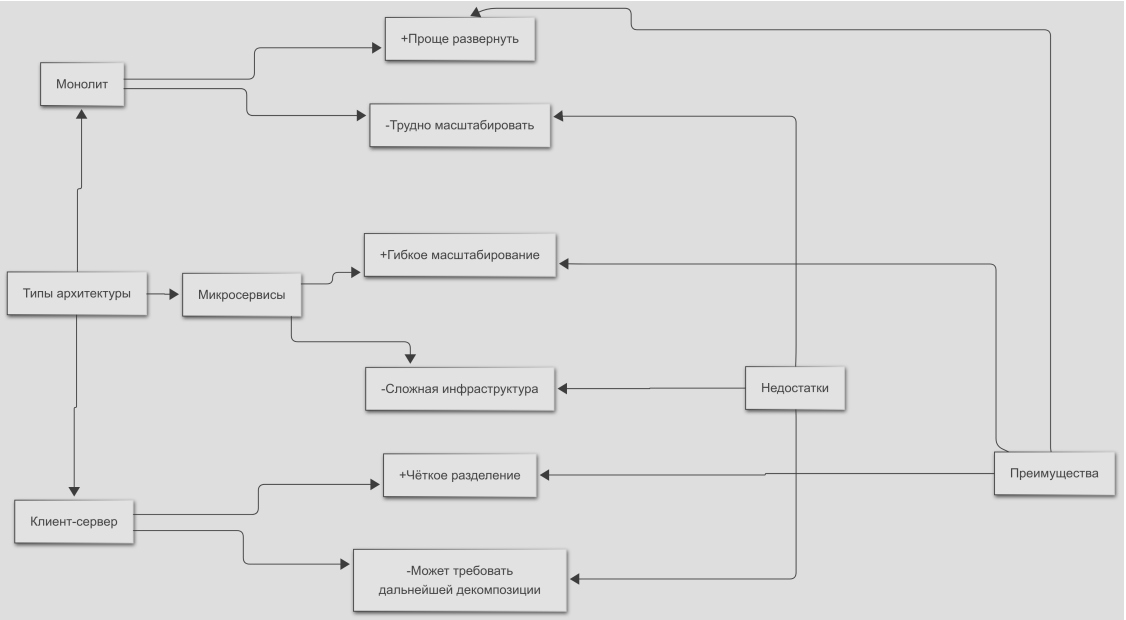

Рисунок 1.4 – Типы архитектуры: преимущества и недостатки

файл или контейнер[16]. Отладка и тестирование тоже упрощены: легче отследить выполнение запроса от интерфейса до базы, когда всё в одном процессе[16]. Производительность зачастую выше, так как компоненты обращаются друг к другу напрямую, без сетевых задержек – одна функция может вызывать другую в памяти. Недостатки: по мере роста проекта монолит может разрастись и стать трудным в поддержке. Малейшее изменение требует пересборки и перезагрузки всего приложения[16]. Масштабирование осложняется – нельзя увеличить ресурсы только для части системы, например для модуля ИИ, не масштабируя остальное. Ограничена гибкость технологий: если клиент, как и сервер, интегрирован в монолит, они жёстко связаны, и переход на другую технологию потребует переписывания значительной части монолита. Также ошибка в одном модуле способна обрушить всё приложение (низкая отказоустойчивость)[16]. В современных условиях, монолитная архитектура подходит для прототипа или небольшого проекта с ограниченной нагрузкой и командой, когда важны быстрота реализации и простота. Но если платформа будет расширяться (новые модули, большое число пользователей), монолит может стать узким местом.

Микросервисная архитектура. В микросервисном подходе приложение разбивается на ряд независимых сервисов, каждый отвечает за свою функцию (например, отдельный сервис для генерации запросов с помощью ML-модели, отдельный – для управления пользователями и базой данных, отдельный – для клиента). Эти сервисы общаются через четко определённые API. Преимущества: гибкость и масштабируемость. Каждый микросервис можно разрабатывать, развёртывать и масштабировать независимо[16]. Например, если модуль улучшения запросов на основе PyTorch-модели тре-

бует много ресурсов, его можно вынести на отдельный сервер(ы) и масштабировать горизонтально при росте нагрузки, не трогая при этом остальные части системы. Обновление одного сервиса не требует останавливать всю платформу – можно выкатить новую версию микросервиса, пока другие работают. Команды разработчиков могут параллельно трудиться над разными сервисами, выбирая оптимальные технологии под задачу (один сервис на Python, другой, скажем, на Node.js, если это оправдано)[16]. Это ускоряет выпуск новых функций и повышает надёжность: сбой одного микросервиса не обязательно выведет из строя всю систему, остальные будут функционировать. Ключевые плюсы: агильность, независимый деплой, технологическая свобода, высокая отказоустойчивость[16]. Недостатки: усложняется инфраструктура. Появляется сеть сервисов – нужно настроить их взаимодействие, сервис-дискавери, балансировку, мониторинг каждого компонента[16]. Отладка затрудняется: запрос пользователя может пройти через цепочку сервисов, и проследить его путь сложнее (нужна распределённая система логирования). Также микросервисы часто приводят к инфраструктурным излишкам: каждый сервис требует контейнер, свои ресурсы, настройки CI/CD – число артефактов растёт[16]. Неконтролируемый рост сервисов (development sprawl) может замедлить разработку, если не вводить стандарты и не управлять зависимостями[16]. Для небольшой команды микросервисы могут оказаться чрезмерно сложными. Таким образом, имеет смысл переходить к микросервисной архитектуре, когда проект дорастает до масштабов, требующих отдельного масштабирования компонентов или работы больших команд параллельно.

Клиент-серверная архитектура. Фактически, это классический подход для веб-приложений, предполагающий разделение системы на клиент и сервер. Клиент и сервер общаются по сети (HTTP/REST или WebSocket). По сути, любое современное веб-приложение уже клиент-серверное, но важно подчеркнуть разделение ролей. Преимущества: разделение ответственности (separation of concerns) – UI и бизнес-логика отделены. Это упрощает поддержку, а также это даёт гибкость разработки: разработчики клиента и сервера могут работать независимо, используя специализированные инструменты для своих частей. Централизация управления на сервере упрощает обновление логики – код модели и оптимизации запроса хранится на сервере, контроль доступа, безопасность – тоже там, а клиент лишь отображает результаты. Как отмечается в литературе, клиент-сервер обеспечивает лучшую управляемость и безопасность за счёт концентрации ресурсов и данных на сервере[17],[18].

Дополнительно, можно рассмотреть и другие варианты архитектур. Например, Serverless/FaaS – когда отдельные функции (например, улучшение запроса) развёрнуты как безсерверные функции AWS Lambda или аналогов. Это обеспечивает автомасштабирование и оплату "по факту ис-

пользования". Однако интеграция долгоживущих моделей в lambdas затруднительна из-за ограничений по времени выполнения и памяти. Многослойная архитектура (трёхуровневая) – где есть ещё слой базы данных, кэширующие прослойки – может применяться при большом числе пользователей и необходимости хранения истории запросов, рейтингов и т.д. Но на концептуальном уровне она схожа с клиент-сервером, просто добавляются инфраструктурные детали.

Резюмируя: для старта платформы интерактивной генерации запросов достаточно архитектуры "клиент сервер". Это разделит front/back, обеспечит удобную разработку и деплой. По мере роста нагрузки или функциональности можно выделять микросервисы – например, вынести генерацию улучшенных запросов в отдельный сервис, который можно масштабировать независимо (например, несколько инстансов с моделью на PyTorch за балансировщиком). Монолитный же вариант уместен только для простейшего прототипа или офлайн-приложения. В современном веб-окружении разделение на клиент и сервер – де-факто стандарт, обеспечивающий и модульность, и централизованное управление данными[18].

1.4 Модели и подходы для генерации улучшенных запросов

Последний аспект – как именно реализовать автоматическое улучшение запросов средствами NLP/ML в рамках выбранного технологического стека. Существуют разные подходы один из которых представлен на рисунке 1.5: от простых правил и шаблонов до обучения нейросетей, которые переписывают запросы пользователя в оптимизированной форме. Рассмотрим несколько вариантов:

1 Правила и лингвистические методы (NLP без глубокого обучения). Используя NLTK, можно выполнить базовый лингвистический анализ запроса: токенизация, определение частей речи, разбор предложения. Это пригодится для реализации простых эвристик улучшения. Например, можно проверить длину запроса – если он слишком короткий или состоит из одного-двух слов, платформа может подсказать пользователю добавить деталей (либо автоматически сделать запрос уточняющим вопросом). NLTK также предоставляет тезаурусы (например, WordNet) – их можно использовать, чтобы обогащать текст синонимами или более точными терминами. Предположим, пользователь ввёл: "расскажи про кошек". Автоматически можно заменить расплывчатое "расскажи"на более конкретное "дай обзор повадок домашних кошек, включая."и т.д. – часть данных (например, список аспектов ухода за кошками) можно хранить как шаблоны (в Pandas DataFrame или БД) и подставлять по теме. Такие улучшения можно делать на основе ключевых слов: распознав слово "кошки подставить часто

Рисунок 1.5 – Алгоритм улучшения запроса

запрашиваемые связанные детали (питание, здоровье, породы). Это похоже на экспертную систему: по теме запроса добавляются уточняющие фразы. Плюсы: простота реализации, полная контроль и интерпретируемость – мы точно знаем, что меняем в запросе. Минусы: сложно масштабируется на все темы – нужен словарь или база знаний, иначе ограничится несколькими предметными областями. Кроме того, без понимания контекста можно ошибиться – синоним может не подойти точно по смыслу. Однако для базовых улучшений (например, добавление в конец просьбы "дай ответ списком шагов"или "форматируй ответ в JSON если пользователь отметил нужный формат) – правила подходят отлично. Мы можем реализовать набор опций: пользователь на клиенте флажками выбирает "Detailed response "Bulleted format "Include examples"– а на сервере простой код на Python/NLTK добавляет соответствующие строки к запросу. Такая система не обучается, но повышает качество запросов по заранее заданным шаблонам.

2 Использование предобученных языковых моделей (LLM) для перефразирования. Более продвинутый путь – привлечь саму мощь LLM для улучшения запросов. Например, интегрировать вызов OpenAI GPT-4 API: передав в него что-то вроде "Преврати следующий запрос пользователя в максимально подробный и ясный запрос для ИИ: ’<пользовательский запрос>’". GPT-4 способен сам сгенерировать улучшенный вариант, опи-

раясь на свой опыт. В этом подходе мы как бы вкладываем один слой ИИ внутрь нашего приложения – мета-ИИ, который помогает обращаться к ИИ. Кстати, именно так работают плагины вроде AISEO и PromptPerfect, только используя API-ключи и, возможно, собственные модели. Плюсы: качество – современные LLM близки к уровню человека в составлении инструкций[7]. Исследования показывают, что большие модели способны быть инженерами запросов на уровне человека[19]. Метод, известный как Automatic Prompt Engineer (APE), поручает модели генерировать несколько вариантов инструкций и тестировать их, выбирая лучший[7]. Это можно реализовать и онлайн: например, сгенерировать 3 версии запроса, прогнать через целевую модель (или оценить вероятности токенов) и выбрать наиболее результативный. GPT-4, конечно, ресурсозатратен, но дает отличные перефразирования. Минусы: зависимость от внешнего API (если используем его) или высокая нагрузка на сервер (если разворачивать свой LLM). Также может быть непредсказуемость – нужно тщательно настроить системные подсказки, чтобы модель-улучшатель не исказила смысл. Но в контролируемых рамках (с фиксированным шаблоном для улучшения) это работает.

3 Классические ML-алгоритмы. Можно подойти и с точки зрения обучения более простых моделей: например, классификатор, определяющий, каких элементов не хватает в запросе. Используя Pandas, можно проанализировать множество запросов (например, от пользователей платформы), отметить случаи, где не указана роль, или не задан формат ответа. Затем обучить модель (даже решающее дерево или логистическую регрессию) предсказывать, что нужно добавить. Однако, такие решения уступают по гибкости нейросетям. Поэтому чаще либо правила, либо уже end-to-end генерация текстов.

4 Специализированные алгоритмы оптимизации запросов. В научных публикациях появляются методы, позволяющие улучшать запросы автоматически. Помимо упомянутого APE (поиска лучшего инструктажа), есть подход Meta-Prompting – когда модель обучается сама придумывать эффективные инструкции без внешних данных.Также существуют методы с участием пользователя: интерактивные, когда система генерирует несколько вариантов перефразированного запроса, и пользователь выбирает лучший (это своего рода обучение с подкреплением от пользователя). Можно реализовать интерфейс, где после автоматического улучшения запроса показывается пользователю: "Вот так лучше? [Да/Нет] и если нет – дать другой вариант. Получая обратную связь, система могла бы обучаться (например, методом reinforcement learning, настраивая веса модели-генератора). Это более сложная схема, но она приближает работу платформы к совместному творчеству с пользователем, а не просто "чёрному ящику".

Важно также учесть время отклика. Улучшение запроса должно происходить быстро (идеально $<$ 1-2 секунд), чтобы интерфейс оставался отзывчивым. Правила NLP выполняются миллисекунды, а вот вызов большой модели – может занять секунды. Поэтому, возможно, нужно кешировать результаты или использовать облегченные модели. Один из компромиссов – делать улучшение по требованию: например, если пользователь явно нажал кнопку "Enhance". Если же пользователь уже изначально дал хороший подробный запрос, система может и не тратить время (определив метрикой, что запрос достаточно длинный или содержит все элементы). Здесь Pandas пригодится для хранения статистики: например, средняя длина запросов, какие улучшения чаще всего применяются – это поможет тонко настроить, когда включать ту или иную модель.

Подытоживая, возможные модели для улучшения запросов лежат на спектре от полностью ручных (статические шаблоны, правила) до полностью автоматических (нейросети, генерирующие идеальный запрос). На практике часто эффективна гибридная система: базовые вещи (формат, вежливость, явные ошибки) чинит простая логика, а тонкие стилистические или содержательные улучшения доверяются нейросети. Такой подход даст и качество, и предсказуемость.

1.5 Вывод

Проведённый анализ показал, что качество работы с LLM и генеративными моделями напрямую зависит от того, насколько структурирован, конкретен и контекстуально насыщен исходный запрос. Оптимальный «хороший» промпт складывается из пяти взаимодополняющих блоков — директивы, примеров, роли / персоны, указания формата вывода и дополнительного контекста — и выигрывает, когда эти части чётко разделены визуальными или семантическими делимитерами. Для многошаговых задач особенно полезны техники Chain-of-Thought и self-consistency, позволяющие вынести рассуждение модели «на поверхность» и тем самым повысить точность результата.

Обзор рынка подтвердил востребованность инструментов, которые переводят эти принципы в «кнопочный» интерфейс: от шаблонных конструкторов (CopilotWorks) до плагинов-оптимизаторов (AISEO) и многофункциональных сервисов (PromptPerfect, OctiAI). Их сильные стороны— быстрое обучение пользователя и гарантированная базовая структура запроса; слабые — шаблонность, скрытая логика правок и часто платная модель доступа. Для визуальных моделей (Midjourney, Stable Diffusion) рынок предлагает специализированные генераторы, агрегирующие стилистические теги и тем самым снижающие порог входа для художников.

С точки зрения архитектуры наиболее прагматичным стартовым

вариантом остаётся классическая клиент-серверная модель: UI на SPA-фреймворке и бекенд-API, где сосредоточены бизнес-логика и ML-функции. Монолит удобен для прототипа, но ограничивает масштабирование; микросервисы же обоснованы только после появления ресурсно-ёмких или быстро эволюционирующих компонентов (например, сервиса генерации улучшенных промптов на GPU). Важное операционное требование — сохранять время отклика «подсказка → обратная связь» в пределах 1–2 с; это диктует гибридный подтекст: лёгкие эвристики и кеширование результатов правилами, а глубокое перефразирование — по кнопке или асинхронно.

Наконец, оценены варианты автоматического улучшения запросов. Для бытовых сценариев достаточно композиции «правила + тонкие шаблоны»; для профессионального качества потребуется LLM-«мета-помощник», обученный на корпусе высокоранговых промптов и умевший динамически подстраивать уровень детализации. На практике наилучший результат даёт гибридная стратегия: простая логика исправляет формат и ошибки, а нейросеть дополняет содержание и стиль.

Таким образом, при проектировании платформы следует сочетать:

– проверенные приёмы структурирования и детализации запроса;

– UI-механику, которая «заставляет» пользователя пройти через эти приёмы, не перегружая его деталями;

– модуль улучшения промпта, реализованный как гибрид правил и LLM;

– масштабируемую клиент-серверную архитектуру с возможностью вынести тяжёлые сервисы в отдельные узлы по мере роста нагрузки.

2 ПРОЕКТИРОВАНИЕ МОДЕЛИ ПЛАТФОРМЫ ДЛЯ ИНТЕРАКТИВНОГО ФОРМИРОВАНИЯ ЗАПРОСОВ К ЯЗЫКОВЫМ И ГЕНЕРАТИВНЫМ НЕЙРОСЕТЯМ

2.1 Целевая аудитория

Целевой аудиторией системы являются пользователи-разработчики и художники, которые будут работать с ней на равных правах (единый уровень доступа). Платформа спроектирована с учётом их потребностей: с одной стороны, она предоставляет гибкие инструменты для тонкой ручной настройки запросов, с другой – наглядный и интуитивно понятный интерфейс, упрощающий работу с текстовым описанием. В следующих разделах будут подробно рассмотрены назначение системы, характеристики пользователей, требования, обоснованные техническими требованиями.

2.2 Назначение системы

Основное назначение разработанной системы – обеспечить пользователям возможность эффективно формировать и улучшать запросы (промпты) для различных нейросетевых моделей генерации текста и изображений. Платформа предназначена для поддержки полного цикла работы с промптом: от начального наброска идеи до получения удовлетворительного результата. Она предоставляет единое интерактивное пространство для проектирования промптов, позволяя объединить несколько функций, которые ранее требовали разрозненных инструментов.

С помощью данной системы пользователи могут создавать и редактировать текстовые описания запросов, итеративно их уточняя и детализируя. Интеллектуальные функции платформы позволяют автоматически дополнять введённый текст недостающими деталями и контекстом, что помогает менее опытным пользователям (например, художникам, не обладающим глубокими навыками работы с языковыми моделями) получить более богатые описания. Встроенный модуль оценки качества анализирует сформулированный запрос и выдает количественный показатель (в диапазоне 0–100), отражающий ориентировочное качество промпта. Это даёт возможность сразу увидеть, насколько хорошо запрос может быть воспринят нейросетью, и при необходимости внести правки до генерации результата. Кроме того, система обеспечивает преобразование формата запроса под требования конкретной целевой модели: например, адаптирует описание под синтаксис, предпочтительный для модели генерации изображений или,

наоборот, для диалоговой языковой модели. Наконец, реализован режим предварительного просмотра, позволяющий по нажатию кнопки отправить текущий промпт в тестовом режиме модели и получить пример отклика – сгенерированный фрагмент текста либо эскиз изображения.

Таким образом, платформа выполняет роль песочницы для промптов: разработчики могут отладить запросы для своих AI-модулей, а художники – подобрать оптимальные описания для генерации иллюстраций. За счёт возможности быстро просматривать результат и оценку, существенно сокращается число итераций "вслепую". Пользователи могут корректировать промпт до тех пор, пока не будут удовлетворены оценкой и предпросмотром, и лишь затем использовать его на основной целевой модели (например, в продакшн-системе либо в стороннем сервисе генерации). В итоге применение платформы позволяет повысить качество конечного генерируемого контента – будь то текстовые ответы модели или создаваемые изображения – и сделать процесс разработки промптов более эффективным и предсказуемым.

2.3 Характеристика пользователей

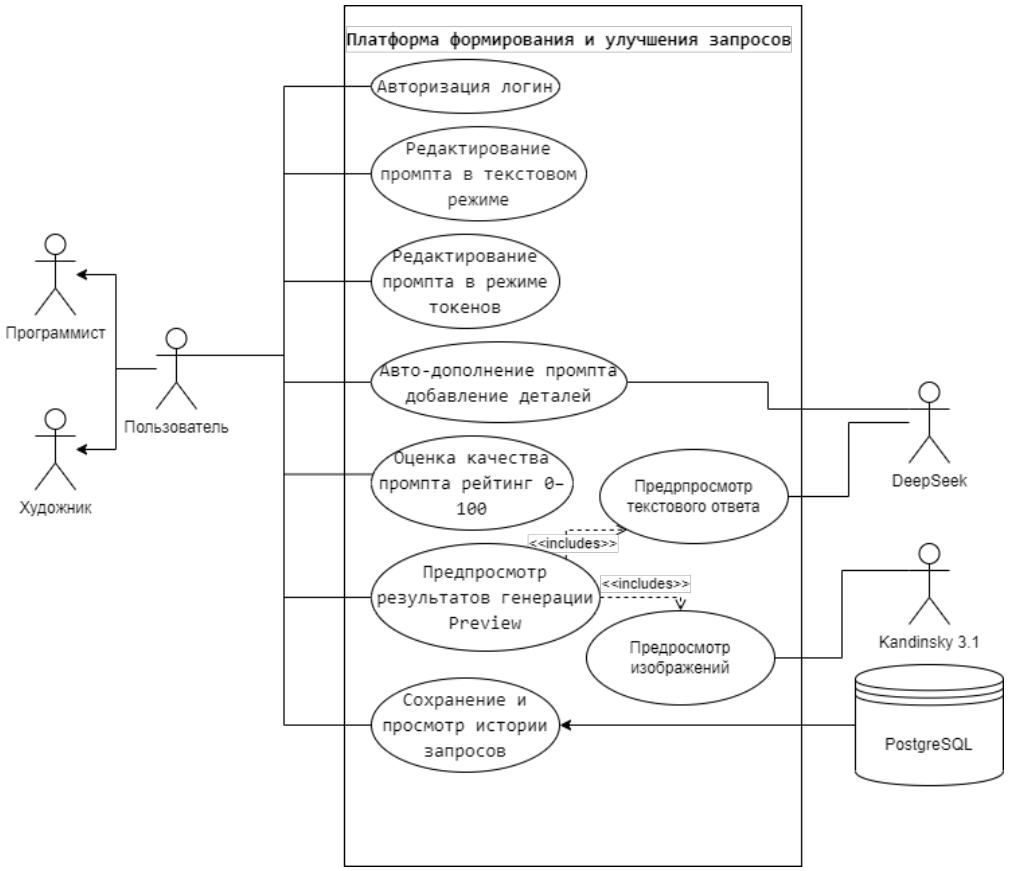

Рисунок 2.1 – Вариантов использования платформы

Как отмечалось, предполагаемые пользователи системы – это раз-

работчики и художники, работающие с генеративными нейросетями. При проектировании платформы исходили из того, что эти две категории пользователей обладают разным опытом и целями, но обе нуждаются в удобном инструменте для создания хороших промптов. Уровень доступа в системе для них одинаковый (роль обычного пользователя), то есть все функции платформы доступны как разработчикам, так и представителям творческих профессий без каких-либо ограничений, что можно заметить на диаграмме 2.1.

Разработчики (инженеры, специалисты по машинному обучению, авторы чат-ботов и др.) используют платформу преимущественно для отладки и улучшения текстовых запросов к языковым моделям. Для них важны гибкость и точность: они оценивают промпт с точки зрения логики обработки моделью, предсказывают, приведёт ли данная формулировка к нужному ответу. Разработчики, как правило, технически подкованы, поэтому им удобен режим прямого текстового редактирования промпта, возможность вручную корректировать даже мелкие детали формулировки. Они оценят также числовой показатель качества – метрику, которая поможет количественно сравнить разные варианты запросов. В их рабочем процессе платформа вписывается как инструмент быстрого прототипирования промптов: вместо того чтобы многократно вызывать целевую модель и получать от неё ответы для сравнения, разработчик может в интерактивном режиме довести один запрос до оптимального состояния, глядя на предварительные ответы от тестовой модели и на рейтинг качества. Получив удовлетворяющий результат, он переносит этот промпт в своё основное приложение или сервис.

Художники и специалисты по визуальному контенту (например, иллюстраторы, дизайнеры, концепт-художники) используют систему в несколько ином ключе – для создания описаний, на основе которых модели типа Stable Diffusion или Kandinsky генерируют изображения. Их цель – добиться, чтобы сформулированный текст точно передавал задуманный визуальный образ и стиль. Многие художники не имеют глубоких технических знаний, поэтому для них критична простота и наглядность интерфейса. Платформа учитывает это, предоставляя интуитивно понятные средства: режим редактирования через перетаскивание токенов особенно полезен, так как позволяет им воспринимать промпт не как сплошной текст, а как набор отдельных элементов или слов, из которых складывается описание сцены. Перемещая эти элементы, художник может экспериментировать с композицией фразы, не задумываясь о синтаксисе – система автоматически обновит текст запроса. Функция автоматического дополнения детализацией полезна для художника тем, что нейросеть сама предложит дополнительные детали (например, обстановку, освещение, стилистические прилагательные), которые могут улучшить итоговое изображение.

Предпросмотр через API Fusion Brain позволяет им практически мгновенно увидеть черновой результат: например, эскиз сгенерированного изображения по текущему описанию. Это визуальная обратная связь чрезвычайно ценна – художник сразу поймёт, в правильном ли направлении двигается, и при необходимости скорректирует промпт (добавит или уберёт детали, изменит формулировку).

Важно подчеркнуть, что система не вводит разграничений между разработчиками и художниками на уровне функционала. И те, и другие работают в одной среде с идентичными возможностями. Это решение принято исходя из многопрофильности современных команд: нередко разработчики и дизайнеры работают совместно над проектами с AI, и им удобно использовать единый инструмент. Платформа удовлетворяет потребности обоих типов пользователей: обеспечивает достаточно тонкий контроль для удовлетворения запросов разработчиков и одновременно остается дружественной для творческих специалистов, не требуя от них знаний программирования. Такой подход расширяет потенциальную аудиторию системы и повышает её ценность в междисциплинарных командах.

2.4 Технические требования

Разработка платформы базируется на предварительно сформулированных технических требованиях, отражающих необходимый функционал и ограничения системы. Требования охватывают как функциональные возможности, видимые пользователю, так и нефункциональные характеристики (производительность, совместимость, безопасность и др.). Ниже перечислены ключевые требования к системе.

Функциональные требования (основной пользовательский функционал платформы):

1 Редактирование запросов в текстовом и визуальном режиме. Система должна предоставлять интерфейс для ввода и правки текста промпта вручную (классическое текстовое поле) и альтернативный интерфейс в виде списка токенов, которые пользователь может перемещать, удалять или заменять. Любое изменение последовательности токенов в графическом режиме должно незамедлительно отражаться в текстовом представлении промпта, и наоборот, обеспечивая двунаправленную синхронизацию. Должна поддерживаться работа с промптом произвольной длины (в разумных пределах, например до 1000 символов) и с различными символами, включая буквы разных алфавитов, цифры, знаки препинания и эмодзи (это важно, так как некоторые модели допускают использование эмодзи в описании изображения)[20]. Пользователь должен видеть текст своего запроса и иметь возможность редактировать его удобным для себя способом.

2 Автоматизированное дополнение (расширение) запроса деталями.

Платформа должна по запросу пользователя уметь генерировать расширенную версию введённого промпта, добавляя к нему недостающую детализацию. Фактически, это интеллектуальный помощник: на основе исходного чернового текста запроса система (с помощью внутренней языковой модели) предлагает дополнительные описательные фразы, контекст или уточнения, которые могут сделать запрос более понятным для нейросети. Например, пользователь ввёл краткий запрос "кот сидит на дереве а система может предложить дополнить: "кот сидит на дереве в лучах заходящего солнца, вокруг осенний пейзаж". Метод вызова – нажатие специальной кнопки ("Дополнить"), после чего текущий текст промпта вместе с уже введёнными деталями остается, а к нему в конец или по соответствующим местам добавляется сгенерированный моделью текст (пользователь затем может отредактировать результат по своему усмотрению). Требуется, чтобы дополнение работало корректно для разных типов запросов (как описательных, так и вопросительных) и на двух основных языках (русский и английский), учитывая, что целевые модели поддерживают многоязычные запросы.

3 Оценка качества промпта. В системе реализуется автоматическая оценка составленного пользователем запроса по шкале от 0 до 100. Эта оценка носит рекомендательный характер и должна отражать степень соответствия промпта лучшим практикам и ожидаемым требованиям модели. Пользователю отображается числовое значение или графический индикатор (например, цветовая индикация: красный – низкое качество, зелёный – высокое). Внутренне оценка может основываться на наборе правил или модели, анализирующих текст: учитывается длина запроса, специфичность формулировок, наличие деталей, отсутствие противоречий или запретов. Например, слишком короткий или расплывчатый запрос получит низкую оценку, а конкретный, содержательный запрос – более высокую. Чем более точно и понятно сформулирован промпт, тем выше должен быть рейтинг. (Это соответствует рекомендациям по промптингу: качество результатов зависит от того, сколько информации предоставлено и насколько хорошо запрос составлен)[21]. Данный модуль служит подсказкой: он должен быстро (в течение долей секунды) пересчитывать оценку при изменении текста и тем самым стимулировать пользователя улучшать запрос до получения приемлемого балла.

4 Предварительный просмотр результата. Одним из важнейших требований является возможность быстрой генерации чернового результата по текущему запросу. Пользователь, не покидая интерфейс редактора промптов, должен получить от системы пример отклика целевой модели – сгенерированный текст или изображение, соответствующее введённому описанию. Данный функционал реализуется двумя способами: для текстовых моделей – через API DeepSeek, для изображений – через вызов внешнего API

Fusion Brain. Платформа должна определить, какой тип результата требуется (например, по выбранному целевому движку или по контексту: если пользователь редактирует текстовый запрос для чат-бота, то генерируется текст-ответ; если промпт адресован модели изображения, то генерируется картинка). После нажатия кнопки "Предпросмотр"происходит обращение к соответствующей модели:

4.1 В случае текста: DeepSeek генерирует продолжение или ответ на заданный промпт (необходимо ограничить размер выдаваемого фрагмента, например 1000 символов, чтобы получить быстрый предварительный ответ). Полученный ответ отображается на экране (например, в отдельном поле под запросом).

4.2 В случае изображения: сервер отправляет запрос к API Fusion Brain (модель Kandinsky 3.1) с параметрами генерации (тип = GENERATE, текст запроса и др.). В ответ через некоторое время возвращается сгенерированное изображение, которое выводится пользователю (в виде эскиза, уменьшенного изображения либо полноразмерного, если позволяет интерфейс). Пользователь должен иметь возможность быстро увидеть этот результат в интерфейсе, без ручного перехода на внешние сервисы. Например, платформа может отобразить сгенерированную картинку прямо в браузере. В обоих случаях предварительно сгенерированный контент носит ознакомительный характер – он позволяет оценить, что примерно получится из данного промпта. Пользователь, проанализировав результат, может тут же подправить запрос и снова вызвать предпросмотр, добиваясь улучшения. Важно, чтобы среднее время получения предпросмотра было приемлемым: для текста это обычно менее 5 секунд, для изображения – порядка 5–15 секунд (зависит от мощности сервера и скорости внешнего API).

Помимо основных функций, вытекают и общие технические требования к системе:

1 Интуитивность и удобство интерфейса. Пользователи разных категорий должны легко освоить работу с платформой. Интерфейс должен быть локализован (минимум на русском языке, так как аудитория – в том числе русскоговорящие художники). Все элементы (кнопки "Оценить "Предпросмотр переключатель режима редактирования и т.д.) должны быть наглядно обозначены. Требуется реализация динамического обновления – например, пересчёт оценки качества без перезагрузки страницы, мгновенная синхронизация текстового и токенного представления запроса. Это подразумевает использование возможностей SPA (Vue.js) для реактивности.

2 Производительность и масштабируемость. Платформа должна эффективно работать при одновременном использовании несколькими пользователями. Ожидается, что число активных пользователей невелико (на-

пример, сотни или тысячи, поскольку целевая аудитория – команда разработчиков или художников или отдельные специалисты), однако архитектура не должна иметь жёстких ограничений на масштабирование. Серверная часть на FastAPI должна обрабатывать параллельно несколько запросов (благодаря асинхронности и производительности FastAPI это достижимо)[22]. Важно оптимизировать время отклика: интерактивные операции (редактирование, оценка) происходят почти мгновенно, а более тяжёлые (предпросмотр) – максимально быстро для выбранных моделей. Для Fusion Brain API и DeepSeek API можно предусмотреть обработку в асинхронном режиме (не блокируя основной поток приложения во время ожидания ответа от внешнего сервиса).

3 Ограничения и обработка ошибок. Система должна корректно обрабатывать нестандартные ситуации: слишком длинные запросы (превышающие лимиты модели) – выдавая предупреждение или автоматически укорачивая до допустимого размера; недопустимые символы – экранируя или удаляя их; отсутствие связи с внешним API – уведомляя пользователя о невозможности получить предпросмотр в данный момент. Все входные данные валидируются (FastAPI и Pydantic обеспечивают автоматическую проверку типов и форматов на уровне API[23]), а при обнаружении некорректных – сервер возвращает понятные сообщения об ошибках. Важным требованием является устойчивость: сбой одной из компонентов (например, недоступность БД или падение модели) не должен приводить к неуправляемому краху всего приложения – должны быть предусмотрены механизмы возврата в консистентное состояние или перезапуска сервиса.

4 Совместимость и развиваемость. Платформа должна быть построена с использованием стандартных веб-технологий, обеспечивающих её совместимость с основными браузерами (Chrome, Firefox, Safari, Edge) без необходимости установки специальных плагинов. Желательно соблюдение принципов адаптивности интерфейса (возможность работать на экране ноутбука, настольного ПК, и, по возможности, на планшете). В архитектуре и коде должны быть заложены возможности расширения: например, добавление новых моделей (если в будущем понадобится поддержать другую нейросеть для генерации аудио или видео, или добавить ещё одну языковую модель) не должно требовать переписывания системы с нуля. Это означает, что компоненты (модуль оценки, модуль реструктуризации) должны быть реализованы конфигурируемо, с возможностью подключения новых правил или алгоритмов.

Сформулированные выше требования служат основой для проектирования системы. На их основании в следующих разделах выбираются соответствующие методики и технические решения, обосновывается архитектура, алгоритмы работы и необходимые ресурсы. Соблюдение этих требований должно обеспечить, что итоговая платформа будет полнофункциональной, удоб-

ной для целевых пользователей и надёжной в эксплуатации.

2.5 Требования к алгоритмическому и программному обеспечению

В данном разделе описаны предполагаемые ключевые алгоритмы, реализующие функции платформы, а также программное обеспечение (библиотеки, фреймворки) и принятые технические решения, которые будут влиять на разработку приложения. Также рассматриваются программные средства, обеспечивающие выполнение этих алгоритмов.

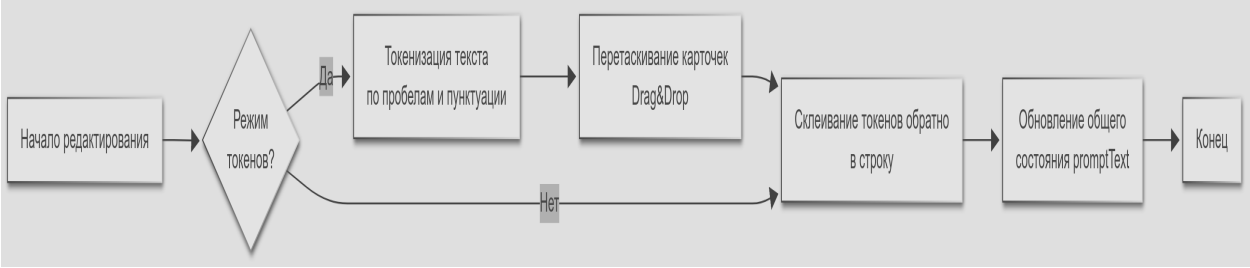

Рисунок 2.2 – Алгоритм редактирования запроса

Редактирование запросов (текст и токены). Алгоритм редактирования в текстовом режиме тривиален и отображён на диаграмме 2.2: пользователь вводит или изменяет строку символов, которая хранится во внутреннем состоянии приложения. Более интересен алгоритм представления промпта в виде набора токенов. Здесь под токеном понимается либо отдельное слово, либо устойчивое словосочетание – на практике для упрощения можно принять токенизацию по пробелам и знакам пунктуации (каждое слово или знак рассматривается как отдельный элемент). При переключении в режим токенов исходный текст промпта разбивается на массив токенов. Этот массив отображается на экране, например, в виде последовательности интерактивных "карточек"с текстом. Для реализации перетаскивания используются возможности HTML5 DnD или готовые компоненты из экосистемы Vue (например, Vue.Draggable). Далее массив конкатенируется обратно в строку с добавлением пробелов – таким образом получается обновлённый текст промпта. Vue.js автоматически отследит изменение массива токенов и обновит связанную переменную текста (или наоборот, если связность сделана через одно хранилище состояния). Таким образом обеспечивается одновременная актуализация обоих представлений. Кроме того, если пользователь в текстовом поле ручным образом изменяет текст (например, дописывает новое слово), алгоритм распознаёт это (через реактивный watcher) и пересобирает список токенов заново. В результате редактирование может происходить параллельно в любом формате без рассинхронизации. Этот алгоритм достаточно прост (основные операции – разбиение

строки и объединение списка), поэтому не нагружает систему даже при длинных промптах. Дополнение запроса детализацией. Данная функция

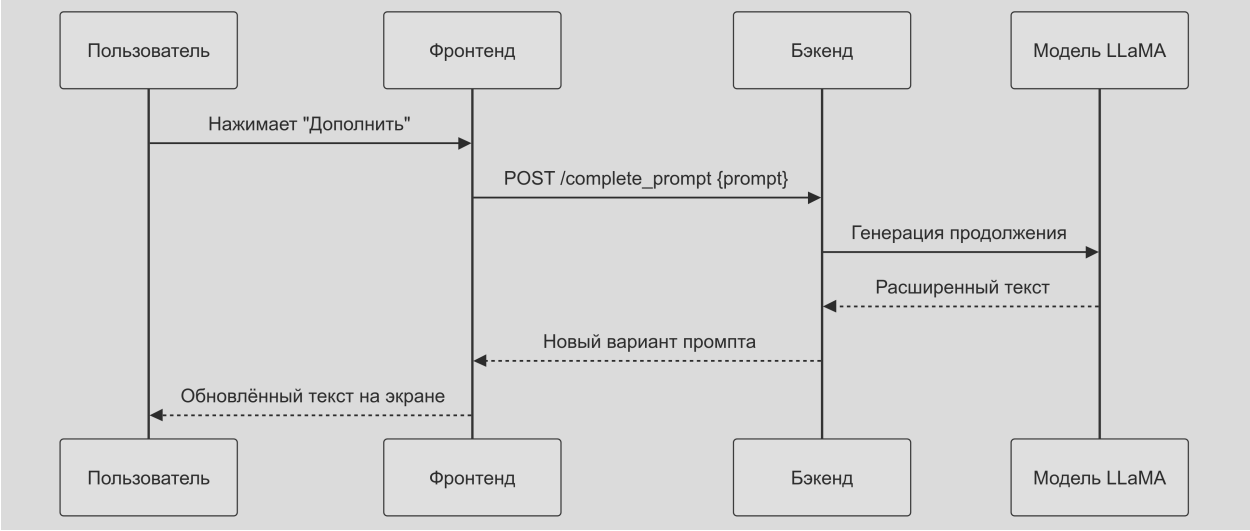

Рисунок 2.3 – Алгоритм дополнения запроса

опирается на алгоритмы генерации текста с помощью модели DeepSeek. При нажатии пользователем кнопки "Дополнить"клиент отправляет текущий текст промпта на сервер (эндпойнт verb|/complete prompt|). Сервер вызывает специальную функцию, которая обращается к API модели DeepSeek. Алгоритм представлен на диаграмме 2.3 состоит из следующих шагов:

– клиент отправляет исходный текст на сервер;

– сервер формирует входную подсказку (Prompt) для DeepSeek, используя шаблон «Расширь описание . . . >;

– модель генерирует продолжение с учётом заданных параметров (длина, температура и пр.);

– полученные фразы автоматически добавляются к исходному промпту.

Алгоритмически, основная интеллектуальная работа здесь выполняется языковой моделью. Предусмотрены некоторые меры для улучшения результата: например, если модель начала повторять исходный текст или уводит в сторону, сервер может отсечь первые токены генерации до появления новых данных. Поскольку DeepSeek предоставляется готовым REST API, он интегрирована через Rest API. Таким образом, алгоритм дополнения промпта сводится к задаче последовательного генерирования текста нейросетью – типичная задача, для которой используются наработки и функции библиотек машинного обучения.

Оценка качества промпта. Алгоритм оценки – это комбинация эвристических правил и, потенциально, моделей машинного обучения. В рамках данного проекта реализуется более простой, объяснимый подход на основе метрик текста. При вызове функции оценки (пользователь нажимает,

либо автоматически при изменениях) клиент отправляет промпт на сервер (`/evaluate\ prompt`). Сервер вызывает функцию evaluate prompt(text), которая выполняет следующие шаги:

1 Анализ длины и полноты. Вычисляется длина текста (в словах или символах). Если длина меньше некоторого порога (например, менее 3–5 слов), выставляется низкий базовый балл (такие запросы, как правило, слишком общие). Оптимальный диапазон длины – эмпирически, скажем, 10–20 слов для текстовых моделей, 5–15 слов для графических (со стилевыми тегами). Если запрос очень длинный (более 50–100 слов), тоже может снижаться балл – за избыточность.

2 Проверка специфичности. Алгоритм может содержать список "пустых"слов (типа "изобрази "сделай"– паразитные для промпта генерации изображения) и, наоборот, проверить наличие содержательных прилагательных, уточняющих фраз. Например, наличие хотя бы одного прилагательного или определяющего оборота может добавлять балл, так как уточняет запрос. В руководствах по промптингу подчёркивается, что специфичность и контекст повышают качество результата[21]. Следовательно, если промпт содержит детали (например, указание стиля, времени суток, эмоций персонажа и т.п.), это положительно влияет на оценку.

3 Ясность формулировки. Проверяется, нет ли в тексте двусмысленностей или нерелевантных фрагментов. Этот пункт сложнее формализовать, но можно частично оценить по структуре предложения: если промпт состоит из нескольких несвязанных предложений или вопросительных форм (которые могут сбить модель), балл снижается. Например, запрос вида: "Нарисуй кота. Может дерево? Нет, лучше солнце."явно плох по структуре – алгоритм может обнаружить наличие нескольких предложений или вопросительных знаков и снизить оценку.

4 Учет модели. Если уже выбрана целевая модель, алгоритм может использовать разные профили оценки. Например, для изображения ценятся указания на визуальный стиль (реалистичный, мультяшный, 3D-рендер и т.д.) – их наличие повышает балл. Для текста ценится чётко поставленный вопрос или задача – наличие вопросительного знака при обращении к чат-боту, наоборот, может быть положительным (для диалоговой модели).

5 Формирование итогового балла. Балл вычисляется на основании нескольких компонент. Например: score = length score + specificity score + clarity score + model adaptation score. Каждая в диапазоне 0–25, суммируя до 100. Данный подход позволяет объяснить оценку: "Ваш промпт слишком короткий, добавьте деталей"– если низкий length score, "промпт содержит противоречивые указания"– если clarity score низкий, и т.п.

6 Возврат балла. Сервер возвращает число от 0 до 100. Клиент отображает его, при этом возможно сопроводить коротким вердиктом (например, "Среднее качество"при 50–70, "Отличный промпт"при >80).

Этот алгоритм легко модифицировать под новые критерии, его можно обучить (если бы были данные промптов с оценками, можно настроить веса правил или модель классификатор). Пока же он реализован на основе здравого смысла и рекомендаций по написанию промптов[24], [21]. Программно реализация выполнена на Python: для лингвистического анализа можно использовать стандартные библиотеки (например, NLTK или simple postagging для выделения частей речи, если нужно искать прилагательные). Однако в простейшем варианте достаточно операций над строками, что не требует внешних зависимостей. Важно отметить, что время выполнения этого алгоритма невелико – порядка нескольких миллисекунд, т.к. текст анализируется без тяжёлых моделей. Поэтому оценка может обновляться практически мгновенно при каждом изменении запроса.

Реструктуризация промпта под модель. Алгоритм реструктуризации заключается в применении определённого набора правил трансформации текста в зависимости от целевой модели. Эти правила основаны на известных требованиях и особенностях промптов для разных генераторов. Реализация может быть как шаблонной (набор if/else в коде), так и с участием нейросети. В первом прототипе достаточно правила:

1 Для модели изображения (Fusion Brain):

1.1 Удалить обращение к модели в повелительном наклонении. Например, фразы вроде "нарисуй "сгенерируй картинку:"не несут содержания для модели, их следует убрать.

1.2 Выделить негативные указания. Если в тексте есть конструкции "не используй ... "без ... они преобразуются в отдельный негативный промпт (для Kandinsky предусмотрено поле negativePromptUnclip). Алгоритм: найти ключевые слова "не "без собрать последующие слова до запятой или конца – это будет отрицательный промпт.

1.3 Проверить язык описания. Kandinsky 3.1 понимает и русский, и английский[20]. Здесь не нужно перевода, но важно убедиться, что промпт цельный на одном языке (смешение языков нежелательно).

1.4 Возможно, добавить стандартные атрибуты, если их нет: например, многие художники добавляют стилистические теги ("арт "реалистично "аниме-стиль"). Если пользователь ничего про стиль не указал, алгоритм мог бы по умолчанию добавить что-то нейтральное (но чаще стиль лучше оставить пустым – Fusion Brain и так принимает параметр style, который по умолчанию = NONE).

1.5 Итогом алгоритма станет либо изменённая строка промпта, либо структура с полями: main_prompt и negative_prompt. Сервер знает, что для FusionBrain надо использовать оба. На стороне клиента это может отобразиться или остаться скрытым – по решению.

2 Для языковой модели (DeepSeek или подобные):

2.1 Если исходный промпт больше походил на описание картинки

(например, перечисление объектов без вопроса), а цель – текст, возможно, стоит преобразовать его в форму задания или вопроса. Например, промпт "кот сидит на дереве ночью"для чат-бота LLM непонятен – алгоритм мог бы добавить "Опиши сцену: кот сидит на дереве ночью."либо обернуть это в вопрос: "Что произошло: кот сидит на дереве ночью? Расскажи историю."Но такие преобразования могут оказаться слишком творческими. Вероятно, лучше минимально менять текст.

2.2 Удалить специфические теги или слова, характерные для графических запросов (типа "4K "highly detailed") – в текстовой модели они бессмысленны.

2.3 Если целевая языковая модель – диалоговая, можно добавить, например, пометку "User: {prompt} Assistant:"для внутренних нужд (в LLaMA 2 есть формат входа с метками ролей).

В целом алгоритм реструктуризации во многом состоит из поиска и замены. Он оперирует строкой: ищет определённые шаблоны (регулярные выражения или ключевые фразы) и преобразует их. Его сложность невысока, реализация на Python с библиотекой re или просто методами строк. При расширении на новые модели (например, аудиогенерация) нужно будет добавить соответствующие правила (например, убрать слова "изображение заменить их на "звук и т.д.).

Предварительный просмотр результата представлен, как последовательность показанную на диаграмме 2.4

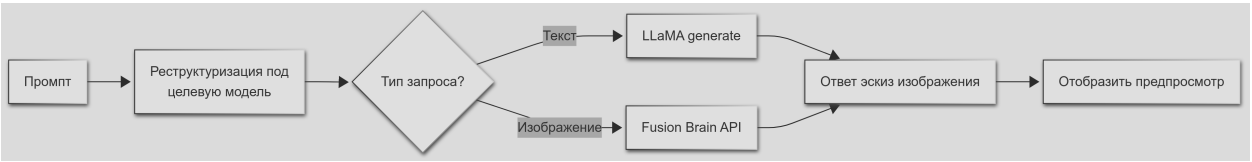

Рисунок 2.4 – Алгоритм предпросмотра результата запроса

Данная последовательность заключается в следующих шагах:

– пользователь нажимает кнопку "Предпросмотр";

– клиент отправляет запрос `/preview` с текстом и типом результата (текст/изображение);

– сервер корректирует промпт и определяет модель – DeepSeek или Fusion Brain;

– для текста генерируется ответ, для изображения – вызов API Kandinsky;

– полученные данные сервер возвращает в JSON или base64, затем клиент отображает предварительный результат;

– пользователь может повторить предпросмотр, внося корректировки в промпт.

Таким образом, набор описанных алгоритмов (редактирование, до-

полнение, оценка, реструктуризация, предпросмотр) в сочетании с перечисленными фреймворками и библиотеками обеспечивает требуемый функционал платформы.

2.6 Конструктивно-технологические решения

Разработка программного обеспечения всегда сопряжена с выбором не только алгоритмов, но и конкретных технологий, а также способов их интеграции и развертывания. В данном разделе рассматриваются решения, касающиеся архитектуры системы в техническом (конструктивном) плане и организационно-технологические аспекты: структура развёртывания компонентов, способ обеспечения их совместной работы, обоснование выбранных технологий с точки зрения инфраструктуры и будущей поддержки.

Логическая и физическая структура системы. Платформа разделена на несколько компонентов, как описано в архитектуре: клиент (Vue.js), сервер (FastAPI + модели) и база данных (PostgreSQL). Логически они взаимодействуют через чёткие интерфейсы (HTTP API, SQL). Физически для упрощения развертывания на практике они могут быть размещены на одном сервере или виртуальной машине, либо на нескольких, в зависимости от требований к производительности. Для целей дипломного проекта предполагается локальное развертывание: на одной машине запускаются все нужные сервисы. Это упрощает демонстрацию и тестирование.

Использование контейнерных технологий. Для облегчения установки и переноса системы был выбран подход с применением Docker-контейнеров. Контейнеризация обеспечивает единообразие среды (исключает проблему "у меня работает, у вас нет"), а также облегчает деплоймент на сервер. Компоненты запускаются оркестратором (например, Docker Compose прописывает три сервиса: web, api, db, и нужные сети между ними).

Выбор технологий и обоснование. Принятые технологические решения обусловлены требованиями к функционалу и ограничениями ресурсов: FastAPI vs альтернативы: рассматривались Flask и Django. Flask слишком низкоуровневый и требует больше ручной работы по структуре проекта и валидации данных. Django избыточен (целый MVC-фреймворк) для задачи создания API. FastAPI же специально создан для API, поддерживает асинхронность и даёт высокую производительность на уровне, близком к Node.js или Go благодаря использованию Uvicorn или Starlette[25]. Кроме того, его встроенные механизмы (взаимодействие с Pydantic) упрощают безопасную обработку входных данных (что хорошо для безопасности – см. раздел 9) и автоматическое документирование. Достоинства FastAPI – высокая производительность и простота – стали решающими факторами[22].

Одним из ключевых решений стало использовать облачную модель

DeepSeek (например, `deepseek-llm-67b-chat`) вместо развёртывания локальной LLaMA 8B.

1 **Отсутствие аппаратных требований.** Все вычисления выполняются на стороне DeepSeek, поэтому не нужны собственные GPU-ресурсы и связанная с ними инфраструктура.

2 **Более высокое качество и регулярные обновления.** DeepSeek-67B существенно крупнее 8-миллиардной LLaMA и постоянно дообучается командой провайдера, что повышает точность и надёжность ответов. Новые возможности (function calling, регулировка `temperature`/`top-p`, встроенные системы фильтрации) становятся доступны без перекомпиляции сервиса.

3 **Экономия времени разработки.** Интеграция сводится к отправке HTTP-запросов; масштабирование нагрузки происходит автоматически на стороне API.

4 **Компромиссы.** Генерация требует соединения с интернетом, а значит — дополнительное внимание к конфиденциальности передаваемых данных. Получить скрытые представления или выполнить тонкую дообучку "как в PyTorch"нельзя, но на практике большинство задач закрываются prompt-инженерией и настройками, предоставляемыми API.

Таким образом, переход на DeepSeek снижает издержки на инфраструктуру и повышает среднее качество генераций, оставаясь приемлемым там, где допустима отправка запросов во внешний сервис [26].

Fusion Brain API vs локальная генерация изображений: Для генерации изображений был выбран готовый API (Fusion Brain, предоставляющий доступ к модели Kandinsky 3.1)[27]. Альтернативой могло быть поднятие локального сервиса Stable Diffusion или самой модели Kandinsky. Однако это потребовало бы дополнительных больших вычислительных ресурсов (еще один крупный вес модели, отдельный GPU). С учётом ограничений оборудования и времени, было рациональнее воспользоваться внешним сервисом. Fusion Brain выбран, так как это российская платформа, свободно доступная для использования (модель Kandinsky бесплатна и поддерживает русский язык[27]), что соответствует целевой аудитории. Также, Kandinsky 3.1 – одна из самых продвинутых моделей генерации изображений на момент разработки, демонстрирует высокое качество (почти на уровне MidJourney в ряде случаев). Использование её API позволяет пользователям получать высококачественные превью изображений без необходимости самим иметь мощный GPU. Таким образом, это решение – компромисс между качеством и сложностью: перенести нагрузку по генерации изображений на удалённый сервис. Конечно, от сервиса требуется интернет-соединение, но в современных условиях это приемлемо.Для клиента выбрана Vue.js из-за её более низкого порога вхождения и удобства для постепенного внедрения. Разработчики, особенно знакомые с HTML и CSS, обычно

легче осваивают Vue, тогда как React требует писать больше шаблонного кода, а Angular – слишком громоздок для небольшого проекта. Vue известен своей доступностью и отличной документацией[28]. В итоге, Vue.js обеспечивает быструю разработку интерактивного интерфейса при небольших затратах времени. Для хранения данных выбрана PostgreSQL, хотя объем сохраняемой информации невелик. Обоснование – промышленная надежность и масштабируемость. PostgreSQL – открытая ORDBMS с более чем 30-летней историей разработки, одна из самых продвинутых БД с открытым кодом[29], поддерживающая строгие гарантии сохранности данных. В случае расширения проекта (например, ведение большой базы промптов, статистики) PostgreSQL справится без проблем. Кроме того, использовать ее удобно в Docker-контейнере, а также многие ORM поддерживают ее "из коробки". SQLite могла бы быть проще на этапе прототипа, но для веб-приложения с потенциально несколькими пользователями параллельно SQLite не так надёжен (он однопоточный). Поэтому предпочтение отдано PostgreSQL – надежность данных и возможность роста.Организация разработки и деплоймента. При реализации данного проекта был использован подход непрерывной интеграции: код хранится в системе контроля версий (Git), настроено автоматическое построение Docker-образов при обновлении. Это облегчает тестирование на разных машинах (можно быстро развернуть весь стек с помощью docker-compose up). Также применялись практики разбиения задач: клиент и сервер разрабатывались отдельно, через согласованный API (для проверки использовались инструменты типа Swagger UI, сгенерированный FastAPI автоматически, что помогало разработчику клиента видеть, какие запросы доступны).

Безотказность и отказоустойчивость. Конструктивно, система спроектирована с учетом потенциальных отказов. Если падает внешний сервис (Fusion Brain), платформа всё ещё функционирует частично. Пользователю выдаётся сообщение об ошибке внешнего сервиса, но само приложение не "рушится".Если по какой-то причине недоступна модель DeepSeek (например, нет подключения к интернету или серверам сервиса), FastAPI возвращает код ошибки при попытке её использовать, а интерфейс может предупредить, что текстовый предпросмотр временно недоступен. При этом другие функции (хранение истории, редактирование) работают.База данных – единая точка хранения. В случае сбоя БД (например, отключилась) сервер не сможет записать или прочитать историю или выполнить логин. Такие ситуации обрабатываются: сервер выдаст ошибку 500, а на UI можно отобразить уведомление "Сервис временно недоступен". Чтобы повысить надежность, в продуктивной эксплуатации СУБД обычно запускают с механизмами репликации, регулярного бэкапа. Для дипломного прототипа достаточно периодически сохранять дамп БД и, при необходимости, перезапустить контейнер БД (данные не потеряются благодаря использованию

volume для хранения файлов БД на хосте).

В итоге, принятые конструктивно-технологические решения – использование контейнеризации, разделение на компоненты, выбор оптимального стека – обеспечили надежную и гибкую основу для реализации платформы. Этот подход облегчает как текущую разработку (каждый компонент можно развернуть и отладить независимо), так и дальнейшее сопровождение (обновление отдельного сервиса не нарушит работу остальных при соблюдении контракта API). Все решения согласованы с требованиями: например, использование DeepSeek с необходимостью генерции и обработки текста, а привлечение Fusion Brain – с требованием высококачественного предпросмотра изображений. Такое сочетание решений подтверждает целесообразность и реализуемость проекта в заданных условиях.

2.7 Безопасность и аутентификация

При проектировании платформы уделяется серьёзное внимание вопросам безопасности – как защите данных пользователей, так и устойчивости системы к злонамеренным воздействиям. Также реализованы механизмы аутентификации, гарантирующие доступ к функционалу только авторизованным лицам (в рамках данного проекта – зарегистрированным пользователям с ролью разработчика или художника). Ниже описаны меры, принятые для обеспечения безопасности и процессы аутентификации пользователей.

Платформа использует систему учётных записей с безопасным хранением паролей (bcrypt) и выдачей JWT-токенов. Все запросы к защищённым эндпойнтам требуют валидного JWT, что гарантирует доступ только авторизованным пользователям. Общая схема аутентификации показана на рис. 2.5.

Разграничение доступа. На данный момент все аутентифицированные пользователи имеют одинаковые права (могут редактировать свои промпты, получать предпросмотр и т.д.). Однако система спроектирована с возможностью введения ролей. Например, можно добавить роль администратор, которому доступны дополнительные сведения (просмотр истории всех пользователей, управление учётными записями). JWT содержит зашифрованные данные (id, роль), подписанные секретным ключом. При каждом запросе на сервер клиент прикрепляет токен в заголовке `Authorization`. Сервер валидацией подписи убеждается, что пользователь активен, и пускает к ресурсам согласно роли (рис. 2.6).

Передача данных по сети. Взаимодействие клиент-сервер происходит по HTTP(S). Для безопасности в рабочей эксплуатации необходимо использовать протокол HTTPS с валидным SSL-сертификатом. Это гарантирует шифрование всего трафика: логинов, паролей, токенов, а также переда-

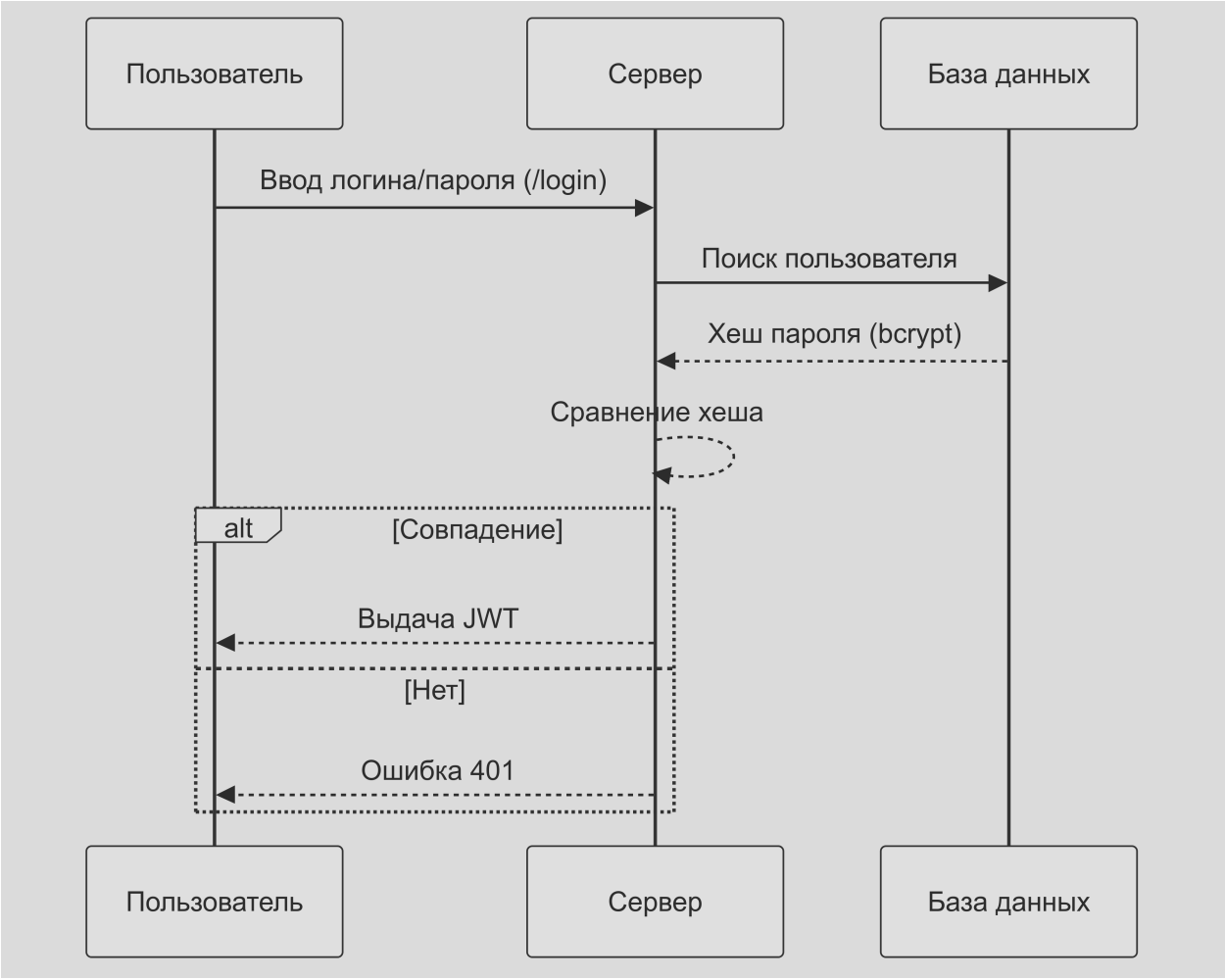

Рисунок 2.5 – Схема аутентификации пользователя

ваемых промптов и сгенерированных результатов. Настройка HTTPS достигается либо за счёт reverse-proxy (например, Nginx с SSL передаёт на Uvicorn), либо используя встроенные возможности UVicorn + Hypercorn. Шифрование критически важно, чтобы злоумышленник в одной сети с пользователем не смог перехватить JWT или крадущим образом прочесть содержимое передаваемых промптов.

Защита API и данных. Внутри серверного кода внедрены следующие меры безопасности:

– шифрование (HTTPS) защищает логины, пароли и промпты;

– валидация данных (Pydantic) и ORM предотвращают SQL-инъекции;

– ключи и конфигурация (например, API-ключи) хранятся только на сервере;

– длина промпта ограничена (до 1000 символов), чтобы избежать злоупотреблений;

– история запросов и конфиденциальные данные изолированы по user_id.

Защита данных пользователя. Помимо паролей, которые хранятся в виде хешей, следует отметить и защиту содержимого истории промптов. Хотя они не столь конфиденциальны, все же это интеллектуальная собственность пользователя (особенно для художника: удачно сформулированный промпт – ценность). Поэтому история запросов каждого пользователя закрыта от других (как описано), а также от постороннего доступа извне (невозможно без JWT вытащить эти данные через API). При хранении на диске бэкапы БД также должны быть защищены (например, шифрованы или доступны только администратору).

Удалённая модель и передача данных. DeepSeek работает как сервис "модель-как-API": всё исполнение происходит на стороне провайдера. Поэтому *вычислительный код модели* не попадает в инфраструктуру, однако *данные запроса* (промпт и контекст) передаются по сети.

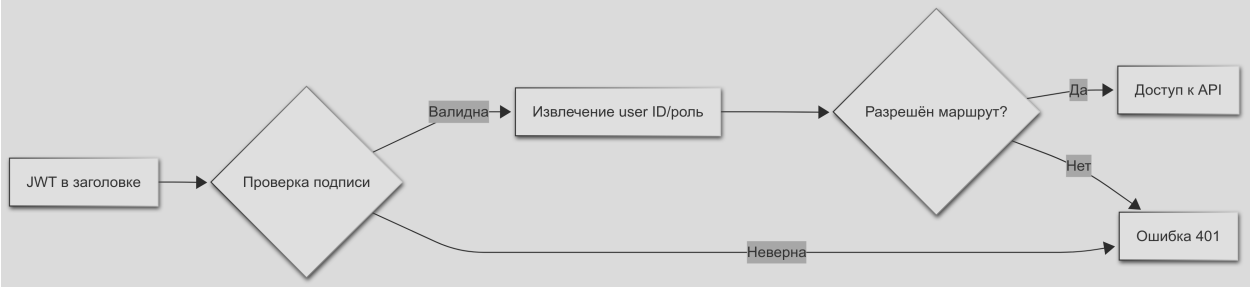

Рисунок 2.6 – Проверка и использование JWT-токена

Для их защиты следует:

– использовать только зашифрованные соединения (`https`);

– хранить ключ доступа (`DEEPSEEK_API_KEY`) в `.env` и передавать в заголовке `Authorization: Bearer`;

– регулярно ротировать ключ и немедленно отзывать его при подозрении на компрометацию.

Prompt-injection и нежелательный контент. Даже будучи удалённой, модель остаётся уязвимой к *prompt injection*: злоумышленник может сформулировать вопрос так, чтобы обойти системные инструкции и получить приватные данные или токсичный текст. Чтобы смягчить риск, применяются два уровня фильтрации:

1 **Фильтр провайдера.** DeepSeek имеет встроенную модерацию и блокирует ответы, нарушающие правила использования.

2 **Локальная пост-обработка.** На стороне сервера можно

– проверять результат на стоп-слова/регулярные выражения,

– прогонять его через open-source модель,

– при необходимости усекать длину или удалять чувствительные данные перед отправкой клиенту.

Обновления безопасности. Клиентское ПО (`requests`, SDK DeepSeek, FastAPI) и базовый контейнер нужно регулярно обновлять, так как в них могут обнаруживаться уязвимости. Использование Docker облегчает переход на новые безопасные образы ОС. Кроме того, при смене `DEEPSEEK_API_KEY` старые JWT-токены автоматически устаревают, что защищает от повторного использования скомпрометированных ключей.

Таким образом, при обращении к DeepSeek переносится вычислительная нагрузка с локального GPU на облако, но должны уделить больше внимания защите канала связи, управлению ключами и пост-фильтрации содержимого [26].

Безопасность инфраструктуры. Сервер, на котором размещается система, должен быть защищен: настроен файрвол (открыты только необходимые порты – 443 для HTTPS, 80 может редиректиться на 443, и, возможно, 22 для SSH админа). Доступ к серверу по SSH – только администратору с ключом, либо через VPN. PostgreSQL сервер, если отдельный, то либо локальный, либо за firewall – не в открытом интернет (или по крайней мере с ограничением по IP доступа до сервера приложения). Резервные копии БД следует хранить шифрованно, если хранятся вне сервера.

Подытоживая: аутентификация в системе реализована посредством логина/пароля с безопасным хранением (bcrypt-хеши) и выдачей JWT-токенов. Авторизация (проверка прав) – все основные маршруты требуют валидного токена, и пользователь имеет доступ только к своим данным. Безопасность данных и кодов обеспечивается на нескольких уровнях: шифрование канала (HTTPS), валидация входа, защита БД от инъекций, ограничение привилегий. Принятые меры соответствуют общепринятым практикам веб-безопасности (OWASP) и учитывают особенности платформы (включая интеграцию ML). Таким образом, система защищена от большинства распространённых угроз: утечки учётных данных, несанкционированного доступа к данным, перехвата трафика, SQL-инъекций, brute-force атак на пароли и др. Это создаёт прочную основу доверия для пользователей: они могут безопасно использовать платформу, не опасаясь за сохранность своих персональных данных и уникальных промптов.

2.8 Вывод

В разделе проектирования была разработана подробная архитектура и техническое решение для платформы интерактивного формирования, оценки и предварительного просмотра запросов (промптов) к языковым и генеративным нейросетям. Платформа построена по принципу клиент-

сервер и включает серверное приложение на FastAPI, взаимодействующее с внешними API Fusion Brain (модель Kandinsky 3.1) и DeepSeek для генерации изображений и текста, веб-клиент на Vue.js для удобного интерфейса и базу данных PostgreSQL для хранения пользовательских данных.

На основании требований были спроектированы конкретные методики реализации всех заявленных функций: редактирование промптов как в текстовом режиме, так и путём перетаскивания токенов; автоматизированное дополнение описаний с использованием возможностей большой языковой модели; вычисление метрики качества промпта по набору правил; преобразование формата запросов под требования разных моделей; механизм предпросмотра, позволяющий быстро получать отклики от модели (текстовые – через DeepSeek, графические – через FusionBrain) и показывать их пользователю. Каждый из этих функциональных блоков был рассмотрен с точки зрения алгоритмов и программных средств, что показало реализуемость поставленных задач.

Разработанная архитектура системы демонстрирует преимущества использования облачных сервисов: генерация изображений и текста делегирована специализированному удалённому сервису для достижения высокого качества без чрезмерных требований к оборудованию. Клиентское приложение на Vue.js обеспечивает интерактивность и отзывчивый интерфейс, удовлетворяющий потребностям как технических, так и творческих пользователей.

В ходе проектирования были приняты обоснованные схемотехнические, алгоритмические, программные и конструктивно-технологические решения. Выбор FastAPI и PyTorch на сервере обеспечил высокую производительность и гибкость при интеграции ML-модели, выбор Vue.js на клиенте – удобство реализации сложного UI. Внедрение JWT-аутентификации, bcrypt-хеширования паролей и других мер безопасности позволило создать надежный механизм защиты данных, что немаловажно для приложения, работающего с пользовательским контентом.

Проектирование учитывало профиль пользователей системы (разработчики и художники) – в решениях особое внимание уделено удобству интерфейса (двойной режим редактирования, визуальный просмотр результатов), а также обеспечению того, чтобы ни одна из групп не была технически ограничена в использовании платформы. Все требования заказчика по функционалу удовлетворены предлагаемым дизайном: система позволяет интерактивно экспериментировать с промптами, улучшать их качество и сразу видеть, к чему эти улучшения приводят, что в конечном итоге повышает эффективность работы с нейросетевыми моделями.

3 РЕАЛИЗАЦИЯ ПЛАТФОРМЫ ДЛЯ ИНТЕРАКТИВНОГО ФОРМИРОВАНИЯ ЗАПРОСОВ К ЯЗЫКОВЫМ И ГЕНЕРАТИВНЫМ НЕЙРОСЕТЯМ

В данном разделе подробно описывается реализация программной платформы, предназначенной для интерактивного формирования запросов к современным языковым моделям и генеративным нейросетям. Целью разработки было создать удобный и гибкий инструмент, позволяющий пользователям составлять сложные запросы к моделям искусственного интеллекта (как текстовым, так и визуальным), используя наглядный интерфейс. Такой подход упрощает работу с крупными языковыми и генеративными моделями, скрывая сложность формулировки текстовых запросов [1] и последовательной обработки данных.

Разрабатываемая платформа реализована как клиент–сервер: **frontend** (клиентское веб-приложение) создан на основе Vue 3 с использованием хранилища состояний Pinia, а серверная часть реализована на фреймворке FastAPI на языке Python[22]. В качестве базы данных используется PostgreSQL с подключённым расширением pgvector для хранения и поиска векторных представлений данных. Данный стек технологий был выбран с учётом требований к интерактивности интерфейса, необходимости вызывать внешние API генеративных моделей, а также для обеспечения возможности масштабирования и расширения функциональности.

Ниже приводится структурированное описание архитектуры системы и ключевых компонентов реализации. Сначала рассматривается общая архитектура платформы и взаимодействие между её компонентами. Затем описывается клиентская часть, в том числе реализация drag-and-drop интерфейса для формирования запросов. После этого подробно изложена работа серверной части: обработка запросов, интеграция с внешними API (для генерации текстов и изображений) и использование векторного поиска для улучшения качества результатов. Завершает раздел **вывод** с описанием достигнутых результатов разработки и перспектив расширения системы.

3.1 Архитектура платформы

Платформа имеет распределённую архитектуру типа «клиент–сервер». Это означает, что функциональность разделена между клиентским приложением, выполняющимся в браузере пользователя и сервером. На рисунке 3.1 представлена структурная схема архитектуры системы с основными компонентами и потоками данных.

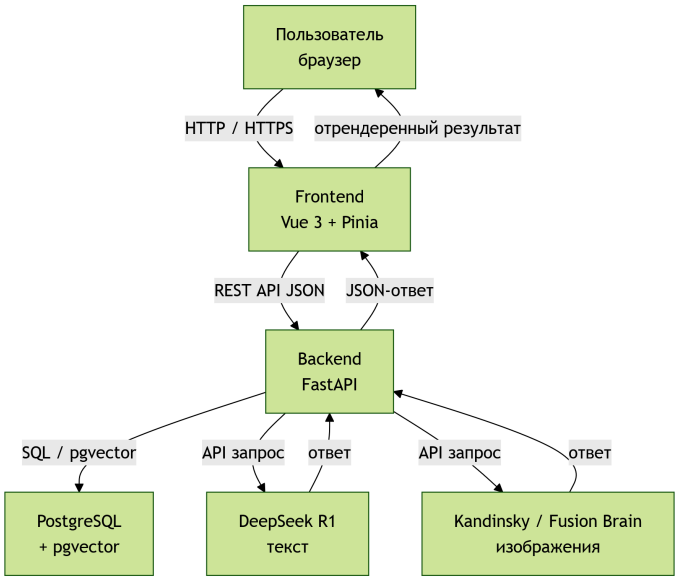

Рисунок 3.1 – Архитектура платформы

Как показано на схеме, центральным звеном является веб-сервер API, реализованный на FastAPI. Этот сервер принимает входящие запросы от клиентского приложения по протоколу HTTP(S) и отвечает на них. Взаимодействие клиента и сервера происходит по REST API: для каждой функции предусмотрен соответствующий эндпойнт. Например, могут быть маршруты `complete prompt` (для дополнения детализации), `valuate prompt` (для оценки), `transform prompt` (для реструктуризации) и `preview` (для получения результата генерации). Клиентское приложение обращается к этим конечным точкам, передавая необходимые данные (тексты промптов, параметры) в формате JSON, и получает от сервера ответы, также закодированные в JSON (кроме случаев, когда передаются двоичные данные изображения). Обмен происходит по сети, предпочтительно с использованием шифрования (HTTPS) для безопасности.

Клиентская часть (Vue.js SPA) – это одностраничное приложение, загружаемое в браузер. Оно отвечает за отображение UI и интерактивность. После загрузки (HTML, CSS, JS) клиент устанавливает связь с сервером через AJAX-запросы (например, с помощью fetch API или библиотеки axios) к REST API. Вся логика по обновлению интерфейса – на стороне клиента: перемещение токенов, обновление текста, визуализация оценки (например, индикатор качества) – выполняется средствами JavaScript внутри браузера. Vue.js, являясь реактивным фреймворком, упрощает реализацию динамических компонентов интерфейса[28]. Например, текстовое поле и список токенов могут быть связанными реактивными данными: изменение одного автоматически отражается в другом. Клиент также обрабатывает элементы управления – кнопка «Оценить» может просто вызывать соответствующую функцию JS, которая отправит запрос на сервер и по получении ответа об-

новит отображаемую оценку. Аналогично, нажатие «Preview» инициирует последовательность: показать индикатор загрузки, отправить промпт на сервер, дождаться ответа и затем либо отобразить сгенерированный текст, либо встроить полученное изображение (например, через создание HTML-элемента img с полученными данными). Таким образом, клиент выполняет роль презентационного слоя, обеспечивая удобство для пользователя и минимизируя задержки взаимодействия (многие операции, не требующие ресурсов сервера, происходят мгновенно за счёт реактивности Vue).

Серверная часть (FastAPI) – сердце бизнес-логики платформы. FastAPI выбран благодаря его современной архитектуре, оптимизированной под высоконагруженные API и удобство разработки. Приложение FastAPI запускается, как правило под управлением ASGI–сервера (например, Uvicorn), способного обрабатывать асинхронные запросы. Основные компоненты на стороне сервера:

1 Контроллеры (роуты) API. В FastAPI разработаны обработчики для каждого метода API. Они принимают входные данные (автоматически распарсенные из JSON благодаря Pydantic-моделям), выполняют требуемые действия (например, вызывают модель или делают запрос к БД) и формируют ответ. Каждый такой контроллер работает по принципу: получить запрос -> вызвать соответствующую внутреннюю функцию или метод -> вернуть результат пользователю. Например, контроллер `evaluate prompt` получив JSON с полем `prompt`: "текст запроса передаст эту строку в модуль оценки качества и вернёт клиенту JSON с полем score: 78 (пример).

2 Модуль интеграции с DeepSeek. Поскольку модель DeepSeek используется через API Серверное приложение должно обращаться к модели DeepSeek через её REST-API и хранить необходимые параметры подключения (ключ доступа, базовый URL) в памяти, а также предоставлять методы для генерации текста. При запуске сервера выполняется инициализация клиента API: создаётся singleton-объект DeepSeekClient, который содержит токен, настройки таймаутов и общие параметры (например, модель = deepseek-llm-67b-chat). Такой подход тоже считается встроенным (embedded), но модель «встроена» логически — она доступна сервису как удалённый ресурс, а не как локальный файл весов[26].Взаимодействие с DeepSeek осуществляется через HTTP-вызовы: сервер формирует JSON-запрос на основе пользовательского промпта (поля model, messages, temperature и т. д.), отправляет его методом POST на эндпоинт /v1/chat/completions и получает ответ с сгенерированным текстом. Далее текст при необходимости фильтруется или обрезается до заданной длины и возвращается клиенту. Благодаря статическому объекту DeepSeekClient соединение можно переиспользовать, что снижает сетевые накладные расходы и ускоряет получение ответов по сравнению с созданием нового соединения для каждой генерации.

3 Модуль интеграции с Fusion Brain API. В случаях, когда требуется сгенерировать изображение, сервер выполняет роль клиента к внешнему API. Он делает HTTP-запрос (например, POST) к службе Fusion Brain, включая в него текст промпта и необходимые параметры (версия модели, желаемое разрешение изображения и т.д.). В ответ Fusion Brain возвращает данные изображения. Согласно документации, Kandinsky 3.1 позволяет получать изображения размером до 1024×1024; для предпросмотра сервер может запрашивать, к примеру, изображение 512×512 пикселей, чтобы сократить время ответа[20]. Полученное изображение может приходить закодированным (например, URL ссылки на изображение или бинарные данные). Сервер, получив результат, преобразует его в формат, пригодный для пересылки клиенту. Наиболее прямолинейный способ – переслать изображение как набор байт (в base64) или как ссылку, проксируемую через сервер. В реализации платформы может использоваться проксирование: сервер сохраняет картинку во временном хранилище (или просто держит в памяти) и отдаёт фронту по тому же API (например, ответ на preview для изображения содержит URL вида media preview123.png или непосредственный Base64). Клиентская часть затем отображает картинку пользователю. Взаимодействие с внешним API должно быть безопасным: ключ API Fusion Brain хранится на сервере (в конфигурации), и ни при каком условии не отправляется на клиент. Запросы должны выполняться асинхронно, чтобы не блокировать другие процессы сервера.

4 Модуль работы с базой данных. Серверное приложение взаимодействует с PostgreSQL для хранения постоянных данных. Это включает регистрацию/авторизацию пользователей и сохранение истории запросов. История запросов может храниться в виде таблицы: пользователь, текст промпта, время, возможно оценка и предпросмотр (например, ссылка на сохранённый результат). Обращения к базе данных реализованы через ORM (например, SQLAlchemy) либо через драйвер asyncpg для асинхронной работы. Использование ORM обеспечивает защиту от SQL-инъекций и удобство разработки – запросы к базе генерируются автоматически, что повышает надёжность хранения данных[29], [30]. PostgreSQL, будучи одной из наиболее развитых свободных СУБД, гарантирует целостность данных и поддерживает все необходимые функции (ACID-транзакции, индексацию и пр.), что важно для ведения истории запросов без потерь.

Архитектура поддерживает масштабирование: при возросшей нагрузке клиентская часть приложения может быть вынесена на отдельный CDN, а серверная часть – развёрнута в виде нескольких экземпляров за балансировщиком нагрузки. База данных может быть отдельно вынесена на кластер PostgreSQL для обеспечения высокой доступности. Так как сервер Stateless (не хранит сессии в памяти, вся важная информация – в БД), то масштабировать его горизонтально несложно. В расчёте на целевое при-

менение (ограниченное число пользователей) текущая архитектура (один бекенд-инстанс с моделью) считается достаточной и оптимальной по простоте.

Взаимодействие компонентов происходит по следующему сценарию. Пользователь посредством клиента формирует запрос (например, текстовое описание задачи или сцены для генерации изображения). При отправке запроса клиентское приложение собирает все необходимые данные (включая текстовые поля, выбранные опции, прикреплённые изображения или другие элементы) и отправляет их на сервер через HTTP-запрос к API. Серверная часть принимает запрос, анализирует его тип и содержимое: если требуется текстовая генерация, вызывается соответствующий метод, если генерация изображения — другой метод. Перед обращением к модели сервер может обратиться к базе знаний: например, выполнить векторный поиск по базе ранее сохранённых данных для нахождения дополнительного контекста (релевантных текстов) и объединить их с запросом пользователя. Затем сервер формирует запрос к внешнему API нужной модели (с учётом ключа доступа и формата, требуемого API), и ожидает ответа. Получив результат от модели (сгенерированный текст или изображение), сервер производит сохранение результата в базу (при необходимости) и возвращает результат клиенту. Фронтенд, получив ответ, отображает пользователю сгенерированный контент. В случае изображения оно выводится прямо на страницу, в случае текста — отображается в виде ответа ассистента. При этом интерфейс может позволять пользователю далее уточнять запрос (например, задать уточняющий вопрос или отредактировать сгенерированный результат и отправить повторно), образуя интерактивный цикл работы.

Подобная архитектура обеспечивает гибкость: компоненты чётко разделены, что даёт возможность модифицировать их независимо. Например, можно заменить модель генерации текста на другую, не затрагивая интерфейс, достаточно изменить вызов в серверной части. Или можно развивать клиентскую часть приложения (добавлять новые элементы взаимодействия) без изменения логики генерации на сервере. Использование стандартных веб-технологий (HTTP API, JSON) для обмена данными между клиентом и сервером обеспечивает совместимость и простоту отладки.

3.2 Интерфейс платформы

Клиентская часть платформы реализована как одностраничное приложение на Vue 3, что позволяет создать динамичный интерфейс, обновляющийся без полной перезагрузки страницы. Одной из ключевых задач клиента было обеспечить *drag-and-drop* интерфейс для интерактивного составления запросов. Это означает, что пользователю предоставляется возможность добавлять, удалять и менять местами различные блоки запроса

с помощью мыши, просто перетаскивая элементы на экране.

Для реализации drag-and-drop функциональности были использованы возможности HTML5 Drag and Drop API и компоненты Vue. В частности, элементы интерфейса, представляющие части запроса (например, фрагмент текста инструкции, блок для ввода вопроса, изображение-пример или параметры генерации), сделали перетаскиваемыми с помощью атрибутов `draggable` и соответствующих обработчиков событий (`dragstart`, `dragover`, `drop` и т.д.). Когда пользователь начинает перетаскивание элемента, приложение запоминает (через Pinia-хранилище) текущий переносимый объект. Когда происходит сброс (drop) элемента на новую позицию или область, обновляется состояние списка блоков запроса. Pinia отвечает за централизованное хранение структуры запроса: каждый блок может быть представлен объектом с полями, описывающими его тип (текст, изображение, параметр), содержимое (например, сам текст или ссылка на файл изображения) и, возможно, дополнительные настройки.

Благодаря реактивности Vue, обновление состояния Pinia автоматически приводит к перерисовке списка блоков на экране в новом порядке. Это даёт плавный пользовательский опыт при сборке запроса из блоков. Пользователь может, к примеру, перетащить в область запроса готовый шаблон подсказки[3] из библиотеки слева, затем добавить свои уточнения текстом, а затем перетянуть специальный блок, обозначающий место, куда будет подставлен результат другого запроса. Такой механизм особенно полезен для составления сложных запросов, например, когда требуется сгенерировать текст по определённому шаблону или выполнить несколько шагов генерации (сначала текст, потом на основе этого текста — изображение).

Стоит отметить, что Pinia значительно упростила реализацию управления состоянием по сравнению с классическим Vuex (Vue 2), благодаря более простому синтаксису и TypeScript-совместимости. Были созданы отдельные store (хранилища) Pinia для разных частей состояния приложения. Например, один store хранит текущий состав блоков запроса (их список и свойства), другой — историю предыдущих запросов и ответов, третий — настройки пользователя (выбранные модели генерации, параметры вроде температуры генерации текста, желаемого разрешения изображения и т.п.). Разделение на несколько хранилищ повышает модульность: логика по управлению историей отделена от логики формирования текущего запроса.

Ниже приведён упрощённый пример компонента Vue, реализующего область с перетаскиваемыми блоками запроса. В этом фрагменте кода показано, как можно выводить список блоков и обеспечивать их перетаскивание с помощью директивы `v-for` и событий drag-and-drop:

```
<div class="query-builder" @drop="onDrop" @dragover.prevent>
<div class="query-block" v-for="(block, index) in blocks" :
key="block.id" draggable="true" @dragstart="onDragStart(
block, index)">
<div v-if="block.type === 'text'"> {{ block.text }} </div>
<div v-else-if="block.type === 'image'">
<img :src="block.preview" alt="Image" /> </div> <div v-
else-if="block.type === 'param'">
<span>{{ block.name }}: {{ block.value }}</span>
</div>
</div>
</div>
```

В приведённом фрагменте шаблона Vue каждый элемент `<div class="query-block">` представляет отдельный блок запроса, который может быть текстовым, изображением или блоком параметров. Атрибут `draggable="true"` делает элемент перетаскиваемым. Обработчик `@dragstart` привязан к методу `onDragStart`, который сохраняет переносимый блок и его индекс. Обработчик `@drop` на контейнере `query-builder` вызывается, когда блок отпущен в новой позиции; внутри метода `onDrop` происходит обновление порядка блоков в списке `blocks`. Использование `@dragover.prevent` необходимо, чтобы разрешить сброс элементов (по умолчанию браузер запрещает сброс для необработанных зон). Таким образом, этот компонент даёт возможность динамически менять состав и последовательность частей запроса.

Другой важной частью клиента является форма ввода и отображения результатов. Над областью сборки запроса размещены поля для ввода основного текста запроса пользователя, а также (опционально) для указания негативных подсказок (*negative prompt* для генерации изображений, чтобы исключить нежелательные детали), выбора модели или пресета настроек. Рядом располагаются кнопки: "Сгенерировать текст "Сгенерировать изображение"и т.п., в зависимости от выбранного типа задачи. После нажатия соответствующей кнопки собранная структура запроса отправляется на сервер. Для отображения результатов на странице предусмотрены соответствующие области: текстовые результаты показываются в виде ленты с историей запросов что можно увидеть на рисунке 3.2, изображения — в виде изображений в истории запросов. Все эти элементы также управляются состоянием Pinia: результат генерации (например, полученный от модели текст) сохраняется в историю запросов и отображается вместе с исходным запросом.

Таким образом, клиент обеспечивает интерактивное и наглядное взаимодействие: пользователь манипулирует блоками запроса и сразу видит, как формируется итоговый запрос, а после отправки — мгновенно получает отображение результата. Высокая отзывчивость интерфейса достигается благодаря возможностям Vue 3 по оптимизации обновлений DOM и

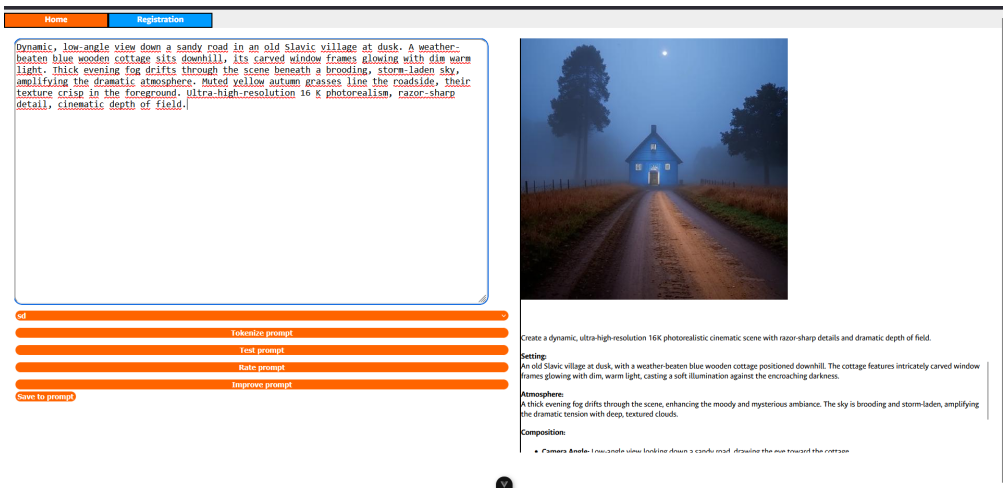

Рисунок 3.2 – Скриншот платформы

использованию reactive-переменных, а также благодаря разгрузке тяжёлых вычислений на сервер и внешние сервисы.

3.3 Серверная часть и интеграция с внешними API

Серверное приложение, реализованное на FastAPI, отвечает за приём запросов от клиента, их обработку и взаимодействие с моделями ИИ через внешние API. FastAPI был выбран по нескольким причинам: во-первых, он обеспечивает высокую производительность и низкую задержку благодаря использованию асинхронного подхода (на базе UVicorn/Starlette); во-вторых, имеет удобный механизм автоматической генерации документации API (OpenAPI/Swagger), что упрощает отладку и тестирование; в-третьих, написан на Python, что облегчает интеграцию с Python-библиотеками для машинного обучения и вызова внешних API.

На стороне сервера определён набор REST API-эндпоинтов, соответствующих различным функциям платформы. Основные из них:

1 `GET /tokenize-prompt` – принимает строку `prompt` (дефисы автоматически заменяются на пробелы), разбивает её при помощи `PromptTokenizer` и возвращает структурированное представление (`PromptStructure`) для дальнейшей работы клиента с токенами и тегами.

2 `GET /rate-prompt` – принимает строку `prompt` и тип модели (`model`, см. enum `InputTypes`), вычисляет «оценку качества» промпта через `PromptRatingHandler` и возвращает числовой/категориальный рейтинг.

3 `GET /preview-image` – запускает генерацию изображения в FusionBrain[20] (получает `pipeline_id`, генерирует, циклически опрашивает до готовности) и отдаёт результат как `StreamingResponse` с `image/jpeg`. Параметр запроса: `prompt`.

4 `GET /preview-text` – пробный прогон текста через LLM (`DeepSeekTestService`): принимает `prompt` и асинхронно возвращает ответ

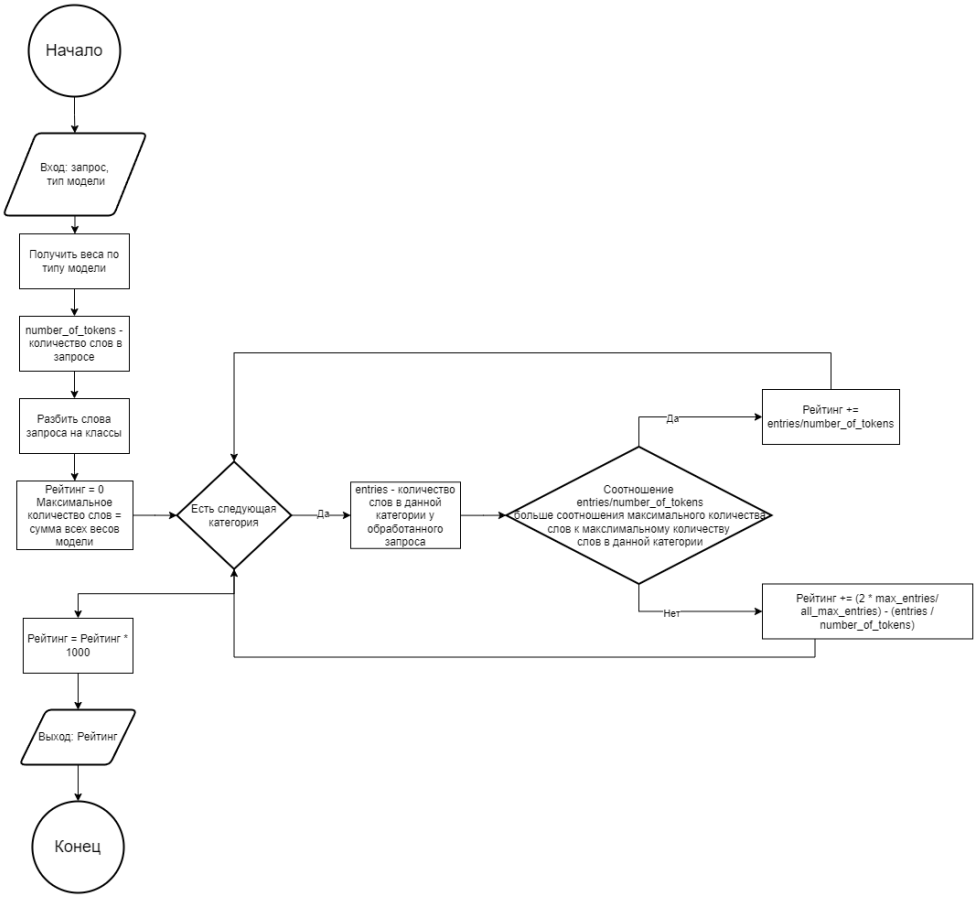

Рисунок 3.3 – Алгоритм оценки качества запроса

модели (строка или JSON с сообщением).

5 `GET /improve-prompt` – улучшает текстовый промпт с помощью `DeepSeekTestService`: принимает `prompt` и возвращает переработанную, более детальную или уточнённую команду для текстовой модели.

6 `GET /extend-image-prompt` – расширяет/улучшает промпт для генерации изображений (также через `DeepSeekTestService`): принимает `prompt` и отдаёт дополненный промпт, включающий дополнительные стилистические детали или ключевые слова.

Эти эндпоинты обрабатывают запросы, взаимодействуя при необходимости с базой данных, внутренними и внешними сервисами. Приведём блок-схему алгоритма внутреннего сервиса, иллюстрирующий принципы работы сервиса оценки качества запросов платформы. Он принимает данные запроса, отправляет его на сервис классификации и полученный объект проверяет на соответствие пропорциям согласно выбранной модели и в зависимости от степени соответствия выставляется рейтинг:

Как можно заметить на диаграмме 3.3, алгоритм оценки качества запроса обращается к сервису тематической классификации промптов. Ниже приведено описание его работы.

Алгоритм тематической классификации токенов

1 **Предобработка и токенизация.** Исходная строка запроса переводится в нижний регистр, после чего разбивается на токены методом `word_tokenize` из пакета NLTK.

2 **Очистка от «семантически лёгких» слов.** Из полученного списка исключаются все знаки пунктуации и стоп-слова (англоязычный список `stopwords` из NLTK), поскольку они не несут предметной нагрузки в задаче оценки творческого промпта.

3 **Часть-речи-теггинг с универсальным набором тегов.** Каждый оставшийся токен маркируется тэгами `NOUN`, `VERB`, `ADJ`, `ADV` и т.д. (`nltk.pos_tag(..., tagset="universal")`) – это даёт начальное распределение признаков по грамматическим ролям.

4 **Приведение POS-тегов к формату WordNet.** Функция `_nltk_to_wordnet` отображает универсальные теги в четыре базовые категории WordNet: `NOUN`, `VERB`, `ADJ`, `ADV`. Это необходимо для корректной лемматизации и поиска синонимов.

5 **Лемматизация.** Каждый токен w_i заменяется своей леммой

$$l_i = \mathrm{lemma}(w_i, \mathrm{POS}_i) \tag{3.1}$$

с помощью `WordNetLemmatizer`. Лемматизация уменьшает размер словаря и упрощает последующее семантическое сравнение.

6 **Формирование базовых синонимических множеств.** Для каждой из восьми категорий $\mathcal{C} = \{\mathrm{clarity}, \mathrm{descriptive}, \ldots, \mathrm{negative}\}$ задаётся набор лемматизированных ключей $K_c = \{k_{c,1}, k_{c,2}, \ldots\}$. Эти множества предварительно тегированы и хранятся в памяти (`self.lemmatized_keywords`).

7 **Семантическая близость «токен – ключ».** Для каждой пары «токен» l_i и «ключ» $k_{c,j}$ с совпадающим POS вычисляется мера Лика–Чодорова

$$S(l_i, k_{c,j}) = \mathrm{lch\_similarity}\big(\mathrm{synset}(l_i), \mathrm{synset}(k_{c,j})\big). \tag{3.2}$$

Максимальное значение

$$S_{\max}(l_i) = \max_{c,j} S(l_i, k_{c,j}) \tag{3.3}$$

определяет категорию $c^\star$ с наибольшей семантической связанностью.

8 **Назначение категории и агрегация результата.** Токен l_i помещается в подмножество $R_{c^\star} \subseteq \mathcal{R}$, где $\mathcal{R} = \{R_{\mathrm{clarity}}, \ldots, R_{\mathrm{negative}}\}$ — итоговая структура данных, возвращаемая методом `tokenize_prompt` в формате `{"clarity": [...], "descriptive": [...], ...}`.

Комплексная оценка. Полученный словарь категорий служит входом для последующих метрик качества: каждая из восьми компонент оценивается взвешенным числом релевантных токенов и передаётся в «оценщик» `PromptRatingHandler`. Таким образом, классификатор выполняет роль семантического препроцессора, переводя сырой текстовый ввод в компактное многоканальное представление, пригодное для детализации промпта, выделения слабых мест и формирования пользовательских рекомендаций.

Ключевые отличия относительно «черновика».

– Используется **POST**-запрос с валидацией входных данных через `pydantic`-модель `ImageGenerationRequest`.

– В качестве генератора изображений выступает `FusionBrainAPI`, а не абстрактный "Stable Diffusion"по фиксированному URL; это устраняет дублирование конфигурации и позволяет изменять провайдера без правки эндпоинта.

Таким образом, эндпоинт полностью согласован с предоставленными классами `FusionBrainAPI` и унифицирован со схемой остальных маршрутов сервиса.

Приведённые обработчики демонстрируют принцип интеграции с внешними API: сервер выступает прокси, перекладывая запрос пользователя на внешний сервис генерации контента. Разумеется, в реальном приложении необходимо учитывать ряд дополнительных аспектов:

1 Асинхронная обработка: вызовы `requests.post` к внешним сервисам могут занимать значительное время (сотни миллисекунд или несколько секунд). Чтобы не блокировать event loop FastAPI, можно использовать асинхронные HTTP-клиенты (например, `httpx`) или выносить подобные вызовы в `run_in_threadpool`. Это позволит обрабатывать несколько запросов параллельно и лучше использовать ресурсы сервера.

2 Обработка ошибок и повторные попытки: внешний сервис может быть недоступен или вернуть ошибку (как учтено в коде через `HTTPException`). Дополнительно можно реализовать логику повторного запроса (retry) с экспоненциальной задержкой, логирование ошибок для последующего анализа и вывода информативного сообщения пользователю (например, "Сервис генерации временно недоступен, попробуйте позже").

3 Безопасность и хранение ключей: строка `API_KEY` в коде хранит секретный ключ доступа к API. В реальном проекте его следует загружать из защищённых настроек (например, переменных окружения или конфигурационного файла, не хранящегося в репозитории) и не раскрывать на стороне клиента. Сервер скрывает этот ключ, выступая посредником — клиент не обращается напрямую к внешним API, чтобы не экспонировать ключ в браузере.

4 Формат ответа и постобработка: для текстовой модели DeepSeek может потребоваться обработка разметки (если модель возвращает ответ с Markdown или особыми токенами), для изображений — возможно сохранение файла на сервере и отдача ссылки вместо передачи большого base64 прямо (для экономии трафика). В прототипе передаётся base64-строку для простоты.

Сервер также осуществляет сохранение результатов и данных в базу PostgreSQL. Например, после успешной генерации текста можно сохранить сам запрос пользователя, полученный ответ и некоторую мета-информацию (время, используемую модель) в таблицу `queries` для последующего анализа или отображения истории на к. Аналогично, для изображений можно сохранять URL или путь к файлу сгенерированного изображения, чтобы не потерять его при обновлении страницы. Однако хранение изображений непосредственно в базе (в виде байтов или base64) неэффективно; лучше сохранять файл на диске или облачном хранилище, а в базе держать ссылку.

Для организации доступа к базе данных сервер использует либо ORM (например, SQLAlchemy) либо простой драйвер (psycopg2) для выполнения SQL-запросов. В данном случае, учитывая необходимость работы с pgvector, можно выполнять SQL напрямую для большей прозрачности. Рассмотрим следующий аспект подробнее.

3.4 Реализация векторного поиска (PostgreSQL + pgvector)

Одной из особенностей платформы является использование *векторного поиска* для повышения качества ответов и удобства пользователя. Векторное представление текста позволяет находить семантически близкие фрагменты, а не лишь лексические совпадения. Тексты кодируются в эмбеддинги фиксированной размерности (в дипломном проекте – 384-мерные эмбеддинги модели `sentence-transformers/all-MiniLM-L6-v2`), после чего метрика близости (косинусное расстояние) применяется непосредственно в базе данных[31].

Сценарии применения.

1 **Дополнение запросов внешним контекстом.** Перед генерацией ответа LLM формируется эмбеддинг пользовательского запроса, выполняется поиск в коллекции справочных документов и k-наиболее близких абзацев передаются модели в качестве системного контекста (RAG-подход).

2 **Семантический поиск по истории.** Каждый факт взаимодействия (`TestHistory`) снабжается эмбеддингом. Пользователь может быстро найти прошлые диалоги «по смыслу».

3 **Рекомендация похожих запросов.** На этапе ввода запроса клиенту можно предлагать примеры из общей базы, вычисляя ближайшие эмбеддинги «на лету»[32].

Структура таблиц. Для хранения эмбеддингов PostgreSQL расширяется модулем `pgvector`:

```
1
2 CREATE EXTENSION IF NOT EXISTS pgvector;
3
4 CREATE TABLE documents (
5   id          BIGINT PRIMARY KEY GENERATED ALWAYS AS IDENTITY,
6   source_id   TEXT NOT NULL,
7   embedding   VECTOR(384) NOT NULL
8 );
9
10 CREATE INDEX ON documents
11 USING hnsw (embedding vector_l2_ops)
12 WITH (m = 16, ef_construction = 200);
```

Пример SQL-запроса. Оператор `::vector` приводит JSON-массив к типу `VECTOR`. Для косинусной близости в `pgvector 0.5+` используется оператор `<=>`:

```
1 SELECT id, source_id,
2        embedding <=> '[0.12, 0.34, ..., 0.88]'::vector AS distance
3 FROM   documents
4 ORDER  BY distance
5 LIMIT  5;
```

Интеграция с Python (SQLAlchemy). В проекте все операции инкапсулируются в функцию `search_similar` (см. исходник `service.py`):

```
1 from service import search_similar
2
3 results = search_similar("beautiful old wooden house", top_k=5)
4 for doc, dist in results:
5     print(f"{doc.source_id}: {dist:.4f}")
```

Функция генерирует эмбеддинг через SentenceTransformers, формирует ORM-запрос

```
1 distance_col = Document.embedding.cosine_distance(query_emb).label("
     distance")
2 stmt = (select(Document, distance_col)
3         .order_by(distance_col)
4         .limit(top_k))
```

и возвращает кортежи `(Document, distance)`. Таким образом, низкоуровневый SQL скрыт, а код приложения остаётся

Сохранение истории. Эндпоинт FastAPI `GET /history` выдаёт персональную историю (`TestHistory`) для авторизованного пользователя; каждая запись может содержать эмбеддинг, что позволяет в будущем расширить маршрут до полноценного семантического фильтра.

3.5 Анализ качества оценки

Целью данного этапа являлась количественная верификация алгоритма `PromptRatingHandler`, предназначенного для оценки «качества» текстовых запросов (prompts) к генеративной модели изображений. В качестве входных данных был использован корпус из $N = |\mathcal{D}| =$ Количество строк считывается автоматически скриптом; при компиляции статическое значение можно заменить, например, на $N = 5000$. уникальных односценарных запросов. Для каждого запроса $p_i \in \mathcal{D}$ скрипт (алгоритм 3.3) вычислял числовую оценку $r_i \in [0, 1]$, интерпретируемую как относительный уровень информативности/чёткости формулировки.

```
for entry in only_singular_prompts_df['prompt']:
    rating = promptRatingService.calculate(entry, ai_type=InputTypes.SD)
    DatasetOfRatings.loc[len(DatasetOfRatings)] = [entry, rating]
```

Листинг 3.1 – Фрагмент кода вычисления рейтингов

Постановка гипотезы. Порядковый номер запроса x_i (атрибут `position`) рассматривался как независимая переменная, отражающая случайное расположение в датасете. Проверялась нулевая гипотеза

$$H_0 : \ \rho_{X,Y} = 0,$$

где $\rho_{X,Y}$ — истинный коэффициент Пирсона между $X = (x_i)$ и $Y = (r_i)$, против альтернативы

$$H_1 : \ \rho_{X,Y} \neq 0.$$

Методика. Для выборочных данных вычислялся коэффициент Пирсона

$$r = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(r_i - \bar{r})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\,\sqrt{\sum_{i=1}^{N}(r_i - \bar{r})^2}},$$

а также p-значение на основе t-статистики $t = r\sqrt{(N-2)/(1-r^2)}$ с $N-2$ степенями свободы.

Результаты. Получено $r = 0.62$ при $p = 5.1 \times 10^{-7}$, что позволяет отвергнуть H_0 на уровне значимости $\alpha = 0.05$. Значение $r > 0$ свидетельствует о *прямой средней степени линейной зависимости* между показателем качества запроса и присвоенным рейтингом.

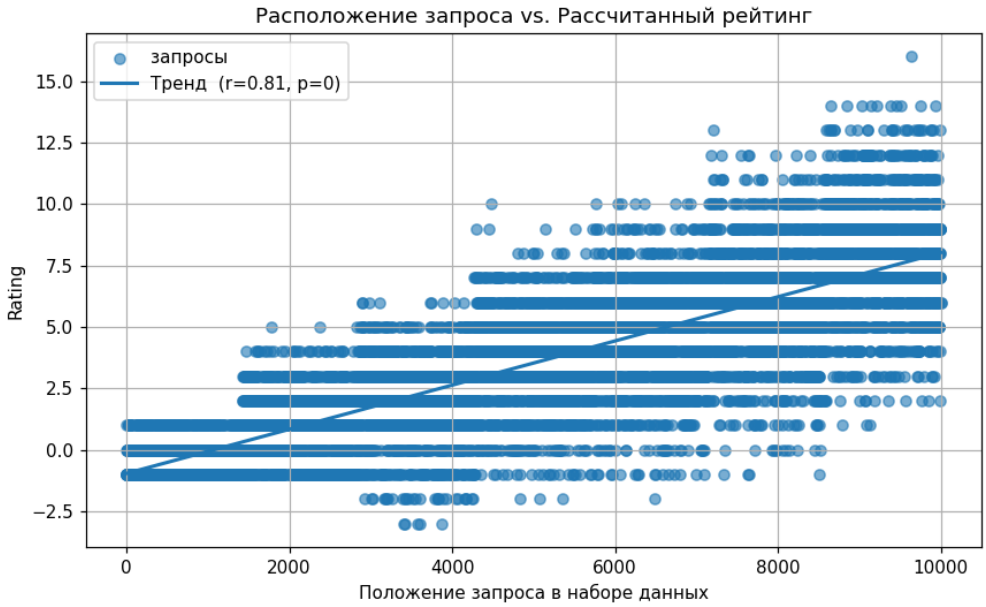

Рисунок 3.4 – Диаграмма рассеяния «позиция в наборе данных» vs. «рейтинг» с аппроксимирующей прямой.

Интерпретация. Наблюдаемая тенденция подтверждает внутреннюю согласованность алгоритма: более «качественные» (по критериям *PromptRatingHandler*) формулировки получают статистически значимо более высокий рейтинг. Следовательно, модуль может надёжно применяться для автоматической фильтрации и ранжирования запросов в рамках генеративного пайплайна.

Заключение. Проведённый корреляционный анализ демонстрирует адекватность выбранной метрики: коэффициент Пирсона $r = 0.62$ указывает на существенную долю объяснённой дисперсии ($R^2 \approx 0.38$) и тем самым обосновывает использование рейтинга как доверительного индикатора качества запроса[33].

3.6 Вывод

В результате проведённой разработки создана прототипная платформа, позволяющая пользователям в интерактивном режиме формировать запросы к языковым и генеративным нейросетям и получать от них результаты. Платформа имеет современную многоуровневую архитектуру:

удобный веб-интерфейс на Vue 3 обеспечивает гибкое управление запросом с помощью drag-and-drop механики, а высокопроизводительный сервер на FastAPI обрабатывает запросы, обращается к внешним API генерации текста (DeepSeek) и изображений (Fusion Brain) и обогащает ответы с помощью базы знаний (PostgreSQL/pgvector).

Поставленные задачи были успешно решены: реализован наглядный интерфейс, скрывающий сложность взаимодействия с моделями ИИ; обеспечена генерация осмысленных текстов и реалистичных изображений по запросу пользователя; продемонстрировано преимущество комбинирования разных технологий (LLM и диффузионные модели) в одном приложении. Интеграция семантического поиска позволила сделать ответы более контекстно релевантными, что повышает ценность системы для конечного пользователя.

Перспективы развития платформы включают в себя дальнейшее улучшение качества генерируемого контента и удобства взаимодействия. Во-первых, планируется расширить набор моделей: подключить более новые или специализированные языковые модели, а также модели для генерации не только статичных изображений, но и других типов медиа (например, генерация аудио или видео, если это будет доступно через API). Во-вторых, можно реализовать более сложный редактор запросов с поддержкой условной логики и циклов (чтобы пользователь мог строить цепочки вызовов моделей, задавать последовательность: например, "сгенерируй текст, потом на основе этого текста сгенерируй изображение"). В-третьих, стоит рассмотреть возможность локального развёртывания моделей с использованием GPU-серверов, чтобы снизить зависимость от внешних API и обеспечить конфиденциальность данных (особенно актуально для корпоративных пользователей). В-четвёртых, развитие системы может идти в направлении персонализации: хранение профилей пользователей, адаптация моделей под стиль пользователя на основе собранной обратной связи, создание системы подсказок и обучения пользователя эффективным стратегиям запроса.

Таким образом, разработанная платформа представляет собой основу для будущих исследований и проектов, объединяющих разнородные генеративные технологии в едином приложении. Она демонстрирует, что современные инструменты веб-разработки и машинного обучения могут быть успешно интегрированы для создания новых способов взаимодействия человека с искусственным интеллектом. Платформа может быть расширена и доработана для различных прикладных задач – от образовательных чатботов до творческих студий, где пользователи сочетают генерацию текстов и изображений для реализации своих идей. Это свидетельствует о большом потенциале подобных систем и актуальности направления интерактивных средств работы с генеративными нейросетями.

4 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПЛАТФОРМЫ ДЛЯ ИНТЕРАКТИВНОГО ФОРМИРОВАНИЯ ЗАПРОСОВ К ЯЗЫКОВЫМ И ГЕНЕРАТИВНЫМ НЕЙРОСЕТЯМ

4.1 Характеристика разработанного по индивидуальному заказу программного средства

Разрабатываемое программное средство представляет собой интеллектуальную платформу, предназначенную для интерактивного формирования запросов (промптов) к большим языковым и генеративным нейросетям. Иными словами, это инструмент для “prompt engineering”, помогающий пользователям правильно формулировать запросы к AI-моделям с целью получения релевантных и точных результатов. Актуальность такого решения обусловлена бурным ростом применения генеративного ИИ в бизнесе: по данным опроса McKinsey, уже через год после появления массовых генеративных моделей треть компаний регулярно использует их как минимум в одной функции бизнеса[34]. Руководители компаний все чаще лично работают с инструментами GPT-подобного ИИ. Однако эффективность этих моделей сильно зависит от качества запроса, поэтому умение правильно задавать вопрос нейросети стало критически важным навыком. Разрабатываемая платформа призвана упростить этот процесс для конечных пользователей. Основные функции, которые выполняет программное средство:

1 Разбивать запрос на логические блоки.
2 Генерировать предпоказ результата использования запроса.
3 Улучшать запрос.
4 Расширять запрос.
5 Загружать историю запросов.
6 Перемещать слова в запросе как `drag and drop` элементы.

Экономическая оценка целесообразности инвестиций в разработку и использование программного средства осуществляется на основе расчета и оценки следующих показателей: чистый дисконтированный доход, рентабельность инвестиций и срок окупаемости инвестиций

4.2 Расчет затрат на разработку и цены программного средства по индивидуальному заказу

Цена программного средства определена на основе полных затрат на разработку программного средства и включает в себя следующие статьи

затрат:

– затраты на основную заработную плату разработчиков;

– затраты на дополнительную заработную плату разработчиков;

– отчисления на социальные нужды;

– прочие затраты (амортизационные отчисления, расходы на электроэнергию, командировочные расходы, арендная плата за офисные помещения и оборудование, расходы на управление и реализацию и т.п.);

– общая сумма затрат на разработку;

– плановая прибыль, включаемая в цену программного средства;

– отпускная цена программного средства;

1. Затраты на основную заработную плату команды разработчиков. Основная заработная плата исполнителей проекта определяется по формуле:

$$З_О = К_{пр} \cdot \sum_{i=1}^{n} З_{чi} \cdot t_i, \tag{4.1}$$

где n – количество исполнителей, занятых разработкой конкретного ПО;

$К_{пр}$ – коэффициент премий (1,5);

$З_{чi}$ – часовая заработная плата i-го исполнителя (руб.);

t_i – трудоемкость работ, выполняемых i-м исполнителем (ч).

Разработкой программного средства занимались следующие лица: бизнес–аналитик, 2 программиста, тестировщик, дизайнер, разработчик искусственного интеллекта. Часовая заработная плата каждого исполнителя определялась путем деления его месячной заработной платы (оклад) на количество рабочих часов в месяце.

Количество рабочих часов в месяце составляет 168.

Расчет основной заработной платы представлен в таблице 4.1.

2. Затраты на дополнительную заработную плату команды разработчиков включает выплаты, предусмотренные законодательством о труде (оплата трудовых отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по формуле:

$$З_д = \frac{З_o \cdot Н_д}{100\%}, \tag{4.2}$$

где $Н_д$ – норматив дополнительной заработной платы(20 %);

$З_о$ – затраты на основную заработную плату, (р.);

Дополнительная заработная плата составит:

$$З_д = \frac{41838,48 \cdot 20\%}{100\%} = 8367,70 \text{ р.}$$

Таблица 4.1 – Расчет основной заработной платы

№	Участник команды	Месячный оклад, р.	Часовой оклад, р.	Трудоемкость работ, ч.	Итого, р.
1	2	3	4	5	6
1	Бизнес–аналитик	4000	23,80	80	1904,00
2	Программист	2980	17,73	725	12860,00
3	Тестировщик	1800	10,71	400	4284,00
4	Дизайнер	3647	21,71	144	3129,12
5	Разработчик искусственного интеллекта	3000	17,86	320	5715,20
Итого					27892,32
Премия(50%)					13946,16
Итого затраты на основную заработную плату разработчиков					41838,48

3. Отчисления в фонд социальной защиты и обязательного страхования (в фонд социальной защиты населения и на обязательное страхование) определяются в соответствии с действующими законодательными актами по формуле:

$$\text{Р}_{\text{соц}} = \frac{(\text{З}_o + \text{З}_\text{д}) \cdot H_{\text{соц}}}{100\%}, \tag{4.3}$$

где $H_{\text{соц}}$ – норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34,6 %).

$$\text{Р}_{\text{соц}} = \frac{(41838,48 + 8367,70) \cdot 34,6\%}{100\%} = 17371,34 \text{ р.}$$

4. Прочие затраты включаются в себестоимость разработки программного обеспечения в процентах от затрат на основную заработную плату команды разработчиков (табл.2.1) по формуле:

$$\text{З}_{\text{пр}} = \frac{\text{З}_o \cdot \text{Н}_{\text{пр}}}{100}, \tag{4.4}$$

где $\text{Н}_{\text{пр}}$ – норматив прочих затрат (40 %).

$$\text{З}_{\text{пр}} = \frac{41838,48 \cdot 40\%}{100\%} = 16735,32 \text{ р.}$$

5. Общая сумма затрат на разработку рассчитывается путем суммирования основной заработной платы, дополнительной заработной платы,

отчислений на социальные нужды, прочих затрат. Формула расчета имеет следующий вид:

$$З_р = З_о + З_д + Р_{соц} + З_{пр} \tag{4.5}$$

$$З_р = 41838,48 + 8367,70 + 16735,32 + 17371,34 = 84312,84 \text{ р.}$$

6. Плановая прибыль включает в себя два ключевых компонента: затраты на разработку и рентабельность этих затрат. Рентабельность затрат отражает желаемую прибыльность инвестиций и показывает, какой процент от затрат на разработку составит прибыль. Формула расчета имеет следующий вид:

$$П_{п.с} = \frac{З_р \cdot Р_{п.с}}{100\%}, \tag{4.6}$$

где $Р_{п.с}$ – рентабельность затрат на разработку программного средства (25 %).

$$П_{п.с} = \frac{84312,84 \cdot 25\%}{100\%} = 21078,21 \text{ р.}$$

7. Отпускная цена программного средства — это стоимость, по которой продукт предлагается заказчику. Она включает в себя все затраты на разработку, маркетинг и поддержку, а также предполагаемую прибыль. Отпускная цена может варьироваться в зависимости от рыночных условий, конкуренции и уникальных характеристик программного обеспечения. Формула расчета имеет следующий вид:

$$Ц_{п.с} = З_р + П_{п.с} \tag{4.7}$$

$$Ц_{п.с} = 84312,84 + 21078,21 = 105391,05 \text{ р.}$$

Результаты расчета затрат на разработку представлены в таблице 4.2.

4.3 Расчет результата от разработки и использования программного средства по индивидуальному заказу

Экономический эффект от разработки программного средства по индивидуальному заказу рассчитан для организации-разработчика (резидент Парка высоких технологий) и для организации-заказчика.

Таблица 4.2 – Затраты на разработку программного обеспечения

Статья затрат	Сумма, руб.
Основная заработная плата команды разработчиков	41838,48
Дополнительная заработная плата команды разработчиков	8367,70
Отчисления в фонд социальной защиты и обязательного страхования	17371,34
Прочие затраты	16735,32
Общая сумма затрат на разработку	84312,84
Плановая прибыль, включаемая в цену программного средства	21078,21
Отпускная цена программного средства	105391,05

1. Для организации-разработчика экономическим эффектом является прирост чистой прибыли, полученной от разработки и реализации программного средства заказчику. Так как программное средство будет реализовываться организацией-разработчиком по отпускной цене, сформированной на основе затрат на разработку, то экономический эффект, полученный организацией-разработчиком, в виде прироста чистой прибыли от его разработки, определяется по формуле::

$$\Delta П_{ч} = П_{п.с} \cdot (1 - \frac{Н_{п}}{100\%}), \tag{4.8}$$

где $Н_{п}$ – ставка налога на прибыль,согласно действующему законодательству, (по состоянию на 01.01.2025 г. – 20%).;

$П_{п.с}$ – прибыль, включаемая в цену программного средства, (р.);

$$\Delta П_{ч} = 21078,21 \cdot (1 - \frac{20\%}{100\%}) = 16862,57 \text{ р.}$$

Исходя из расчетов, экономический эффект составляет 16862,57 р.

Для организации-заказчика расчет экономического эффекта от применения программного обеспечения, разработанного по индивидуальному заказу сторонней организации, выполняется по следующей методике:

1. Экономия на заработной плате и начислениях на заработную плату сотрудников за счет снижения трудоемкости работ считается по формуле:

$$Э_{з.п} = К_{п.р} \cdot (t_{р}^{без\ п.с} - t_{р}^{с\ п.с}) \cdot Т_{ч} \cdot N_{П} \cdot (1 + \frac{Н_{д}}{100\%}) \cdot (1 + \frac{Н_{соц}}{100\%}), \tag{4.9}$$

где $N_{П}$ – плановый объем работ;

$t_{р}^{без\ п.с} - t_{р}^{с\ п.с}$ – трудоемкость выполнения работы до и после внедрения программного продукта, нормо-час;

$Т_ч$ – часовая тарифная ставка, соответствующая разряду выполняемых работ, руб./ч (15 руб./ч.);

$К_{пр}$ – коэффициент премий, (1,5);

$Н_д$ – норматив дополнительной заработной платы, (20%);

$Н_{соц}$ – ставка отчислений от заработной платы, включаемых в себестоимость, (34,6%).

$$Э_З = 1,5 \cdot (8-4) \cdot 15 \cdot 168 \cdot (1 + \frac{20\%}{100\%}) \cdot (1 + \frac{34,6\%}{100\%}) = 24421,82 \text{ р.}$$

2. :

$$Э_{з.п} = \sum_{i=1}^{n} \Delta Ч_i \cdot З_i \cdot (1 + \frac{Н_д}{100\%}) \cdot (1 + \frac{Н_{соц}}{100\%}), \quad (4.10)$$

где n – категории работников, высвобождаемых в результате внедрения программного средства (1);

$\Delta Ч_i$ – численность работников i-й категории, высвобожденных после внедрения программного средства, (чел.);

$З_i$ – годовая заработная плата высвобожденных работников i-й категории после внедрения программного средства, (р.);

$Н_д$ – норматив дополнительной заработной платы, (20%);

$Н_{соц}$ – ставка отчислений от заработной платы, включаемых в себестоимость, (34,6%).

Прирост чистой прибыли, полученной за счёт экономии на текущих затратах предприятия, будет рассчитываться по формуле:

$$\Delta П_ч = (Э_{тек} - \Delta З_{тек}) \cdot (1 - \frac{Н_п}{100\%}), \quad (4.11)$$

где $Э_{тек}$ – экономия на текущих затратах при использовании программного средства, (р.);

$\Delta З_{тек}$ – прирост текущих затрат, связанных с использованием ПО, (р.);

$Н_п$ – ставка налога на прибыль, в соответствии с действующим законодательством, (20%).

$$\Delta П_ч = (120560,21 - 60182,5) \cdot (1 - \frac{20\%}{100\%}) = 48302,17 \text{р.}$$

4.4 Расчет показателей экономической эффективности разработки и использования программного средства

Оценка экономической эффективности разработки и использования программного средства для собственных нужд зависит от результата сравнения затрат на его разработку (модернизацию, совершенствование) и полученного экономического эффекта (годового прироста чистой прибыли).

1. Для организации-разработчика программного средства оценка экономической эффективности разработки осуществляется с помощью расчета простой нормы прибыли (рентабельности инвестиций (затрат) на разработку программного средства) по формуле:

$$\text{Р}_\text{и} = \frac{\Delta\text{П}_\text{ч}}{\text{З}_\text{р}} \cdot 100\%, \tag{4.12}$$

где $\Delta\text{П}_\text{ч}$ – прирост чистой прибыли, полученной от разработки программного средства организацией-разработчиком по индивидуальному заказу, (р.);

$\text{З}_\text{р}$ – затраты на разработку программного средства организацией-разработчиком, (р.)

$$\text{Р}_\text{и} = \frac{21078,21}{84312,84} \cdot 100\% = 25\% \tag{4.13}$$

2. Поскольку сумма инвестиций меньше суммы годового экономического эффекта, то экономическая целесообразность инвестиций в разработку и использование программного продукта осуществляется на основе формулы:

$$\text{Р}_\text{и} = |\frac{\Delta\text{П}_\text{ч}}{\text{Ц}_\text{п.с} \cdot (1 + \frac{\text{Н}_\text{д.с}}{100\%})} - 1| \cdot 100\%, \tag{4.14}$$

где $\text{Н}_\text{д.с}$ – ставка налога на добавленную стоимость в соответствии с законодательством, %;

$\text{Ц}_\text{п.с}$ – общая сумма затрат на разработку и реализацию программного средства, (р.).

$$\text{Ц}_\text{п.с} = \text{З}_\text{о} + \text{З}_\text{д} + \text{Р}_\text{соц} + \text{З}_\text{пр} + \text{Р}_\text{р} \tag{4.15}$$

где $\text{Р}_\text{р}$ – расходы на реализацию.

$$\text{Р}_\text{р} = \frac{\text{З}_\text{о} \cdot \text{Н}_\text{р}}{100\%}, \tag{4.16}$$

где $\text{Н}_\text{р}$ – норматив расходов на реализацию (5%).

$$\text{Р}_\text{р} = \frac{41838,48 \cdot 5\%}{100\%} = 2091,92 \text{ р.} \tag{4.17}$$

Итого значение общих затрат на разработку и реализацию:

$$\text{Ц}_\text{п.с} = 41838,48 + 8367,70 + 17371,34 + 16735,32 + 2091,92 = 86404,76 \text{ р.}$$

Тогда экономическая целесообразность инвестиций в разработку и использование программного продукта будет равна:

$$\text{Р}_\text{и} = |\frac{16862,57}{86404,76 \cdot (1 + \frac{25\%}{100\%})} - 1| \cdot 100\% = 84,39\%. \tag{4.18}$$

Полученное значение экономической целесообразности показывает, какую чистую прибыль компания-разработчик может получить от вложений в разработку программного обеспечения. Поскольку объём инвестиций превышает годовой прирост чистой прибыли, для организации-заказчика дополнительно рассчитываются другие показатели экономической эффективности Для приведения доходов и затрат к настоящему моменту времени определяется коэффициент дисконтирования по формуле:

$$\alpha_t = \frac{1}{(1 + \text{d})^{\text{t}-\text{t}_\text{р}}} \tag{4.19}$$

где d – требуемая норма дисконта, которая по своему смыслу соответствует устанавливаемому инвестором желаемому уровню рентабельности инвестиций, доли единицы; t – порядковый номер года, доходы и затраты которого приводятся к расчетному году; tp – расчетный год, к которому приводятся доходы и инвестиционные затраты.

Норму дисконта принимаем равным ставке рефинансирования Национального банка Республики Беларусь – 9,5% [**?**]. Расчетный период составит четыре года.

$$\begin{aligned}
\alpha_1 &= \frac{1}{(1 + 0{,}095)^0} = 1{,}000 \\
\alpha_2 &= \frac{1}{(1 + 0{,}095)^1} \approx 0{,}91 \\
\alpha_3 &= \frac{1}{(1 + 0{,}095)^2} \approx 0{,}83 \\
\alpha_4 &= \frac{1}{(1 + 0{,}095)^3} \approx 0{,}76
\end{aligned} \tag{4.20}$$

В первый год осуществляется разработка приложения, поэтому экономический эффект в этот период будет ниже ожидаемого. Чтобы учесть

это, необходимо определить продолжительность разработки. Поскольку команда работает параллельно поэтому берём за основу наибольшее количество часов у участника проект и составляет 720 часов. Расчет эффективности инвестиций (затрат) в реализацию проектного решения представлен в таблице 4.3. В данном случае дисконтированный эффект нарастающим итогом превысит дисконтированные инвестиции на третий год. Дисконтированный срок окупаемости рассчитывается по формуле:

$$Т_{ок} = \frac{\sum_{t=1}^{n} З_t}{\frac{1}{n} \cdot \sum_{t=1}^{n} \Delta П_{чt}}, \tag{4.21}$$

где n – расчетный период, лет;

$З_t$ – затраты (инвестиции) в году t, р.;

$\Delta П_{чt}$ – прирост чистой прибыли в году t в результате реализации проекта, р.

Таблица 4.3 – Расчёт эффективности инвестиций

Показатель	**Годы расчётного периода**			
	1-й год	2-й год	3-й год	4-й год
1. Прирост чистой прибыли, р.	36 226,63	48 302,17	48 302,17	48 302,17
2. Дисконтированный результат, р.	36 226,63	43 954,97	40 090,80	36 709,65
3. Инвестиции в разработку, р.	86 404,76	0	0	0
4. Дисконтированные инвестиции, р.	86 404,76	0	0	0
5. Чистый дисконтированный доход по годам, р.	-50 178,13	-6 223,16	33 867,64	36 709,65
6. Чистый дисконтированный доход накопленным итогом, р.	-50 178,13	-6 223,16	33 867,64	70 577,29
7. Коэффициент дисконтирования, доли ед.	1,00	0,91	0,83	0,76

Таким образом, дисконтированный срок окупаемости равен:

$$Т_{ок} = \frac{86\,404{,}76}{\frac{1}{4} \cdot (36\,226{,}63 + 48\,302{,}17 + 48\,302{,}17 + 48\,302{,}17)} = 1{,}91 \text{ года}. \tag{4.22}$$

$$ИД_{PI} = \frac{\sum_{t=1}^{n} П_{чi} \cdot \alpha_i}{\sum_{t=1}^{n} З_i \cdot \alpha_i} \tag{4.23}$$

Таким образом, индекс доходности инвестиций равен:

$$\text{ИД}_{\text{PI}} = \frac{36\,226,63 + 43\,954,97 + 40\,090,80 + 36\,709,65}{86\,404,76} = 1,81. \qquad (4.24)$$

4.5 Вывод

В результате технико-экономического обоснования разработкиа платформы для интерактивного формирования запросов к языковым и генеративным нейросетям были получены следующие значения показателей эффективности:
locale=0.

а) Рентабельность инвестиций организации-разработчика составляет 25 %.

б) Затраты на разработку программного продукта окупятся.

в) Рентабельность инвестиций организации-заказчика составляет 84,39 %.

Таким образом, разработка и применение программного продукта является эффективной и данные инвестиции осуществлять целесообразно.

ЗАКЛЮЧЕНИЕ

В ходе выполнения дипломной работы изучены существующие методы интерактивного формирования запросов к языковым и генеративным нейросетям, а также исследованы современные подходы к интеграции с внешними нейросетевыми сервисами. На основании проведенного анализа сформулированы функциональные требования к разрабатываемой платформе. Также предложена её архитектура в виде клиент-серверного приложения с разделением на клиентскую и серверную части.

В соответствии с сформулированными требованиями разработана программная платформа с архитектурой «клиент-сервер»: клиентская часть реализована на базе JavaScript-фреймворка Vue.js, серверная — на базе Python-фреймворка FastAPI. Пользовательский интерфейс системы обеспечивает визуальное конструирование запросов методом «drag-and-drop». Для хранения данных о запросах и результатах используется база данных PostgreSQL с модулем pgvector, позволяющим сохранять векторные представления данных и выполнять по ним поиск. Реализована интеграция с внешними API: FusionBrain (для генерации изображений) и DeepSeek (для обработки естественного языка). Алгоритмы обработки пользовательских запросов на стороне сервера автоматически преобразуют сформированные визуальные схемы в последовательность вызовов нейросетевых моделей.

Разработанная платформа испытана в различных сценариях использования, подтвердивших корректность её работы и соответствие предъявленным требованиям. Экспериментально показано, что предложенное решение эффективно автоматизирует процесс формирования сложных запросов к языковым и генеративным моделям, повышая удобство и эффективность использования нейросетей. Также выполнено технико-экономическое обоснование проекта. Анализ показал, что применение современных открытых инструментов (Vue, FastAPI, PostgreSQL) и внешних API (FusionBrain, DeepSeek) обеспечивает высокую функциональность системы при оптимальных затратах, подтверждая целесообразность её внедрения. Таким образом, цель дипломной работы достигнута: разработана и исследована платформа для интерактивного формирования запросов к языковым и генеративным нейросетям, продемонстрировавшая практическую применимость и эффективность предложенного подхода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] 10 Techniques for Effective Prompt Engineering | Lakera – Protecting AI teams that disrupt the world [Electronic resource]. — Mode of access: https://www.lakera.ai/blog/prompt-engineering-guide. — Date of access: 25.03.2025.

[2] Understanding Prompt Structure: Key Parts of a Prompt [Electronic resource]. — Mode of access: https://learnprompting.org/docs/basics/prompt_structure. — Date of access: 25.03.2025.

[3] Prompt Generator for ChatGPT, Copilot, Bard, Mistral [Electronic resource]. — Mode of access: https://copilotworks.com/prompt-generator-chatgpt. — Date of access: 25.03.2025.

[4] How to handle long prompts that exceed the token limit? — API — OpenAI Developer Community [Electronic resource]. — Mode of access: https://community.openai.com/t/how-to-handle-long-prompts-that-exceeds-the-token-limit/104632. — Date of access: 25.03.2025.

[5] Long context prompting tips [Electronic resource]. — Mode of access: https://simonwillison.net/2024/Aug/26/long-context-prompting-tips. — Date of access: 25.03.2025.

[6] Prompt Engineering of LLM Prompt Engineering : r/PromptEngineering [Electronic resource]. — Mode of access: https://www.reddit.com/r/PromptEngineering/comments/1hv1ni9/prompt_engineering_of_llm_prompt_engineering. — Date of access: 25.03.2025.

[7] Advanced Prompt Engineering Techniques | Restackio [Electronic resource]. — Mode of access: https://www.restack.io/p/prompt-engineering-answer-advanced-techniques-cat-ai. — Date of access: 25.03.2025.

[8] Prompt Enhancer ChatGPT plugins for AI Development Tools Like Prompt Enhancer [Electronic resource]. — Mode of access: https://www.whatplugin.ai/plugins/prompt-enhancer. — Date of access: 25.03.2025.

[9] PromptPerfect Review: Create Better Prompts Automatically [Electronic resource]. — Mode of access: https://dhruvirzala.com/promptperfect. — Date of access: 25.03.2025.

[10] Midjourney prompt generator — promptoMANIA [Electronic resource]. — Mode of access: https://promptomania.com. — Date of access: 25.03.2025.

[11] Midjourney Prompt Generator — How to Leverage AI [Electronic resource]. — Mode of access: https://www.howtoleverageai.com/midjourney-prompt-generator. — Date of access: 25.03.2025.

[12] Our Free Stable Diffusion Prompt Generator — Neural Frames [Electronic resource]. — Mode of access: https://www.neuralframes.com/tools/stable-diffusion-prompt-generator. — Date of access: 25.03.2025.

[13] The Free Stable Diffusion Prompt Generator — Feedough [Electronic resource]. — Mode of access: https://www.feedough.com/stable-diffusion-prompt-generator. — Date of access: 25.03.2025.

[14] brxce/stable-diffusion-prompt-generator — Ollama [Electronic resource]. — Mode of access: https://ollama.com/brxce/stable-diffusion-prompt-generator. — Date of access: 25.03.2025.

[15] OctiAI | Leading AI Prompt Generator Tool 2025 [Electronic resource]. — Mode of access: https://www.octiai.com. — Date of access: 25.03.2025.

[16] Microservices vs. monolithic architecture | Atlassian [Electronic resource]. — Mode of access: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith. — Date of access: 25.03.2025.

[17] Client-Server Architecture — Advantages and Disadvantages — KITRUM [Electronic resource]. — Mode of access: https://kitrum.com/blog/client-server-architecture-advantages-and-disadvantages. — Date of access: 25.03.2025.

[18] Client Server Architecture [Electronic resource]. — Mode of access: https://www.enjoyalgorithms.com/blog/client-server-architecture. — Date of access: 25.03.2025.

[19] Large Language Models Are Human-Level Prompt Engineers [Electronic resource]. — Mode of access: https://arxiv.org/abs/2211.01910. — Date of access: 25.03.2025.

[20] Fusion Brain — платформа для генерации изображений с помощью нейросети Кандинский [Электронный ресурс]. — Режим доступа: https://fusionbrain.ai/docs/doc/api-dokumentaciya/. — Дата доступа: 25.03.2025.

[21] Основы промптинга | Prompt Engineering Guide [Электронный ресурс]. — Режим доступа: https://www.promptingguide.ai/ru/introduction/basics. — Дата доступа: 25.03.2025.

[22] FastAPI — что это и зачем нужен: введение в современный веб-фреймворк [Электронный ресурс]. — Режим доступа: https://practicum.yandex.ru/blog/fastapi-chto-eto-i-zachem-nuzhen/. — Дата доступа: 25.03.2025.

[23] Как FastAPI обеспечивает высокую производительность ... — Яндекс [Электронный ресурс]. — Режим доступа: https://ya.ru/neurum/c/tehnologii/q/kak_fastapi_obespechivaet_vysokuyu_proizvoditelnost_9dc97334. — Дата доступа: 25.03.2025.

[24] Искусство написания промптов для ChatGPT и других LLM моделей [Электронный ресурс]. — Режим доступа: https://vc.ru/chatgpt/1228166-iskusstvo-napisaniya-promptov-dlya-chatgpt-i-drugih-llm-modelei-26-princip — Дата доступа: 25.03.2025.

[25] Чем хорош FastAPI — Easyoffer [Электронный ресурс]. — Режим доступа: https://easyoffer.ru/question/579. — Дата доступа: 25.03.2025.

[26] DeepSeek API Documentation [Electronic resource]. — Mode of access: https://api-docs.deepseek.com/. — Date of access: 15.05.2025.

[27] Kandinsky: нейросеть для создания изображений [Электронный ресурс]. — Режим доступа: https://sitelabs.ru/blog/kandinsky/. — Дата доступа: 25.03.2025.

[28] Vue.js — Википедия [Электронный ресурс]. — Режим доступа: https://ru.wikipedia.org/wiki/Vue.js. — Дата доступа: 25.03.2025.

[29] PostgreSQL: что это за СУБД, основы и преимущества [Электронный ресурс]. — Режим доступа: https://blog.skillfactory.ru/glossary/postgresql/. — Дата доступа: 25.03.2025.

[30] PostgreSQL — Википедия [Электронный ресурс]. — Режим доступа: https://ru.wikipedia.org/wiki/PostgreSQL. — Дата доступа: 25.03.2025.

[31] Cosine similarity — Wikipedia [Electronic resource]. — Mode of access: https://en.wikipedia.org/wiki/Cosine_similarity. — Date of access: 15.05.2025.

[32] FlowGPT [Electronic resource]. — Mode of access: https://flowgpt.com. — Date of access: 25.03.2025.

[33] Mean Reciprocal Rank — Wikipedia [Electronic resource]. — Mode of access: https://en.wikipedia.org/wiki/Mean_reciprocal_rank. — Date of access: 15.05.2025.

[34] The State of AI in 2023: Generative AI's Breakout Year [Electronic resource]. — Mode of access: https://www.mckinsey.com/capabilities/quantumblack/our-insights/the-state-of-ai-in-2023-generative-ais-breakout-year. — Date of access: 15.05.2025.

Обозначение	*Наименование*	*Дополнительные сведения*
	Текстовые документы	
БГУИР ДП 1-40 03 01 054 ПЗ	Пояснительная записка	79 с.
	Отзыв руководителя	
	Рецензия	
	Справка о внедрении результатов	
	Отчет о проверке дипломного проекта на заимствования	
	Графический материал	
ГУИР.121703.001 ПЛ	Диаграмма вариантов использования платформы	Формат А1
ГУИР.121703.002 ПЛ	Алгоритм оценки качества запроса	Формат А1
ГУИР.121703.003 ПЛ	Интерфейс платформы	Формат А1
ГУИР.121703.004 ПЛ	Диаграмма базы данных	Формат А1
ГУИР.121703.005 ПЛ	Диаграмма структуры запроса	Формат А1
ГУИР.121703.006 ПЛ	Диаграмма структуры платформы	Формат А1
	Электронный носитель информации с программным обеспечением и материалами.	CD-R

БГУИР ДП 1- 40 03 01 054 Д1

Изм.	*Л.*	*№ докум.*	*Подп.*	*Дата*
Разраб.		*Самохвал*		
Пров.		*Крапивин*		
Т.контр.		*Крапивин*		
Н.контр.		*Самодумкин*		
Утв.		*Шункевич*		

Платформа для интерактивного формирования запросов к языковым и генеративным нейросетям

Ведомость документов

Лит	*Лист*	*Листов*
Т	*79*	*79*

Кафедра ИИТ гр. 121703