МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

Национальный исследовательский ядерный университет «МИФИ»

Институт интеллектуальных кибернетических систем

КАФЕДРА КИБЕРНЕТИКИ (№ 22)

Направление подготовки 09.03.04 Программная инженерия

Расширенное содержание пояснительной записки

к учебно-исследовательской работе студента на тему:

Разработка модели интеграции технологического ядра XiYan-SQL во внешний веб-сервис и его программная реализация

Группа	Б22-534		
Студент			Баранов А. Т.
Руководитель	10 б.		Трофимов А. Г.
Научный консультант			

Москва 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

Национальный исследовательский ядерный университет «МИФИ»

Институт интеллектуальных кибернетических систем

КАФЕДРА КИБЕРНЕТИКИ (№ 22)

Задание на УИР

Студенту гр. Б22-534 Баранову Александру Тимуровичу

ТЕМА УИР

Разработка модели интеграции технологического ядра XiYan-SQL во внешний веб-сервис и его программная реализация

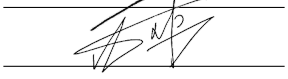
ЗАДАНИЕ

№ п/п	Содержание работы	Форма отчетности	Срок исполнения	Отметка о выполнении (Дата, подпись руководителя)
1.	**Аналитическая часть**			
1.1.	Провести обзор традиционных способов взаимодействия с базами данных.	Список литературы, текст РСПЗ	28.02.2025	
1.2.	Провести обзор интерфейсов для взаимодействия с базами данных.	Список литературы, текст РСПЗ	07.03.2025	
1.3.	Провести сравнение существующих систем, переводящих естественный язык на язык запросов SQL.	Список литературы, текст РСПЗ	14.03.2025	
1.4.	Провести сравнение существующих веб-сервисов, позволяющих производить запросы к базам данных на естественном языке.	Список литературы, текст РСПЗ	24.03.2025	
1.5.	*Оформить расширенное содержание пояснительной записки (РСПЗ).*	Текст РСПЗ	28.03.2025	
2.	**Теоретическая часть**			
2.1.	Поставить задачу разработки модели взаимодействия серверной части внешнего сервиса с системой XiYan.	Текст ПЗ	03.04.2025	
2.2.	Смоделировать работу MCP-клиента на стороне веб-сервиса.	Описание модели, текст ПЗ	10.04.2025	
3.	**Инженерная часть**			
3.1.	Сформулировать требования к системе и графическому интерфейсу пользователя.	Текст ПЗ	24.04.2025	
3.2.	Разработать архитектуру прототипа веб-сервиса.	Текст ПЗ	27.04.2025	
3.3.	Выбрать стек технологий для разработки и привести обоснования его выбора.	Текст ПЗ	01.05.2025	

3.4.	Выполнить программную реализацию прототипа веб-сервиса	Описание основных функций и модулей, текст ПЗ	08.05.2025	
3.5.	Реализовать графический пользовательский интерфейс	Описание графического интерфейса, текст ПЗ	21.05.2025	
3.6.	Провести тестирование прототипа веб-сервиса	Текст ПЗ	17.05.2025	
4.	**Технологическая и практическая часть**			
5.	*Оформить пояснительную записку (ПЗ) и иллюстративный материал для доклада*	Текст ПЗ, презентация	22.05.2025	

ЛИТЕРАТУРА

1. *Бородин Д. С.*, *Строганов Ю. В.* К задаче составления запросов к базам данных на естественном языке // Новые информационные технологии в автоматизированных системах. — 2016. — № 19. — С. 119—126. — URL: https://cyberleninka.ru/article/n/k-zadache-sostavleniya-zaprosov-k-bazam-dannyh-na-estestvennom-yazyke (дата обр. 15.05.2025).
2. Natural Language to SQL: Where Are We Today? / H. Kim [et al.] // Proc. VLDB Endow. — 2020. — June. — Vol. 13, no. 10. — P. 1737–1750. — ISSN 2150-8097. — DOI: 10.14778/3401960.3401970. — URL: https://dl.acm.org/doi/10.14778/3401960.3401970 (visited on 05/15/2025).
3. Large Language Model Enhanced Text-to-SQL Generation: A Survey / X. Zhu [et al.]. — 10/08/2024. — DOI: 10.48550/arXiv.2410.06011. — arXiv: 2410.06011 [cs]. — URL: http://arxiv.org/abs/2410.06011 (visited on 06/14/2025). — Pre-published.
4. Exploring the Landscape of Text-to-SQL with Large Language Models: Progresses, Challenges and Opportunities / Y. Huang [и др.]. — Вер. 1. — 28.05.2025. — DOI: 10.48550/arXiv.2505.23838. — arXiv: 2505.23838 [cs]. — URL: http://arxiv.org/abs/2505.23838 (дата обр. 15.06.2025). — Пред. пуб.
5. A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL / Y. Gao [и др.]. — Вер. 3. — 10.02.2025. — DOI: 10.48550/arXiv.2411.08599. — arXiv: 2411.08599 [cs]. — URL: http://arxiv.org/abs/2411.08599 (дата обр. 15.06.2025). — Пред. пуб.

Дата выдачи задания:	Руководитель		Трофимов А. Г.
24.02.2025	Студент		Баранов А. Т.

Реферат

Общий объем основного текста, без учета приложений — 51 страниц, с учетом приложений — 54. Количество использованных источников — 38. Количество приложений — 2.

Ключевые слова: нетехнические пользователи, базы данных, Text-to-SQL, NLIDB.

Целью работы является разработка модели взаимодействия с ядром XiYan-SQL на основе MCP-клиента и ее практическая реализация в виде прототипа с эмуляцией ключевых компонентов.

0.1 == TODO ==

В первом разделе проводится исследование проблематики, выявлению современных подходов для ее решения и проверка актуальности задачи.

Второй раздел посвящен постановке разработческой задаче и описанию ее решения.

В третьем разделе приводится описание процесса проектирования системы и ее программной реализации.

Четвертый раздел посвящен описанию процесса пользования продуктом и его тестированию.

В приложении А приведены графики лидеров в бенчмарках для Text-to-SQL.

В приложении Б представлены листинги ключевых фрагментов исходного кода, демонстрирующие реализацию основных модулей и функций спроектированного веб-сервиса.

Содержание

Введение

Базы данных (БД) являются неотъемлемой частью множества сфер деятельности, включая научные исследования, бизнес, государственное управление и многие другие. Они предоставляют быстрый и эффективный доступ к огромным массивам данных [1].

Для экспертов в доменной области, обладающих знаниями языков запросов, получение данных является удобным и простым процессом. Однако для пользователей, не являющихся экспертами, особенно тех, кто работает вне IT-сферы, извлечение информации из БД предствляет значительную трудность. Она представляет из себя барьер: необходимость владения специальными навыками, в частности, составлению запросов с помощью специальных языков. Ситуация усугубляется для тех, чей родной язык отличается от английского. Поскольку в большинстве случаев формальные языки взаимодействия с базами данных являются англоязычными, возникает дополнительный языковой барьер, который еще больше усложняет взаимодействие ними [2; 3].

Таким образом, формируется категория пользователей, которые имеют доступ к необработанным данным и испытывают потребность в их анализе, но не могут самостоятельно извлечь информацию из-за отсутствия технических навыков работы с базами данных и/или языкового барьера.

Существующие пользовательские интерфейсы, адаптированные для данной категории, могут быть «менее дружелюбными или вовсе отсутствовать», что создает потребность в альтернативных решениях [2]. Они позволят сократить время при принятии решений, ускорят бизнес-процессы в компаниях, а также позволят их командам быстрее обучаться работе с данными и сократит нагрузку на дата-аналитиков, демократизируя доступ к данным и делая их доступными для более широкого круга пользователей.

В данной работе для решения обозначенной проблемы проводится комплексный анализ современных подходов. В первом разделе рассматривается эволюция пользовательских интерфейсов к базам данных, обосновывается выбор в пользу естественно-языковых решений (NLIDB) и исследуется история развития технологии Text-to-SQL. На основе этого анализа для реализации прототипа выбирается и обосновывается передовое технологическое ядро — открытый фреймворк XiYan-SQL, лидирующий в ключевых отраслевых бенчмарках.

Далее, работа переходит от выбора технологии к моделированию взаимодействия серверной части веб-сервиса с XiYan-SQL для интеграции. Во втором разделе описывается со-

временная многоагентная архитектура, лежащая в основе XiYan-SQL, и протокол взаимодействия *Model Context Protocol (MCP)*, который является стандартом для таких систем. Центральным элементом раздела становится проектирование архитектуры **MCP-клиента** — компонента-оркестратора, который будет управлять сложным жизненным циклом обработки запроса на стороне разрабатываемого веб-сервиса. В заключительном разделе эта теоретическая модель интеграции транслируется в инженерное решение: с помощью UML-диаграмм проектируется архитектура прототипа веб-сервиса, описывается технологический стек для его реализации, приводится программная реализация и тестирование.

Целью данной работы является проектирование модели интеграции современного многоагентного NLIDB-ядра XiYan-SQL во внешний веб-сервис посредством разработки модели компонента-оркестратора (MCP-клиента). Практическим подтверждением жизнеспособности предложенной модели служит ее реализация в виде программного прототипа, предоставляющего естественно-языковой интерфейс к базам данных.

Таким образом, ключевым результатом работы является проект архитектурного решения для встраивания современных многомодульных Text-to-SQL систем в прикладные приложения, а также его прототипная реализация с эмуляцией внешнего компонента, что закладывает фундамент для дальнейшей полноценной разработки.

1. Анализ проблематики взаимодействия нетехнических пользователей с базами данных и обзор перспективных способов решения данной проблемы

Аннотация. *В данной аннотации представлен обзор проблематики взаимодействия нетехнических пользователей с базами данных. Проведен анализ существующих моделей представления данных и способов взаимодействия с СУБД, выявлен основной барьер для пользователей без технической подготовки. Рассмотрены и сравнены современные интерфейсы работы с базами данных, ориентированные на нетехнических специалистов, выделен наиболее перспективный подход — естественно-языковые интерфейсы (NLIDB). Описана эволюция методов преобразования естественного языка в SQL-запросы, проведён анализ современных моделей и веб-сервисов, а также обоснована актуальность разработки собственного веб-сервиса на базе открытого фреймворка XiYan-SQL.*

1.1 Обзорный анализ традиционных способов взаимодействия разработчиков с базами данных и постановка задачи исследования

Аннотация. *Для разработки эффективных методов помощи технически неподкованным пользователям необходимо, в первую очередь, систематизировать виды баз данных, с которыми они могут столкнуться, и проанализировать традиционные способы взаимодействия программистов с ними. В данном разделе рассматриваются методы структурирования данных, известные на данный момент, а также освещаются методы общения разработчиков с СУБД. На этой основе формулируется основной предмет дальнейшего исследования.*

Информационные технологии уже стали частью жизни общества и продолжают своё развитие, а объём данных постоянно увеличивается в геометрической прогрессии, поэтому вопрос оптимальной систематизации и хранения данных особенно актуален в современном мире [1].

Информация и данные лежат основе информационных технологий. В теории баз данных понятия *информации* и *данных* являются ключевыми. Под *информацией* понимаются любые сведения о каком-либо событии, процессе, объекте. Под *данными* понимается информация, представленная в определенном виде, позволяющем автоматизировать ее сбор, хранение и дальнейшую обработку человеком или информационным средством [4].

Приведём еще несколько основных понятий и определений из теории баз данных:

– *База данных* (БД) — это именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области. БД состоит из множества связанных файлов.
– *Система управления базами данных* (СУБД) — совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

1.1.1 Классификация баз данных по модели представления данных

Исторически сложились различные подходы к структурированию данных, реализованных в СУБД, поэтому их можно классифицировать по модели представления данных.

Устаревшие модели. До появления современных информационных технологий и развития компьютерной техники, методы хранения данных, использовавшиеся в библиотеках и архивах, были взяты за основу для их упорядоченного хранения [5]. Эти ранние модели, хотя и менее распространены сегодня, заложили основу для более сложных систем:

– **Иерархическая модель**. Данные в ней представлены в виде дерева. Она отлично подходит для описания предметной области, где объекты можно упорядочить по степени подчиненности друг другу. Данная модель БД быстро предоставляет пользователю запрашиваемую информацию, однако она не отличается гибкостью. Примеры использования: LDAP и Active Directory, база настройки Windows WMI и Реестр Windows, Google App Engine DataStore API.
– **Сетевая модель**. Является расширением иерархической базы данных. В отличие от иерархической, у потомка может быть более одного родителя, что позволяет описывать более широкий класс взаимоотношений между данными, например отношения «многие ко многим». Недостатком является сложность и неудобство работы, возникающих при увеличении размера БД. Примеры использования: графические системы формирования 3D-изображений, системы пространственной координации объектов.

Реляционные базы данных (SQL). Данные в такой модели представляются в виде таблиц. Столбец одной таблицы может быть связан со столбцом другой таблицы. Так реализовано взаимоотношение между объектами [1; 5—7]. Из преимуществ данного класса баз данных можно выделить строгую стандартизированность и основное взаимодействие с помощью специального языка SQL, а также надежность, популярность и успешность в сообществе. Из недостатков можно выделить отсутствие горизональной масштабируемости и стро-

гую структуризацию данных [5]. Примеры реляционных СУБД (РСУБД): Oracle, MySQL, Microsoft SQL Server, PostgreSQL, SQLite, DB2, Sybase.

NoSQL базы данных. Изначально термин «NoSQL» появился в конце 90-х годов 20-го века, когда он расшифровывался «Без SQL», то есть взаимодействие с ними производилось без языка SQL. Нынешняя интерпретация как «Not only SQL» сформировалось лишь в 2009 году, и подразумевает под собой, что поддержка SQL является лишь одним из компонентов системы [1; 6]. Они предлагают более гибкий подход, идеальный для работы с неструктурированными или динамическими данными, без жесткой предопределенной схемы. Из преимуществ можно отметить поддержку горизонтального масштабирования и поддержку хранения различных типов данных, из недостатков — большее потребление дискового пространства, сниженную надежность по сравнению с SQL-решениями, а также отсутствие жесткой стандартизации, что означает вариацию способа взаимодействия при переходе от базы к базе [1; 5—7]. Ниже приведены основные типы NoSQL СУБД и способы взаимодействия с ними.

- **Графовые СУБД**. Предназначены для обработки данных, которые легче представить в виде графа, чем в виде таблиц. Используются в основном при создании транспортных маршрутов, семантических сетей и социальных сетей. Примеры СУБД: Neo4j (самая распространённая), OrientDB, InfiniteGraph. Запросы к Neo4j в основном выполняются на декларативном языке Cypher [1; 8].
- **Документо-ориентированные СУБД**. Представляют информацию в форме иерархических структур данных. Единицей представления данных является документ, который имеет набор индивидуальных атрибутов. Используются в издательском деле, документальном поиске, системе управления содержимым. Примеры СУБД: MongoDB (самая распространённая), CouchDB, ElasticSearch. Для взаимодействия с MongoDB используется JSON-подобный язык запросов [1; 8].
- **Хранилища пар «ключ-значние»**. Представляют собой хэш-таблицу, содержащую значения и соответствующие им ключи. Примеры использования: хранение изображений, использование в качестве кэша объектов. Примеры СУБД: Redis (самая распространенная), Memcached, Riak, MemcacheDB. Запросы выполняются через вызовы API на языках программирования. Reddis поддерживает скрипты на языке Lua [1; 9].
- **Колоночные СУБД**. В отличие от реляционных БД, хранят данные не строками, а в колонках. Это позволяет применять эффективное сжатие, так как каждая колонка содержит только один тип данных. Примеры использования: веб-индексирование, задачи с большими данными, но с пониженными требованиями к согласованности. Примеры

СУБД: Cassandra(самая распространенная), HBase, Vertica. Запросы к Cassandra выполняются через декларативный язык CQL (Cassandra Query Language) и вызовы API [1; 9].

Мультимодельные базы данных. Также существуют мультимодельные базы данных, которые включают две или более из представленных выше категорий.

Приведённая классификация освещает большинство СУБД, использующихся в современных IT-проектах.

1.1.2 Способы взаимодействия технических специалистов с базами данных

Основываясь на представленных моделях данных, можно выделить несколько основных парадигм взаимодействия разработчика или программного обеспечения с СУБД для выполнения запросов и манипуляции данными. Эта классификация важна для понимания, какие навыки требуются от пользователя для работы с той или иной системой.

С помощью декларативных языков запросов. Это наиболее распространённый способ взаимодействия, при котором пользователь описывает, какие данные он хочет получить, но не указывает, как именно системе следует их извлекать. Система сама оптимизирует и выполняет запрос. К примерам таких языков можно отнести SQL для запросов к реляционным БД, Cypher для взаимодействия с Neo4j, CQL для Cassandra.

С помощью документо-подобных языков запросов. Взаимодействие происходит с помощью объектов, структура которых напоминает структуру данных в самой базе. Ярким примером является язык запросов в MongoDB, где для поиска, фильтрации и модификации данных используются JSON-подобные документы. Хотя этот подход мощен и гибок, он менее стандартизирован, чем SQL, и тесно связан со структурой хранимых документов, поэтому данный способ может показаться достаточно сложным даже для подкованного пользователя, решившего применить документо-ориентированную СУБД в своём проекте [10].

Программные интерфейсы (API). В этой модели взаимодействие с базой данных происходит не через специальный язык запросов, а через вызовы функций (API) из кода приложения, написанного на языках программирования вроде Python, Java, C# и других. Этот подход характерен как основной для многих NoSQL-хранилищ. Он достаточно прост и предоставляет разработчикам полный контроль, но требует навыков программирования [9; 10].

Специализированные и графические интерфейсы. Некоторые системы, такие как Active Directory или Реестр Windows, используют узкоспециализированные утилиты командной строки или графические интерфейсы для взаимодействия. Также для многих популярных

СУБД существуют административные панели (например, pgAdmin для PostgreSQL), которые предоставляют графический интерфейс для выполнения запросов и управления базой данных. Часто они являются надстройкой над основными способами взаимодействия, и поэтому пользователи должны иметь опыт работы с основным языком запросов к используемой базе данных.

1.1.3 Выбор объекта исследования

Проведенный анализ показывает, что мир баз данных разнообразен как по моделям хранения данных, так и по способам взаимодействия с ними. Для нетехнического специалиста, желающего извлечь информацию, это многообразие создает значительные трудности. Декларативные языки запросов требуют опыта работы с ними. Документо-подобные подходы и API требуют специальных знаний в области программирования и понимания конкретной СУБД. Графические интерфейсы служат скорее удобной средой для технических специалистов, чем полноценным решением для пользователей, не владеющих языком запросов.

На этом фоне исследовательский интерес представляют пользовательские интерфейсы, адаптированные для людей без технических навыков. Они позволяют не только программистам, но и каждому пользователю общаться с системой, не проходя специализированную подготовку в части формальных языков.

Таким образом, в качестве центральной задачи данного исследования выбирается обзор существующих пользовательских интерфейсов, которые предлагают альтернативные способы взаимодействия с базами данных.

1.2 Сравнительный анализ интерфейсов баз данных для нетехнических пользователей

Аннотация. *В данном разделе проводится сравнение подходов к разработке интерфейсов баз данных, ориентированных на нетехнических пользователей: запросов по примеру (QBE), визуальных систем запросов (VQS) и естественно-языковых интерфейсов (NLIDB). Для каждого подхода выделяются ключевые особенности, преимущества и недостатки с точки зрения доступности, гибкости и требований к пользователю. На основе анализа выделяется наиболее перспективный вид интерфейса для дальнейших исследований и разработки.*

1.2.1 Запрос по примеру (Query By Example)

Query By Example (QBE) — визуальный язык запросов, который был разработан параллельно с SQL с целью предоставить высокоуровневый и унифицированный интерфейс для взаимодействия с реляционными базами данных. Он был изначально ориентирован на пользователей, практически не имевших опыта в разработке, и предполагал ввод данных в пустые схематические таблицы на экране путём заполнения форм [8; 11].

Основным преимуществом QBE является его интуитивность и низкий порог вхождения. Пользователю не требуется изучать сложный синтаксис, так как запросы формируются путем заполнения таблиц-шаблонов, предоставляя пример желаемого результата [11; 12]. Это позволяет постепенно переходить от простых запросов к более сложным, а психологические исследования показали, что непрограммисты могут освоить язык менее чем за три часа [11].

Однако у данного подхода есть и существенные недостатки. Во-первых, как уже отмечалось, QBE работает только с реляционными базами данных. Во-вторых, QBE не позволяет формировать сложные запросы, например, с подзапросами или объединениями, что делает его абсолютно бесполезным в профессиональной деятельности. Некоторые источники отмечают, что обучение QBE вместо SQL является нецелесообразным, поскольку этот язык не используется в профессиональном программировании и не способствует развитию навыков работы с базами данных [13]. Тем не менее, QBE получил широкое признание как концепция и лёг в основу таких популярных инструментов, как Microsoft Access [8].

1.2.2 Визуальные системы запросов (Visual Query Systems)

Для решения проблемы доступа к данным для нетехнических пользователей были разработаны визуальные системы запросов (Visual Query Systems, VQS), которые предоставляют интуитивно понятные интерфейсы, не требующие знания формальных языков [12; 14; 15].

Ключевой особенностью визуальных конструкторов является замена текстового ввода на прямое манипулирование визуальными элементами [14]. Пользователи взаимодействуют с графическими представлениями данных — формами, диаграммами или иконками —, а система автоматически преобразует эти действия в формальный запрос [16; 17].

Существует несколько подходов к визуализации:

- **На основе форм (Form-based):** Пользователь заполняет таблицы-шаблоны, предоставляя пример желаемого результата. Классическим примером является Query-By-Example (QBE) [12].
- **На основе диаграмм (Diagram-based):** Запросы строятся путем взаимодействия со

схемами, такими как диаграммы «сущность-связь» (ER). Например, система VIREX генерирует интерактивную ER-диаграмму, на которой пользователи могут выбирать таблицы и атрибуты с помощью щелчков мыши [18]. Аналогично, Visual Query Builder в DBeaver позволяет перетаскивать таблицы и выбирать столбцы, автоматически генерируя SQL-код. Современные реализации, такие как DBpedia Visualizer, позволяют строить визуальный граф для создания SPARQL-запросов [16].

– **На основе иконок (Icon-based):** Объекты и операции представляются в виде иконок, что особенно полезно для пользователей, не знакомых даже с концепцией схемы базы данных [12].

Эти системы ориентированы на так называемых «случайных пользователей» (casual users), которые взаимодействуют с базами данных нерегулярно и не имеют времени на обучение [12; 14]. Несмотря на свои преимущества, визуальные интерфейсы могут иметь ограничения при работе с большими и сложными наборами данных, и от пользователя все еще может потребоваться некоторое понимание структуры схемы [15].

1.2.3 Естественно-языковые интерфейсы (Natural language interfaces)

В отличие от визуальных конструкторов, интерфейсы на естественном языке (Natural language interfaces to the databases, NLIDB) стремятся полностью устранить барьер между пользователем и базой данных, позволяя формулировать запросы на обычном разговорном языке, например, русском или английском [3; 19; 20]. Основная цель NLIDB — предоставить нетехническим пользователям возможность напрямую взаимодействовать с базами данных, что делает их незаменимым инструментом для широкого круга специалистов [21; 22].

Важно понимать, что в основе работы таких интерфейсов лежат специальные алгоритмы или модели, обеспечивающие функциональность. Задача к конструированию таких моделей в машинном обучении называется «Text-to-SQL». Также возможны формулировки «NL-to-SQL», «Text2SQL», «NL2SQL».

Исследования в этой области ведутся с 1970-х годов, начиная с таких систем, как LUNAR [23—25]. Темп публикации научных статей в этой области на сегодняшний день довольно интенсивный, что свидетельствует о высоком интересе исследовательского сообщества к проблеме построения эффективных моделей, лежащих в основе NLIDB, и постоянном совершенствовании методов преобразования естественного языка в формальные запросы. Основные подходы к реализации функционирования NLIDB можно классифицировать следующим образом:

– **Подходы на основе правил и синтаксиса.** Используют лингвистический анализ, включая синтаксические парсеры и онтологии, для преобразования запроса в SQL. Эти методы точны, но часто ограничены конкретной предметной областью [15; 19; 26—28].
– **Подходы на основе моделей глубокое обучения.** Они уже рассматривают задачу из области машинного обучения как перевод с естественного языка на SQL («Text-to-SQL»). Здесь применяются нейросетевые архитектуры, такие как LSTM и Transformer, обучаемые на датасетах вида множества пар «запрос-SQL» [29; 30]. Они более гибки, чем системы на основе правил, однако часто менее точны.
– **Большие языковые модели (LLM).** Этот этап стал прорывным в развитии NLIDB. Модели вроде GPT способны генерировать SQL-запросы с высокой точностью, не требуя при этом детального описания контекста. Модели могут вовсе не требовать знать о конкретной задаче (Zero-Shot), могут потребовать знать небольшое количество примеров (Few-shot), а также могут использовать техники, основанные на рассуждениях, например Chain-of-Thought [25; 31].
– **Системы, использующие логи запросов.** Такие подходы, как Templar, анализируют логи ранее выполненных SQL-запросов для уточнения сопоставления ключевых слов и вывода правильных объединений таблиц, что повышает точность системы [23; 32].

Уникальность NLIDB по сравнению с VQB заключается в полном абстрагировании пользователя от структуры базы данных. Если VQB требует от пользователя понимания, какие таблицы и поля существуют, и как их визуально соединить для получения результата, то NLIDB позволяет формулировать запрос так, как если бы он был адресован человеку. Главное преимущество NLIDB — это его интуитивность и доступность для самой широкой аудитории [33]. Однако основной недостаток кроется в присущей естественному языку неоднозначности (полисемия, синонимия, идиомы), что создает значительные трудности для точной интерпретации запросов и требует сложных алгоритмов для их разрешения [23; 34].

1.2.4 Результаты сравнения и выявление наиболее перспективного метода

Проанализировав три основных подхода к взаимодействию с базами данных для нетехнических пользователей — Query By Example (QBE), визуальные конструкторы запросов (VQB) и естественно-языковые интерфейсы (NLIDB) — можно выявить их ключевые различия, преимущества и недостатки. Результаты сравнения представлены в виде таблицы (см. табл. 1.1).

Проведенный сравнительный анализ показывает, что каждый из рассмотренных типов

Таблица 1.1 – Сравнительная таблица интерфейсов, адаптированных для нетехнических пользователей

№	Критерий сравнения	QBE	VQS	NLIDB
1	Уровень требований к нетехническому пользователю	**Средний.** Понимание структуры таблиц и базовых концепций запросов	**Средний.** Понимание схемы БД	**Низкий.** Имитирует естественное общение с компьютером
2	Уровень абстракции	**Низкий.** Пользователь напрямую работает с представлением таблиц	**Средний.** Пользователь работает с диаграммами, но нужно помнить о связях таблиц в базе данных	**Высокий.** Пользователь работает только с полем ввода текста
3	Гибкость запросов	**Низкая.** Подзапросы и объединения формировать затруднительно	**Средняя.** Позволяет создавать сложные запросы, но в рамках предложенного визуального инструментария	**Потенциально высокая.** Теоретически ограничена только способностью системы понять естественный язык.
4	Основное преимущество	Простота и наглядность для простых запросов	Наглядность связей между данными, сниженная вероятность синтаксических ошибок	Максимальная доступность и интуитивность. Низкий порог входа
5	Основной недостаток	Слишком ограниченная функциональность и непригодность в профессиональной деятельности	Требует от пользователя предварительного изучения и понимания схемы конкретной базы данных	Требует аккуратной формулировки из-за неоднозначности естественного языка

интерфейсов предоставляет удобства при взаимодействии человека с базой данных. В этом контексте естественно-языковые интерфейсы (NLIDB) нацелены на устранение самого главного барьера — необходимости понимать, как организованы данные. NLIDB позволяют пользователям формулировать свои потребности наиболее естественным для человека способом, фокусируясь на содержании вопроса, а не на его технической форме.

Учитывая вышесказанное, дальнейшая работа будет сконцентрирована на исследовании и разработке систем, основанных на естественно-языковом интерфейсе, как наиболее передовом и многообещающем подходе к взаимодействию с базами данных.

1.3 Обзорный анализ систем, решающих задачу Text-to-SQL

Аннотация. *В данном разделе проводится обзор истории и современного состояния систем, решающих задачу преобразования естественного языка в SQL-запросы (Text-to-SQL), лежащих в основе NLIDB. Рассматриваются ключевые этапы развития моделей: от ранних систем на правилах и синтаксическом анализе до современных решений на основе больших языковых моделей (LLM). Анализируются основные методологии построения современных систем, используемые датасеты и метрики оценки качества. На основании сравнительного анализа обосновывается выбор технологического ядра для реализации собственного NLIDB.*

NLIDB под своей оболочкой скрывают сложные модели и алгоритмы, основная задача которых — преобразование неструктурированного человеческого языка в формальные и исполняемые SQL-запросы. Таким образом, задача создания эффективного NLIDB сводится к задаче разработки и выбора наиболее подходящей модели трансляции Text-to-SQL, способной точно интерпретировать намерения пользователя и корректно взаимодействовать со схемой конкретной базы данных.

1.3.1 Краткая история развития моделей Text-to-SQL

Эволюция моделей для задачи Text-to-SQL отражает общие тенденции в области обработки естественного языка и машинного обучения. Можно выделить три ключевых этапа, каждый из которых знаменовал собой качественный скачок в развитии технологии. Эволюцию подходов также можно представить в виде иллюстрации (см. рис. 1.1).

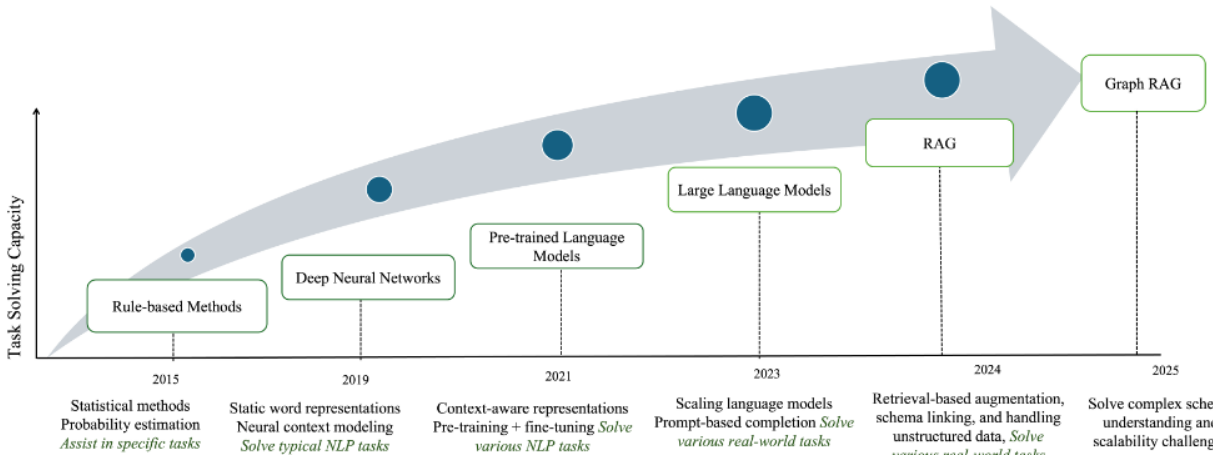


Рисунок 1.1 – Краткая история развития моделей Text-to-SQL

Ранние системы и подходы, основанные на правилах (1970-е – 2000-е). Первые NLIDB, появившиеся еще в 1970-х годах, такие как LUNAR и более поздние системы, основанные на синтаксическом разборе и онтологиях, полностью полагались на вручную созданные грамматики, словари и наборы правил. Эти системы были «зажатыми», имели ограниченный словарный запас и требовали значительных усилий для адаптации к каждой новой предметной области. Их главным недостатком была низкая гибкость и неспособность справляться с лингвистической вариативностью и неоднозначностью естественного языка [23; 28; 29; 35; 36].

Появление машинного обучения и нейронных сетей (2010-е). Первый прорыв произошел с развитием глубокого обучения. Исследователи начали рассматривать задачу Text-to-SQL как задачу машинного перевода. Появились модели на основе рекуррентных нейронных сетей (RNN), в частности, архитектуры Sequence-to-Sequence (Seq2Seq) с механизмом внимания (attention), которые обучались на размеченных данных [29; 30]. Катализатором для бурного развития этого направления стало появление крупных аннотированных наборов данных, таких как WikiSQL и, в особенности, Spider. Это позволило создавать и объективно сравнивать более сложные и точные нейросетевые модели, такие как TypeSQL и SyntaxSQLNet, которые учитывали структуру схемы базы данных и показывали значительно лучшие результаты по сравнению с системами на правилах [23; 30]. Однако по-прежнему оставались проблемы, такие как обработка вложенных запросов, обобщение на конкретную базу данных и сопоставление неоднозначного естественного языка со структурированным SQL [36].

Эпоха больших языковых моделей (LLM) (2020-е – настоящее время). Последние несколько лет ознаменовались доминированием больших языковых моделей (LLM), таких как модели GPT-4, PaLM и LLaMA, что привело к очередной смене парадигмы. LLM лучше справляются в задачах, требующих понимания и генерации человекоподобного текста, часто без дополнительной настройки. Они лучше улавливают сложные взаимосвязи между естественным языком и схемами базы данных [25; 36]. Однако, несмотря на свой потенциал, эти подходы обычно сталкиваются с несколькими проблемами: галлюцинации моделей, неконтролируемый результат, высокое требование к вычислительным ресурсам.

С появлением LLM, разработка новых решений для задачи Text-to-SQL показала заметную тенденцию к росту с сентября 2023 по октябрь 2024 года, что свидетельствует о значительном интересе со стороны сообщества. Количество публикаций достигло пика с июля по октябрь 2024 года, составив примерно 39% от общего их числа на тему LLM-основанных методов решения Text-to-SQL. Ожидается, что текущие достижения в этой области приведут

к последующим инновациям в методах преобразования текста в SQL. Эти достижения позиционируют LLM как основу современных систем преобразования естественного языка в SQL, устраняя разрыв между разговорными и формальными языками с растущей точностью и эффективностью [37].

1.3.2 Текущее положение в исследованиях подходов к решению задачи Text-to-SQL с помощью моделей, основанных на LLM

Как упоминается в источнике [37], современные научные статьи в области Text-to-SQL можно классифицировать следующим образом:

- Методологии
  - Предобработка (Pre-Processing)
  - Обучение в контексте (In-context learning)
  - Тонкая настройка (Fine-Tuning)
  - Постобработка (Post-Processing)
- Наборы данных (Datasets)
- Метрики применения (Evaluation Metrics)

Рассмотрим каждый вопрос, рассматриваемый в академической литературе по-отдельности.

Методологии

Как уже было отмечено, современные подходы к решению задачи Text-to-SQL в основном опираются на большие языковые модели. Однако сами по себе LLM не являются готовым решением и требуют применения целого ряда инженерных практик и методологий для достижения высокой точности и эффективности. Исследователи [37] выделяют четыре основных парадигмы, в рамках которых ведутся разработки и улучшения LLM-основанных систем: предварительная обработка (Pre-Processing), обучение в контексте (In-context learning), тонкая настройка (Fine-Tuning) и постобработка (Post-Processing), что охватывает весь жизненный цикл обработки языка. Это можно продемонстрировать с иллюстрации (см. рис. 1.2).

Предварительная обработка (Pre-Processing)

Этот этап является ключевым для подготовки и структурирования входных данных, что напрямую влияет на точность генерируемого SQL-запроса. Основная задача здесь — максимально снизить неоднозначность и предоставить модели всю необходимую для работы информацию. Одним из центральных методов является *связывание со схемой* (Schema Linking),

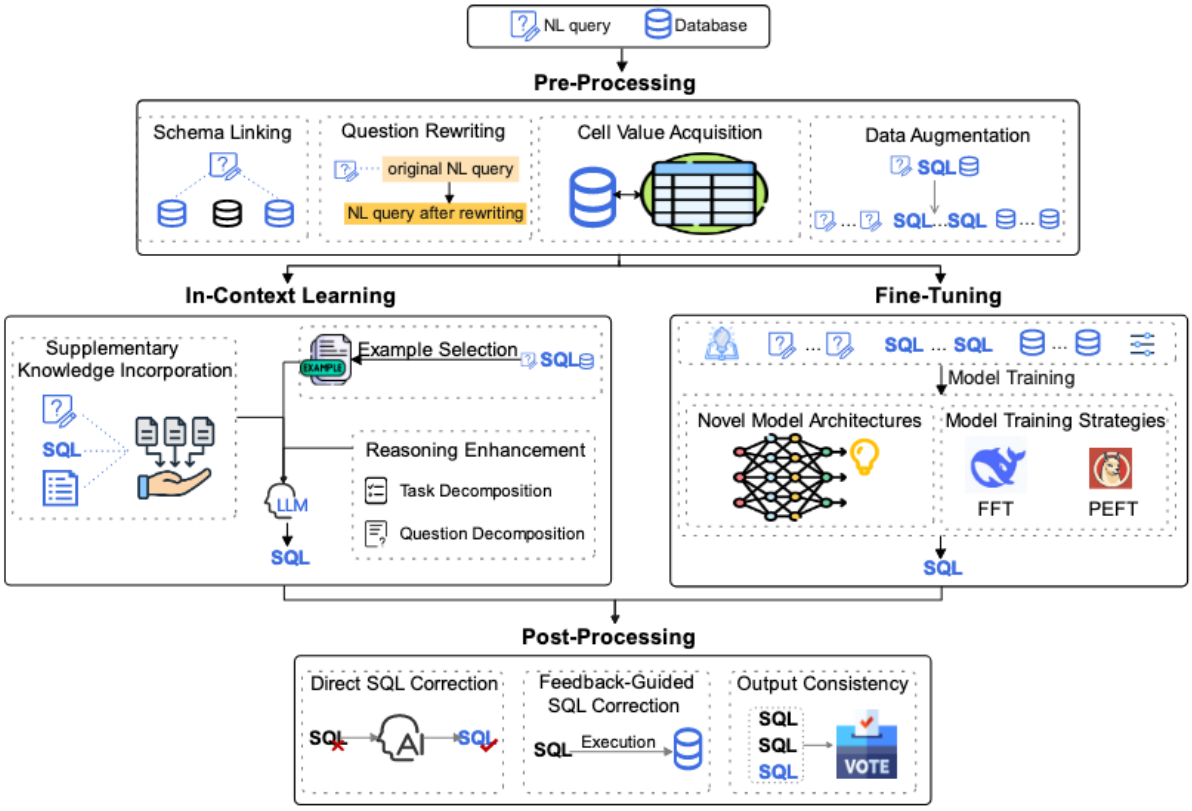


Рисунок 1.2 – Блок-схема типового конвейера обработки запроса в современной LLM-системе.

в ходе которого система сопоставляет слова и фразы из естественного языка с конкретными таблицами, столбцами и их отношениями в базе данных [36; 37]. Другой важный метод — *извлечение значений ячеек* (Cell Value Acquisition), который позволяет обогатить контекст, добавив в него релевантные примеры значений из базы данных, что особенно полезно, когда в запросе пользователя отсутствуют детали, необходимые для точной генерации SQL [37]. Наконец, для борьбы с присущей естественному языку двусмысленностью применяется *переписывание вопроса* (Question Rewriting), где модель сначала переформулирует исходный запрос в более четкий и однозначный, прежде чем приступать к генерации кода.

Обучение в контексте (In-Context Learning, ICL)

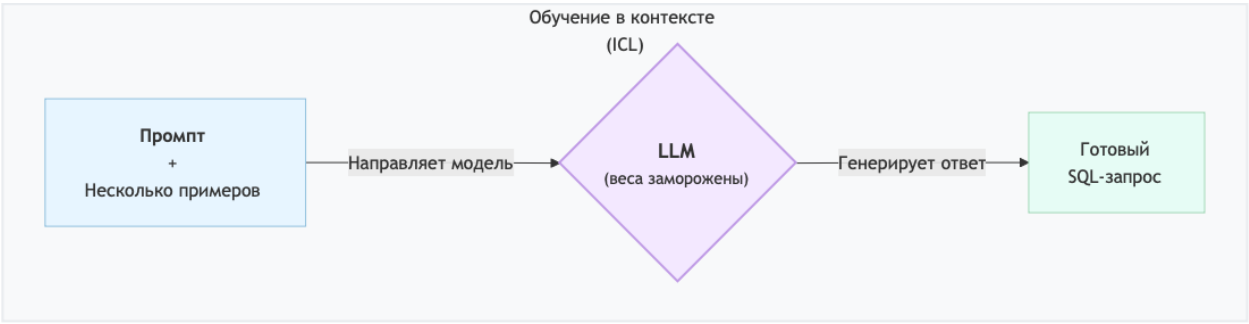
Данная парадигма сосредоточена на инженерии промптов (prompt engineering) — искусстве составления такого запроса к модели, который бы наиболее эффективно направлял ее на решение конкретной задачи без необходимости изменять веса самой модели. Основные подходы здесь — это *Zero-shot* (без примеров) и *Few-shot* (с малым количеством примерам) [25]. В последнем случае ключевую роль играет стратегия отбора примеров. Для решения нетривиальных задач применяется *усиление рассуждений* (Reasoning Enhancement), где использу-

ются такие техники, как «цепочка мыслей» (Chain-of-Thought, CoT) [25; 37].

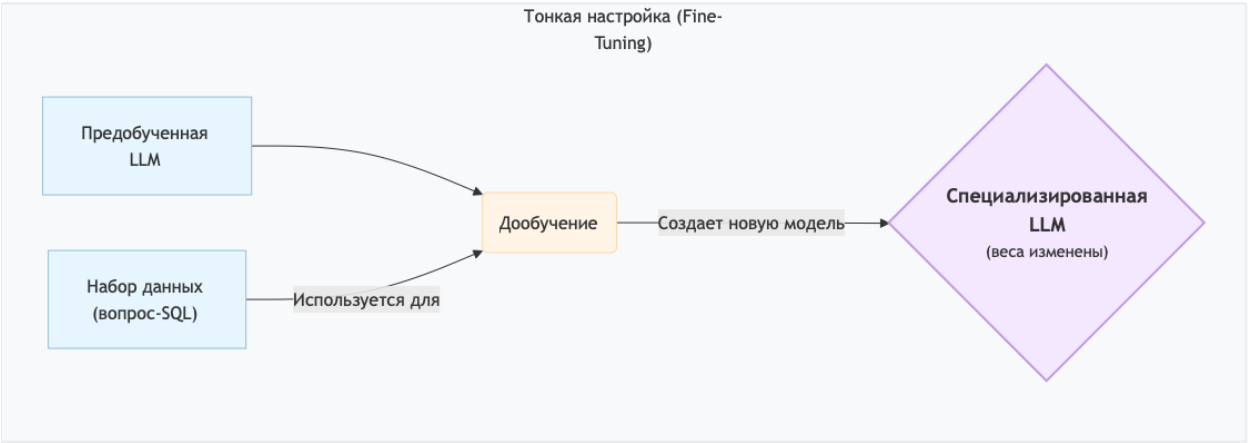
Тонкая настройка (Fine-Tuning, FT)

В отличие от ICL, этот метод предполагает изменение весов предобученной модели путем ее дообучения на специфичном для задачи наборе данных. Это позволяет создавать более специализированные и точные модели, особенно когда используются LLM с открытым исходным кодом (open-source). Выделяют два основных подхода: *полная тонкая настройка* (Full Fine-Tuning, FFT), когда обновляются все параметры модели, и *параметрически-эффективная тонкая настройка* (Parameter-Efficient Fine-Tuning, PEFT), при которой изменяется лишь небольшая часть весов. Второй подход позволяет значительно сократить вычислительные ресурсы, необходимые для дообучения [37].

Отличия в методологиях ICL и FT можно визуализировать с помощью иллюстрации (см. рис. 1.3).



(а) Процесс обучения в контексте (ICL)



(в) Процесс тонкой настройки (FT)

Рисунок 1.3 – Концептуальная схема, сравнивающая In-Context Learning и Fine-Tuning.

Постобработка (Post-Processing)

Даже самые продвинутые модели могут генерировать SQL-запросы с синтаксическими или логическими ошибками. Парадигма постобработки нацелена на их исправление и верификацию. Один из подходов — *прямая коррекция SQL*, когда модель сама или по заготовленным правилам ищет и исправляет ошибки в сгенерированном коде. Более продвинутый метод — *коррекция на основе обратной связи* (Feedback-Guided SQL Correction). В этом случае сгенерированный SQL-запрос выполняется в базе данных, и если возникает ошибка, ее текст вместе с исходным запросом подается в модель для повторной генерации [37]. Такой итеративный подход реализован во многих современных системах, например, в компоненте Refiner системы XiYan-SQL [38] или в агентных системах, таких как MAC-SQL [25]. Наконец, для повышения надежности используется *проверка согласованности вывода* (Output Consistency): система генерирует несколько вариантов SQL-запроса, а затем выбирает наиболее вероятный или наиболее согласованный из них, что является центральной идеей ансамблевого подхода в XiYan-SQL [37; 38].

Наборы данных (Datasets)

Для обучения и объективной оценки моделей Text-to-SQL необходимы качественные и разнообразные наборы данных. Их развитие шло параллельно с эволюцией самих моделей, от небольших и узкоспециализированных до масштабных и сложных. Современные датасеты, как отмечается в [37], можно разделить на несколько категорий.

Во-первых, это *однодоменные* (single-domain) наборы данных, такие как ATIS или GeoQuery, которые ориентированы на конкретную предметную область и используются для оценки производительности моделей в узких задачах. Во-вторых, что более актуально для современных исследований, это *междоменные* (cross-domain) датасеты. Они требуют от моделей способности к обобщению, то есть умения работать с базами данных, схемы которых не были представлены на этапе обучения. Ключевым датасетом в этой категории является Spider [23; 30], который стал отраслевым стандартом для оценки сложности и точности. Более поздние разработки, такие как BIRD [37], еще больше усложнили задачу, добавив требования к обработке «грязных» данных и пониманию их семантики.

Помимо этого, выделяют и более специализированные категории [37]: *контекстно-зависимые* (context-dependent) для оценки диалоговых систем, *ориентированные на робастность* (robustness-centered), которые проверяют устойчивость моделей к синонимам и другим вариациям в запросах и *межъязыковые* (cross-lingual) для оценки качества работы с неанглоязычными за-

просами. Именно наличие таких комплексных и многоаспектных датасетов стимулирует исследователей к разработке все более совершенных моделей, способных работать в условиях, приближенных к реальным [36; 38].

Метрики применения (Evaluation Metrics)

Для количественной оценки качества работы Text-to-SQL систем используются две основные группы метрик [37].

Первая группа — это метрики, основанные на *сопоставлении содержимого* (Content Matching-based). Простейшей из них является *точность полного совпадения строк* (Exact Matching Accuracy), которая требует, чтобы сгенерированный SQL-запрос был посимвольно идентичен эталонному. Это очень строгая метрика, которая не учитывает, что один и тот же результат можно получить с помощью синтаксически разных, но семантически эквивалентных запросов. Более гибкий подход предлагает *точность совпадения компонентов* (Component-Match Accuracy), используемая в бенчмарке Spider. Она разбивает SQL-запрос на составные части (SELECT, WHERE, GROUP BY и т.д.) и оценивает корректность каждой из них по-отдельности.

Вторая, и более важная с практической точки зрения, группа — это метрики, основанные на *результате выполнения* (Execution Result-based). Ключевой метрикой здесь является *точность выполнения* (Execution Accuracy, EX). Она оценивает не синтаксическую схожесть запросов, а идентичность результатов, полученных при выполнении сгенерированного и эталонного SQL-запросов на базе данных. Именно эта метрика является основной в большинстве современных исследований и бенчмарках, включая BIRD и Spider, так как она напрямую отражает способность системы дать пользователю правильный ответ [37; 38].

1.3.3 Выбор и обоснование базовой системы для реализации NLIDB

Для практической реализации собственного естественно-языкового интерфейса в рамках УИР необходимо выбрать конкретную систему или фреймворк, который будет служить технологическим ядром. Выбор должен основываться на объективных показателях производительности, архитектурном соответствии современным тенденциям и практической применимости для решения поставленных задач.

В качестве такой системы была выбрана XiYan-SQL — современный ансамблевый фреймворк, разработанный исследователями из Alibaba Group[38]. Данное решение представляет собой комплексный подход, объединяющий несколько передовых методологий, рассмотрен-

ных ранее.

Выбор в пользу XiYan-SQL обусловлен несколькими причинами:

Лидирующие позиции в ключевых бенчмарках. На момент проведения исследования XiYan-SQL занимает первые места в наиболее авторитетных и сложных междоменных бенчмарках, таких как Spider и BIRD. Согласно данным с ресурса «Papers with Code» [1], данный фреймворк демонстрирует лучшие результаты по метрике Execution Accuracy, опережая другие известные системы. Графически его лидирование может быть проиллюстрировано на графиках (см. рис. А.1). Это является объективным подтверждением его высокой точности и способности генерировать корректные SQL-запросы для разнообразных баз данных.

Современная гибридная архитектура. С архитектурной точки зрения, XiYan-SQL реализует передовой ансамблевый подход, который сочетает сильные стороны как тонкой настройки, так и обучения в контексте. Система использует несколько генераторов — часть из них дообучена для генерации высокоточных и стилистически разнообразных запросов, а другая часть использует ICL для повышения гибкости. Такой подход позволяет достичь баланса между точностью, разнообразием и способностью справляться со сложными запросами. Кроме того, в фреймворк встроены механизмы постобработки (Refiner) и отбора лучших запросов, что полностью соответствует передовым подходам, описанным в академической литературе [37; 38].

Практическая доступность и применимость. Немаловажным фактором, особенно в рамках академической работы, является практическая доступность фреймворка. Разработчики XiYan-SQL предоставили открытый доступ к исходному коду своей системы через репозиторий на GitHub [2]. Это позволяет не просто теоретически изучить подход, но и интегрировать его в собственный проект, адаптировать и использовать в качестве основы для дальнейших разработок. Это значительно снижает порог вхождения и позволяет сконцентрироваться на задачах более высокого уровня, таких как разработка пользовательского интерфейса и логики работы веб-сервиса.

Соответствие задачам исследования. На начальном этапе данная курсовая работа предполагает создание NLIDB для работы с отдельными, не связанными между собой таблицами. Иначе говоря, с реляционными базами данных, состоящих из единственных таблиц. Хотя XiYan-SQL способен обрабатывать и более сложные межтабличные запросы, его фундаментальная точность и робастность являются отличной основой и для более простых сценариев.

При выборе были рассмотрены и другие системы, представленные в таблицах лидеров

[1] URL: `https://paperswithcode.com`

[2] URL: `https://github.com/XGenerationLab/xiyan_mcp_server`

отраслевых бенчмарков (см. рис. А.1). Однако многие из них либо показывают более низкую производительность на комплексных датасетах, либо представляют собой более ранние архитектурные решения, либо не предоставляют готового к внедрению открытого кода, что делает их использование в рамках данного проекта нецелесообразным.

Важно отметить, что данный выбор касается именно технологического ядра для преобразования текста в SQL. Вопрос о новизне и актуальности создания нового веб-сервиса с подобным функционалом требует отдельного анализа существующих коммерческих и открытых продуктов, который будет проведен на следующем этапе работы. Тем не менее, для решения центральной задачи — качественной трансляции естественного языка в SQL — фреймворк XiYan-SQL на сегодняшний день является наиболее мощным, современным и подходящим.

1.4 Обзор существующих веб-сервисов и обоснование актуальности разработки собственного веб-сервиса

Аннотация. *В данном разделе представлен обзор существующих веб-сервисов, реализующих естественно-языковой интерфейс к базам данных (NLIDB), включая как проприетарные, так и решения с открытым исходным кодом. На основе сравнительного анализа выявляются основные ограничения современных решений и обосновывается актуальность разработки собственного веб-сервиса с NLIDB.*

После выбора передового технологического ядра в лице фреймворка XiYan-SQL, необходимо провести анализ существующих на рынке веб-сервисов, предоставляющих функциональность естественно-языкового интерфейса к базам данных (NLIDB). Этот анализ позволит определить, существуют ли готовые решения, полностью удовлетворяющие потребностям целевой аудитории, и выявить незанятые ниши, что в свою очередь обоснует актуальность и научную новизну данного проекта. Для этого был проведен обзор нескольких популярных веб-сервисов и open-source решений.

1.4.1 Обзор существующих веб-сервисов с интеграцией ИИ для упрощения взаимодействия с базами данных

WrenAI. Это мощная платформа, которая существует в двух вариантах: облачный коммерческий сервис (Wren Cloud) и решение с открытым исходным кодом для самостоятельного развертывания (WrenAI Open Source) [3]. Система позволяет создавать как собственные базы

[3]URL: `https://docs.getwren.ai/oss/overview/introduction`

данных (CSV), так и напрямую подключаться к широкому спектру реляционных баз данных. Ключевым недостатком для нетехнического пользователя является необходимость локальной установки и настройки open-source версии, что требует определенных технических навыков. Кроме того, система в своей основе полагается на API от OpenAI, что создает барьер для пользователей из России и повышает зависимость от стороннего платного сервиса. Пользовательский интерфейс, хоть и функциональный, может показаться перегруженным для базовых задач.

SQLAI. Данный сервис позиционируется как многофункциональный инструмент, ориентированный не только на конечных пользователей, но и на разработчиков [4]. Он предлагает генерацию SQL-запросов, их оптимизацию и отладку. Важным преимуществом является поддержка не только реляционных, но и некоторых NoSQL баз данных. Однако, это полностью коммерческий продукт с закрытым исходным кодом, что делает невозможным его кастомизацию и изучение внутреннего устройства. Англоязычный интерфейс и широкий, но избыточный для нетехнического специалиста функционал, делают его менее подходящим для целей нашего исследования.

ai2sql. Этот сервис сфокусирован на простоте и предлагает интуитивно понятный интерфейс для преобразования запросов на естественном языке в SQL [5]. Он поддерживает работу с загружаемыми CSV-файлами, что является удобным сценарием для пользователей, не имеющих прямого доступа к базам данных. Как и предыдущий аналог, ai2sql является проприетарным решением с закрытым исходным кодом и англоязычным интерфейсом, что ограничивает его применимость и прозрачность.

SQLchat. SQLchat — это проект с открытым исходным кодом, который предоставляет веб-интерфейс для «общения» с базой данных [6]. Его основное преимущество — открытость и простота. Однако, он не поддерживает загрузку данных из файлов (например, CSV) и требует от пользователя прямого подключения к уже существующей и настроенной базе данных, что может быть барьером для целевой аудитории.

AI SQL Gen. Данный проект является типичным примером простой системы-обертки над GPT-4 [7], что наглядно демонстрирует его архитектура (см. рис. 1.4). Он требует от пользователя наличия собственного ключа OpenAI API, запускается локально и не обновлялся в течение долгого времени. Несмотря на простоту, такие системы полностью зависят от стороннего сервиса и не предлагают никакой дополнительной ценности в виде продвинутого

[4]URL: `https://www.sqlai.ai/documentation/introduction/introduction`
[5]URL: `https://ai2sql.io`
[6]URL: `https://www.sqlchat.ai`
[7]URL: `https://github.com/marblexyz/aisqlgen`

связывания со схемой или поддержки различных моделей.

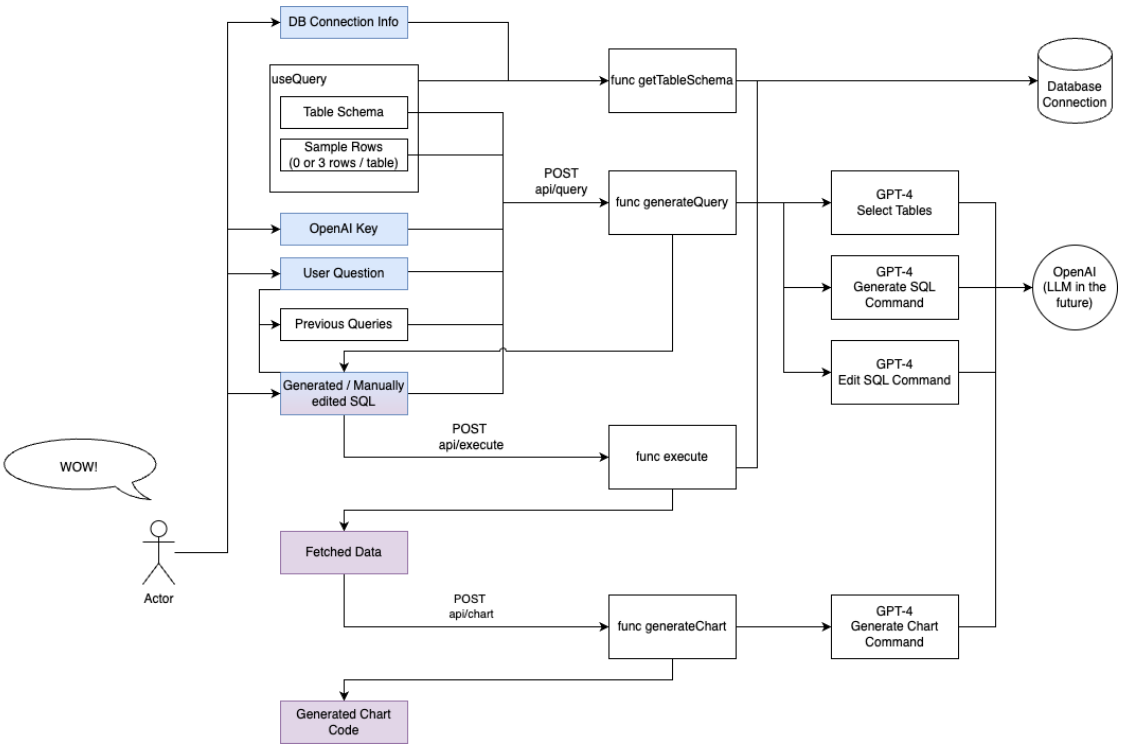


Рисунок 1.4 – Архитектура типичной системы-обертки (на примере AI SQL Gen)

1.4.2 Сравнительный анализ и обоснование актуальности собственной разработки

Проведенный обзор позволяет систематизировать существующие решения и выявить их ключевые характеристики, которые представлены в виде сравнительной таблицы (см. табл. 1.2).

Анализ таблицы и детальный обзор сервисов позволяет сделать несколько ключевых выводов и выявить недостатки у альтернативых решений, что и обосновывает актуальность в собственной разработке веб-сервиса в рамках УИР.

- **Проблема доступности**. На рынке наблюдается четкое разделение: либо платные, закрытые веб-сервисы, либо open-source проекты, требующие от пользователя технических навыков для локального развертывания. Практически отсутствует ниша веб-сервиса, который был бы одновременно с открытым исходным кодом, бесплатным для конечного пользователя и доступным онлайн без необходимости установки.
- **Зависимость от проприетарных API**. Большинство доступных решений, включая open-source, жестко привязаны к API коммерческих LLM, в первую очередь — OpenAI. Это создает существенный барьер для пользователей из России (из-за сложностей с оплатой и регистрацией) и повышает риски, связанные с конфиденциальностью данных, передаваемых на сторонние серверы.

Таблица 1.2 – Сравнительная таблица существующих NLIDB-сервисов

Сервис	**Модель развертывания**	**Подключение данных**	**Модель монетизации и API**
WrenAI	Open-source (локальный) или проприетарный веб-сервис Cloud	CSV, прямое подключение к БД	Требует ключ OpenAI API при использовании open-source решения
SQLAI.ai	Проприетарный веб-сервис	Прямое подключение к БД (SQL & NoSQL)	Коммерческий, закрытый исходный код
ai2sql.io	Проприетарный веб-сервис	Загрузка CSV, прямое подключение	Коммерческий, закрытый исходный код
SQLchat	Open-source (локальный)	Только прямое подключение к БД	Бесплатный, с открытым кодом, требует ключ OpenAI API
AI SQL Gen	Open-source (локальный)	Только прямое подключение к БД	Требует ключ OpenAI API

– **Непрозрачность технологического ядра**. Коммерческие сервисы не раскрывают, какие модели и методы они используют «под капотом». В то же время, многие простые open-source проекты используют базовые подходы без применения передовых техник. Создание сервиса на основе доказуемо лидирующего фреймворка, такого как XiYan-SQL, является важным преимуществом с точки зрения научной и инженерной проработки.

Таким образом, выявлена четкая потребность в создании веб-сервиса с открытым исходным кодом, который не требует от конечного пользователя сторонних API-ключей, прост в использовании и основан на современной, высокопроизводительной модели для генерации SQL-запросов. Разработка такого продукта не только представляет академический интерес как пример интеграции лидирующей модели в прикладной инструмент, но и обладает высокой практической ценностью, предлагая рынку уникальное и востребованное решение. Это и определяет актуальность и новизну дальнейшей работы.

1.5 Выводы

В рамках данного исследования был проведен комплексный анализ проблематики взаимодействия нетехнических пользователей с базами данных, а также обзор и сравнение существующих подходов и технологий для решения этой проблемы. Основной целью анализа было создание концепта итогового продукта при разработке собственного естественно-языкового интерфейса, а также проверка его актуальности и новизны подходов. По результатам проделанной работы были сделаны следующие ключевые выводы, которые определяют дальнейший ход проекта:

1. Выявлена проблема возникновения трудностей у нетехнических пользователей при взаимодействии с базами данных. Анализ традиционных способов взаимодействия с базами данных однозначно показал, что все они требуют от пользователя наличия специальных технических знаний, что и является основным препятствием для нетехнических специалистов.
2. Проделан сравнительный анализ адаптированных пользовательских интерфейсов. На основании преимуществ и недостатков таких интерфейсов, как QBE, VQS или NLIDB — был сделан вывод, что именно естественно-языковые интерфейсы (NLIDB) являются наиболее перспективным направлением, так как они предлагают самый высокий уровень абстракции и интуитивности, полностью скрывая от пользователя сложность структуры данных и синтаксиса формальных языков, что дало повод сконцентрироваться на исследовании именно такого вида взаимодействия с БД.
3. Определена доминирующая технология. Глубокий анализ систем, решающих задачу преобразования естественного языка в SQL (Text-to-SQL), показал четкую эволюцию от ранних систем на правилах к современным решениям на основе больших языковых моделей (LLM). Было установлено, что лучшую производительность сегодня демонстрируют гибридные LLM-основанные фреймворки, которые комбинируют различные методологии для достижения максимальной точности и робастности при решении исходной задачи.
4. Выбрано конкретное технологическое ядро. На основе анализа производительности, архитектурных особенностей и практической доступности был выбран фреймворк XiYan-SQL в качестве базовой системы для реализации собственного NLIDB. Этот выбор обоснован его лидирующими позициями в ключевых отраслевых бенчмарках, современной ансамблевой архитектурой, а также наличием открытого исходного кода, что позволяет интегрировать его в собственный проект.
5. Обоснована актуальность и новизна разработки. Было установлено, что на данный момент практически отсутствуют веб-сервисы, которые бы одновременно являлись:
   - Полностью бесплатными и доступными онлайн без локальной установки.
   - Основанными на открытом исходном коде.
   - Не требующими от конечного пользователя предоставления собственных API-ключей к платным проприетарным LLM.

Таким образом, результаты анализа, проведенного в данной части УИР, подтвердили актуальность поставленной задачи и позволили сформировать план для ее решения. Даль-

нейшая работа будет сконцентрирована на проектировании, разработке и тестировании собственного веб-сервиса, который будет использовать фреймворк XiYan-SQL в качестве ядра для трансляции запросов, предоставляя доступный, бесплатный и прозрачный инструмент для широкого круга нетехнических пользователей.

1.6 Цели и задачи на УИР

Целью данного УИР является проектирование модели интеграции современного многомодульного NLIDB-ядра XiYan-SQL во внешний веб-сервис. Эта интеграция достигается через разработку архитектуры компонента-оркестратора, а именно **MCP-клиента**, который инкапсулирует всю сложность взаимодействия с ядром.

Практической апробацией предложенной модели служит разработка и тестирование программного прототипа веб-сервиса. На данном этапе исследования прототип **эмулирует** взаимодействие с ядром XiYan-SQL, что позволяет проверить корректность и жизнеспособность спроектированной архитектуры в контролируемой среде.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать архитектуру и протокол взаимодействия многоагентного сервера XiYan-SQL MCP Server для определения требований к клиентской части.
2. Спроектировать логику работы (модель взаимодействия) MCP-клиента, определяющую полный жизненный цикл обработки пользовательского запроса. Это включает последовательность вызовов к различным AI-агентам ядра: агенту связывания со схемой, генераторам кандидатов и агенту выбора конечного результата.
3. Определить внутренний API компонента MCP-клиента для его взаимодействия с остальными частями серверной логики веб-приложения.
4. Спроектировать общую клиент-серверную архитектуру веб-сервиса, который предоставляет пользователю интерфейс для выполнения запросов к базам данных на естественном языке.
5. Разработать программный прототип веб-сервиса, реализующий спроектированную архитектуру. Прототип должен включать серверную часть (Backend), пользовательский интерфейс (Frontend) и, самое главное, модуль MCP-клиента с **эмуляцией** вызовов к ядру XiYan-SQL.
6. Провести функциональное тестирование разработанного прототипа для проверки корректности работы реализованной модели взаимодействия MCP-клиента и пользовательского интерфейса.

2. Модель интеграции технологического ядра XiYan-SQL во внешний сервис

Аннотация. *В данной главе рассматривается современный, многокомпонентный подход к построению систем Text-to-SQL. Анализируется эволюция архитектур от монолитных моделей к системам, использующим инструменты и протоколы взаимодействия, такие как Model Context Protocol (MCP). Формулируется ключевая разработческая задача, заключающаяся в проектировании программного компонента-организатора для управления жизненным циклом запроса в такой сложной системе. Предлагается и детально описывается архитектурное решение этой задачи, которое ляжет в основу программной реализации в следующей главе.*

2.1 Рабочий процесс, предлагаемый XiYan-SQL

Фреймворк XiYan-SQL представляет собой инновационный подход к решению задач преобразования естественного языка в SQL (Text-to-SQL), использующий ансамбль из нескольких генераторов для улучшения качества и разнообразия кандидатов [38]. Его рабочий процесс, который проиллюстрировать(см. рис. 2.1), можно разделить на три основных компонента: связывание со схемой, генерация кандидатов и выбор итогового кандидата.

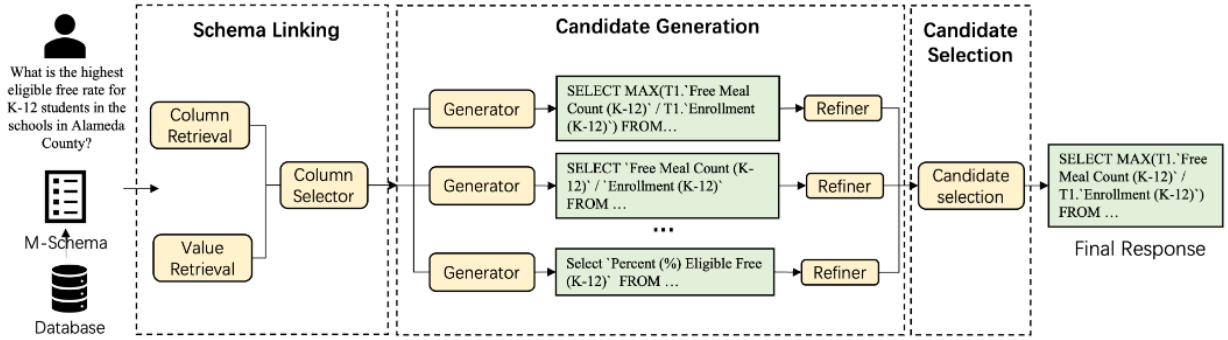


Рисунок 2.1 – Схема рабочего процесса, предлагаемого XiYan-SQL

Связывание со схемой (Schema Linking). Это первый и ключевой этап, на котором система анализирует запрос пользователя и обширную схему базы данных. Его задача — отобрать только релевантные для запроса таблицы и столбцы, а также извлечь из них необходимые значения. Это позволяет минимизировать количество ненужной информации и сосредоточиться на данных, имеющих отношение к задаче. Результат этого этапа представляется в

виде компактной и четкой полуструктурированной схемы *M-Schema*, которая передается на следующий этап.

Генерация кандидатов (Candidate Generation). На этом этапе в работу вступают несколько различных генераторов SQL-запросов. XiYan-SQL комбинирует два подхода:

- **Генераторы на основе Fine-Tuning (SFT)**. Специализированные модели, тонко настроенные на генерацию высокоточных и стилистически разнообразных SQL-запросов.
- **Генераторы на основе In-Context Learning (ICL)**. Используются для повышения гибкости и способности генерировать сложные запросы, используя возможности больших языковых моделей (LLM) с помощью тщательно подобранных примеров.

Каждый сгенерированный кандидат затем проходит через модуль *Refiner*, который пытается исправить возможные логические или синтаксические ошибки, используя результаты выполнения запроса или информацию об ошибках.

Выбор кандидата (Candidate Selection). После получения набора разнообразных и исправленных SQL-кандидатов, финальный компонент определяет наилучший вариант. Вместо простого выбора самого частого запроса (self-consistency), XiYan-SQL использует специально обученную модель, которая оценивает все нюансы кандидатов в контексте исходного вопроса и схемы данных, чтобы выбрать наиболее корректный и разумный SQL-запрос.

Такая многокомпонентная архитектура позволяет XiYan-SQL достигать высокой точности и надежности в решении сложных задач Text-to-SQL в различных сценариях [38].

2.2 Эволюция интеграции LLM с внешними программами

Как было показано в предыдущей главе, большие языковые модели (LLM) стали технологическим ядром современных NLIDB. Анализ показывает, что за последние годы произошла стремительная эволюция подходов к интеграции LLM с внешними инструментами и сервисами, которую можно условно разделить на три этапа (см. рис. 2.2).

Этап I: Монолитная LLM. На начальном этапе LLM использовалась как целостный «черный ящик». Весь контекст, включая вопрос пользователя и полную схему базы данных, передавался в модель одним запросом, а на выходе ожидался, если смотреть в контексте задачи Text-to-SQL, готовый SQL-код. Этот подход прост в реализации, но страдает от ряда фундаментальных проблем: неспособность эффективно работать в контексте задачи, отсутствие контроля над процессом генерации и сложность в отладке.

Этап II: LLM с инструментами. Следующим шагом стало наделение LLM возможностью использовать внешние инструменты через вызовы API. Например, модель могла сначала вызвать инструмент для поиска по веб-страницам, а затем использовать полученную информа-

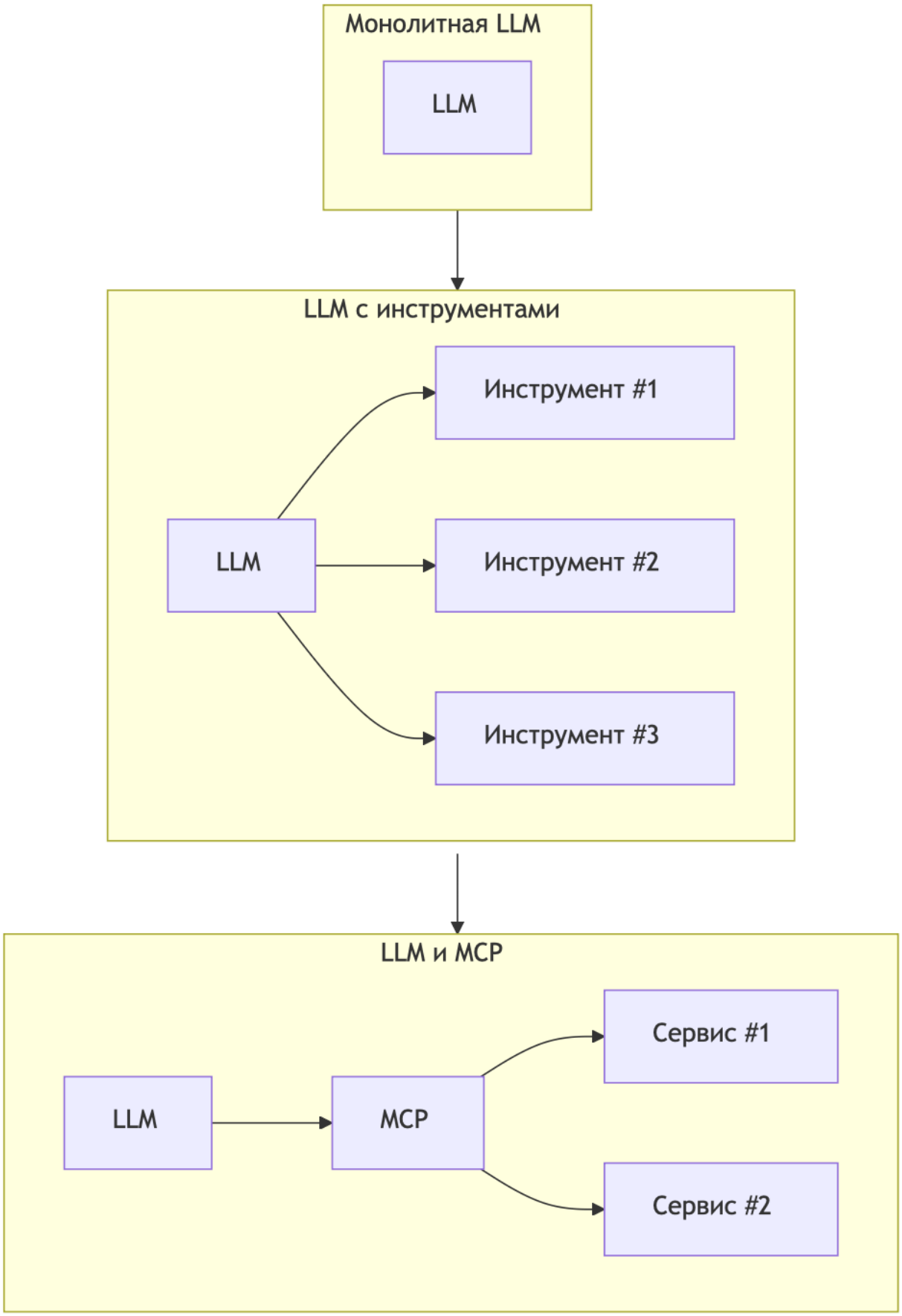


Рисунок 2.2 – Три этапа эволюции внедрения LLM-систем во внешние сервисы

цию для ответа. Этот подход значительно расширил возможности LLM. Однако он породил новую проблему, вызванную отсутствием стандарта. Каждый инструмент и сервис имеет собственный, уникальный способ вызова и формат данных, что приводит к необходимости писать большое количество связующего кода и усложняет масштабирование системы при добавлении новых инструментов.

Этап III: LLM и стандартизированные протоколы (MCP). В конце 2024 года появился стандарт — *Model Context Protocol (MCP)*. MCP — это открытый протокол, который стандартизирует способ, которым LLM (или AI-агенты) взаимодействуют с внешними инструментами и источниками данных. Он действует как универсальное связующее звено между моделью и сервисами, позволяя им общаться на едином, стандартизированном языке. Это решает проблему масштабируемости и значительно упрощает добавление новых инструментов в экосистему.

Фреймворк XiYan-SQL, выбранный в качестве технологического ядра для данного проекта, предоставляет реализацию такого подхода в виде *XiYan-SQL MCP Server* [1]. Этот сервер является не единой моделью, а ансамблем из нескольких независимых модулей, взаимодействие с которыми происходит по протоколу MCP. Такая архитектура является передовой, однако она порождает новую, нетривиальную разработческую задачу.

2.3 Протокол MCP (Model Context Protocol)

Model Context Protocol (MCP) — открытый, стандартизированный протокол, призванный стать универсальным языком для общения между AI-модулями и внешними источниками данных или инструментами.

MCP действует как универсальное связующее звено, которое стандартизирует способ, которым AI-модуль запрашивает информацию или действие у инструмента и получает ответ. Протокол определяет четкую и единую структуру для сообщений, которыми обмениваются компоненты системы. Таким образом, вместо того чтобы напрямую обращаться к уникальному API каждого инструмента, модул формирует стандартизированный MCP-запрос. Специальный адаптер на стороне инструмента преобразует этот запрос в вызов своего API и возвращает ответ также в стандартном MCP-формате.

К примеру, XiYan MCP Server предлагает архитектуру, позволяющую два способа интегрировать сервер в проект: локальный и удаленный (см. рис. 2.3).

Внедрение MCP в архитектуру AI-систем дает несколько значительных преимуществ: модульность и взаимозаменяемость инструментов и моделей, масштабируемость системы

[1] URL: `https://github.com/XGenerationLab/xiyan_mcp_server`

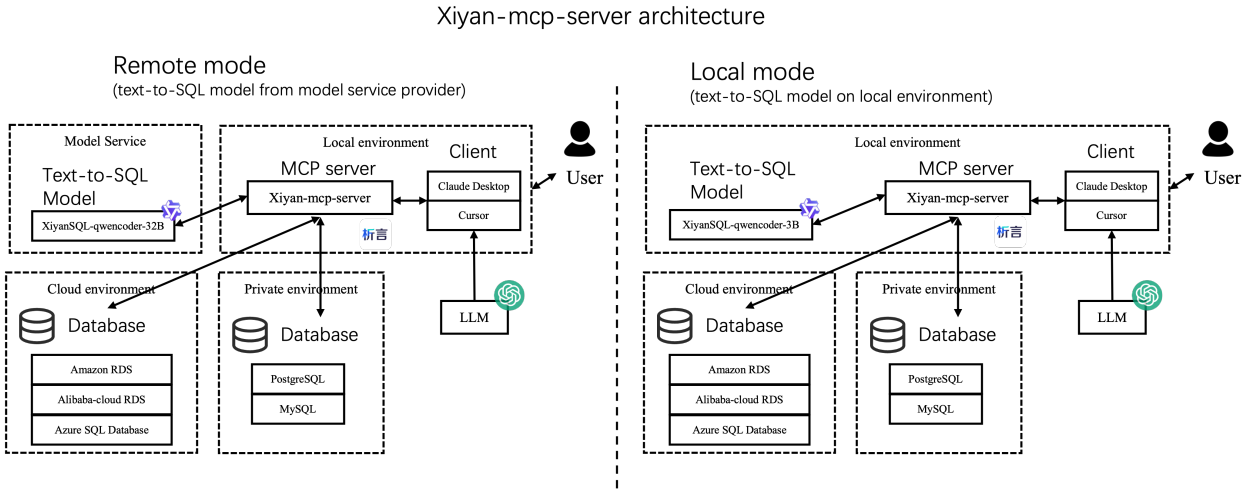


Рисунок 2.3 – Архитектура технологического ядра XiYan MCP Server

и простота разработки. Несмотря на преимущества, у протокола есть и текущие ограничения. Во-первых, как дополнительный уровень абстракции, он может вносить небольшую задержку в выполнение запросов. Во-вторых, будучи новым стандартом, он может пока не покрывать все специфические случаи использования для каждого возможного инструмента. Примеры применения включают:

– AI-ассистентов, использующих внешние сервисы (заказ билетов, прогноз погоды).
– Корпоративные чат-боты, подключенные к внутренним базам знаний.
– Сложные системы, где несколько специализированных AI-модулей совместно решают задачу, обмениваясь данными через MCP.

Перспективы MCP связаны с формированием глобального «рынка» совместимых AI-инструментов. Это стимулирует появление специализированных сервисов-«оркестраторов», управляющих сложными цепочками вызовов между модулями.

2.4 Моделирование MCP-клиента для интеграции с ядром XiYan-SQL

Как было установлено, технологическое ядро XiYan-SQL представляет собой *MCP-сервер*, который предоставляет доступ к ансамблю независимых AI-модулей. Для интеграции такого ядра во внешний веб-сервис необходимо спроектировать на стороне сервиса соответствующий **MCP-клиент**. Этот компонент будет отвечать за всю логику взаимодействия с ядром, выступая в роли «Организатора» или «Оркестратора» полного жизненного цикла обработки пользовательского запроса.

Задача данного компонента — инкапсулировать сложную, многошаговую последовательность вызовов к различным модулям MCP-сервера, предоставляя остальной части веб-приложения

простой и высокоуровневый интерфейс. Цикл обработки одного запроса, управляемый MCP-клиентом, состоит из нескольких последовательных шагов, как показано на схеме (см. рис. 2.4).

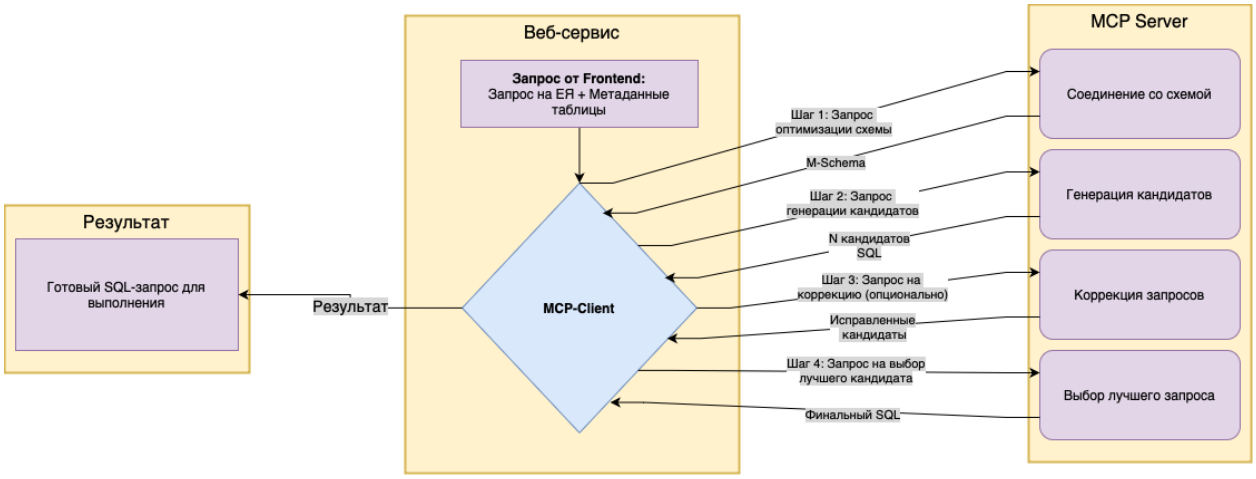


Рисунок 2.4 – Модель взаимодействия MCP-клиента с MCP-сервером XiYan-SQL

Данную модель можно описать следующими последовательно выполняющимися шагами:

1. **Получение запроса**. MCP-клиент получает от основной логики бэкенда текст вопроса пользователя и метаданные целевой таблицы.
2. **Связывание со схемой**. Клиент формирует и отправляет первый MCP-запрос к *Модулю связывания схемы*. В ответ он получает оптимизированную схему данных (M-Schema).
3. **Генерация кандидатов**. Используя полученную M-Schema, клиент параллельно обращается к нескольким *Модулям генерации кандидатов*, передавая им текст вопроса. В результате он получает несколько версий-кандидатов SQL-запроса.
4. **Уточнение и коррекция (опционально)**. При необходимости клиент может передать полученных кандидатов *Модулю уточнения* для автоматического исправления возможных ошибок. Этот шаг может быть итеративным.
5. **Выбор лучшего кандидата**. Финальный набор кандидатов передается *Модулю выбора*, который на основе своих внутренних метрик выбирает один, наиболее вероятный SQL-запрос.
6. **Возврат результата**. MCP-клиент получает от *Модуля выбора* финальный SQL-запрос и возвращает его основной логике бэкенда для последующего выполнения и отображения результата пользователю.

Предложенная архитектура на основе MCP-клиента позволяет полностью абстрагировать основное веб-приложение от сложности внутреннего устройства NLIDB-ядра. В следу-

ющей главе будет описан процесс программной реализации прототипа веб-сервиса, который использует данную архитектуру, где модуль MCP-клиента будет реализован с эмуляцией вызовов к ядру XiYan-SQL.

3. Проектирование и программная реализация прототипа веб-сервиса

Аннотация. *В данной главе описывается процесс проектирования и программной реализации прототипа веб-сервиса с естественно-языковым интерфейсом. На основе сформулированных в предыдущей главе требований, с использованием языка моделирования UML, разрабатывается архитектура системы. Представляются диаграммы вариантов использования, компонентов и последовательности, которые детально описывают как статическую структуру, так и динамику взаимодействия элементов системы. В завершение приводится описание выбранного технологического стека и ключевых аспектов реализации прототипа.*

3.1 Проектирование архитектуры системы с использованием UML

Проектирование является критически важным этапом разработки любого программного продукта. Для визуализации и формализации архитектуры будущего веб-сервиса был использован унифицированный язык моделирования UML (Unified Modeling Language). Были построены три диаграммы, описывающие систему с разных уровней абстракции.

3.1.1 Диаграмма вариантов использования (Use Case Diagram)

Диаграмма вариантов использования является наиболее высокоуровневым представлением системы. Она определяет границы системы, ее основных действующих лиц (акторов) и цели, которые эти акторы могут достигать при взаимодействии с системой.

Для нашего веб-сервиса был определен один актор — **Пользователь**. Его основные цели (варианты использования) представлены на рис. 3.1.

Как видно из схемы, Пользователь может выполнять три ключевые функции:

- **Управлять таблицами**. Этот вариант использования включает в себя загрузку новых таблиц (CSV/Excel), их переименование и удаление, а также добавление описаний для столбцов.
- **Задавать вопросы к данным**. Основная функция системы, позволяющая пользователю формулировать запросы на естественном языке к выбранной таблице.
- **Управлять профилем**. Включает в себя регистрацию, аутентификацию и изменение данных своего аккаунта (псевдоним, пароль, аватар).

Рисунок 3.1 – Диаграмма вариантов использования

3.1.2 Диаграмма компонентов (Component Diagram)

Диаграмма компонентов описывает физическую структуру системы, показывая, из каких крупных программных блоков она состоит и как они связаны между собой. Эта диаграмма является основной архитектурной схемой нашего веб-сервиса (см. рис. 3.2).

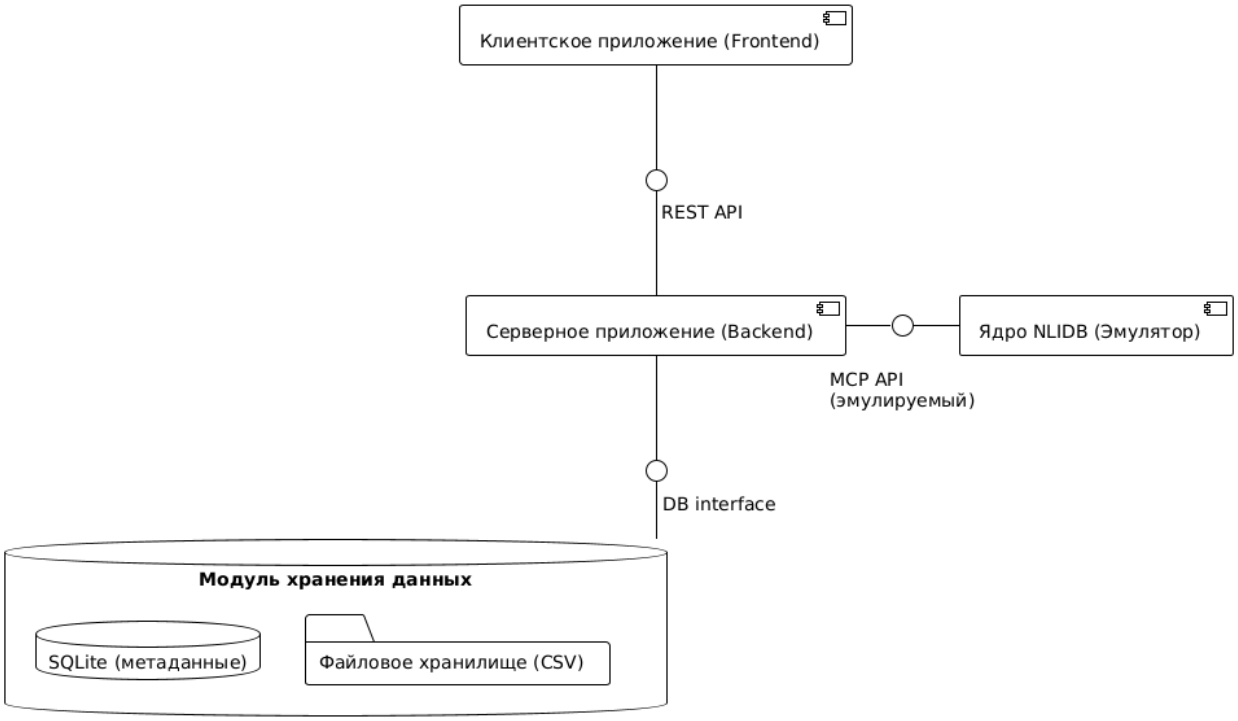


Рисунок 3.2 – Диаграмма компонентов системы

Система состоит из четырех основных компонентов:

1. **Клиентское приложение (Frontend):** Компонент, работающий в браузере пользова-

теля. Реализован на HTML/CSS/JS. Отвечает за пользовательский интерфейс и взаимодействие с бэкендом через REST API.

2. **Серверное приложение (Backend):** Основной компонент, реализованный на Python с использованием фреймворка FastAPI. Он предоставляет REST API, управляет бизнес-логикой и выступает в роли организатора для других компонентов.
3. **Модуль хранения данных:** Отвечает за персистентное хранение информации. Он состоит из базы данных **SQLite** для метаинформации (пользователи, таблицы, описания столбцов) и **Файлового хранилища** для загруженных CSV/Excel файлов и аватаров пользователей.
4. **Ядро NLIDB (Эмулятор):** В рамках прототипа — это компонент-эмулятор, имитирующий работу настоящего XiYan-SQL MCP Server. Он предоставляет внутренний API, который полностью соответствует спроектированному протоколу взаимодействия с целевой системой.

3.1.3 Диаграмма последовательности (Sequence Diagram)

Диаграмма последовательности детализирует взаимодействие между компонентами во времени. Она наглядно демонстрирует, какие вызовы и в какой последовательности происходят для выполнения конкретного варианта использования. На рис. 3.3 показана последовательность действий для самого важного сценария — обработки запроса пользователя на естественном языке.

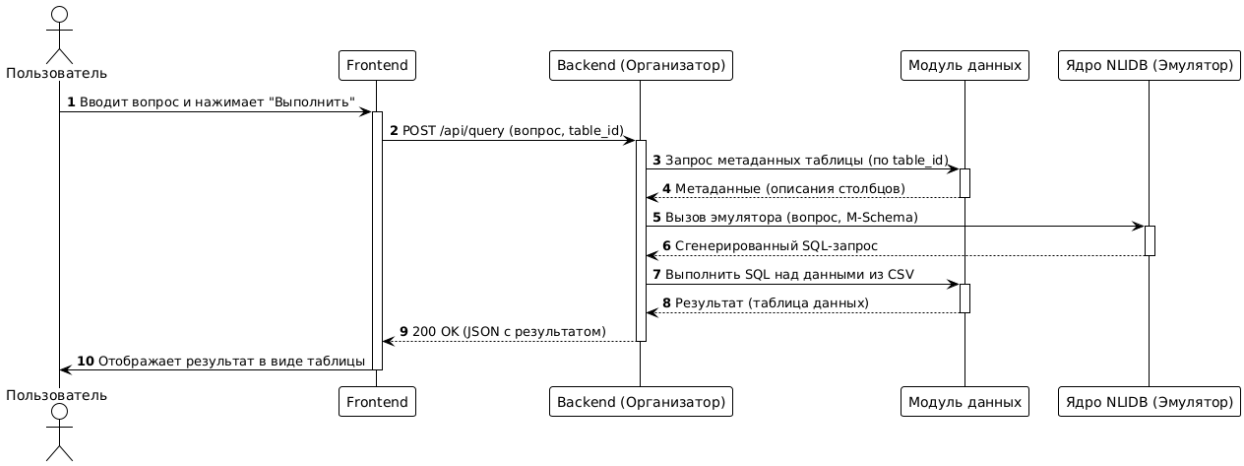


Рисунок 3.3 – Диаграмма последовательности для обработки ЕЯ-запроса

Процесс, изображенный на диаграмме, состоит из следующих шагов:

1. Пользователь вводит вопрос в интерфейсе и нажимает кнопку «Выполнить».

2. **Frontend** отправляет асинхронный `POST` запрос на эндпоинт `/api/query` **Backend**-сервера, передавая текст вопроса и ID выбранной таблицы.
3. **Backend (Организатор)** получает запрос и сначала обращается к **Модулю данных**, чтобы получить метаинформацию о таблице (путь к файлу, описания столбцов) из базы SQLite.
4. Получив метаданные, **Backend** формирует из них M-Schema и обращается к **Ядру NLIDB (Эмулятору)**, передавая ему вопрос и схему.
5. **Эмулятор** для тестовых сценариев возвращает заранее заготовленный SQL-запрос.
6. **Backend** получает SQL. Он снова обращается к **Модулю данных**, но на этот раз для выполнения операции: он считывает нужный CSV-файл в объект pandas DataFrame и выполняет над ним полученный SQL-запрос с помощью библиотеки `pandasql`.
7. **Модуль данных** возвращает результат выполнения — новый DataFrame.
8. **Backend** сериализует результирующий DataFrame в формат JSON и отправляет его обратно на **Frontend** в теле успешного HTTP-ответа.
9. **Frontend** получает данные и отрисовывает их в виде таблицы для Пользователя.

Данные диаграммы полностью описывают спроектированную архитектуру и служат основой для этапа программной реализации.

3.2 Выбор стека технологий

На основе спроектированной в предыдущем разделе архитектуры был выбран конкретный стек технологий для программной реализации прототипа. Выбор каждого инструмента и фреймворка обусловлен функциональными требованиями проекта, необходимостью разработки прототипа и возможностями для дальнейшего масштабирования системы.

Серверная часть (Backend). Для реализации серверной части был выбран язык программирования **Python** и асинхронный веб-фреймворк **FastAPI**. Этот выбор обусловлен следующими причинами:

- **Экосистема Python для ИИ и анализа данных**. Ключевое ядро системы, XiYan-SQL, написано на Python. Использование Python для бэкенда является наиболее нативным и эффективным решением, так как позволяет избежать сложностей межъязыкового взаимодействия и напрямую интегрировать необходимые библиотеки.
- **Высокая производительность FastAPI**. FastAPI построен на базе Starlette и Pydantic, что обеспечивает ему производительность, сопоставимую с решениями на Go и Node.js. Его асинхронная природа идеально подходит для обработки запросов, связанных с длительными операциями, такими как обращение к LLM.

– **Автоматическая документация API**. FastAPI автоматически генерирует интерактивную документацию для API (Swagger UI и ReDoc), что значительно упрощает процесс разработки, тестирования и отладки взаимодействия между клиентской и серверной частями.
– **Работа с данными**. Для манипуляции данными из загружаемых CSV-файлов используется библиотека **pandas** — де-факто стандарт для анализа данных в Python. Для выполнения SQL-запросов над объектами DataFrame используется библиотека **pandasql**, что позволяет обрабатывать данные из файлов так, как если бы они находились в реляционной базе данных.

Клиентская часть (Frontend). Для реализации пользовательского интерфейса был выбран базовый стек из **HTML5, CSS3 и нативного JavaScript (Vanilla JS)** по следующим причинам:

– **Фокус на основной задаче**. Основная сложность и новизна проекта лежат в серверной части и архитектуре взаимодействия с NLIDB-ядром. Использование стандартного стека без тяжелых фреймворков (таких как React или Vue) позволяет сконцентрировать усилия на реализации ключевой функциональности.
– **Отсутствие зависимостей и простота развертывания**. Данный подход не требует сложной системы сборки и зависимостей (Node.js, npm), что упрощает разработку и развертывание прототипа.
– **Гибкость**. Спроектированный REST API полностью отделяет логику бэкенда от представления. Это означает, что в будущем клиентская часть может быть легко заменена на более современный фреймворк без каких-либо изменений в серверной архитектуре.

База данных (Database). Для хранения метаинформации (данные пользователей, сведения о загруженных таблицах и описания столбцов) была выбрана легковесная встраиваемая СУБД **SQLite**. Вот причины, по которым была выбрана именно она:

– **Серверная независимость**. SQLite не требует отдельного серверного процесса. База данных представляет собой один файл, что идеально подходит для прототипа и упрощает его переносимость и настройку.
– **Нативная поддержка в Python**. SQLite встроена в стандартную библиотеку Python, что избавляет от необходимости устанавливать внешние драйверы. Для удобной работы с базой данных используется **SQLAlchemy** — популярный ORM (Object-Relational Mapper), который позволяет работать с таблицами как с Python-объектами.

Таким образом, выбранный технологический стек представляет собой сбалансированное

решение, которое позволяет быстро и эффективно реализовать прототип, полностью соответствующий поставленным задачам, и при этом закладывает прочный фундамент для будущего развития и усложнения системы.

3.3 Описание программной реализации прототипа веб-сервиса

На основе спроектированной архитектуры и выбранного стека технологий был реализован программный прототип веб-сервиса. В данном разделе приводится описание ключевых модулей и функций системы. Фрагменты исходного кода, иллюстрирующие реализацию, вынесены в приложение (см. прил. Б).

3.3.1 Описание реализованных функций и модулей

Программный код проекта организован в модульную структуру, соответствующую современным практикам разработки на FastAPI. Корневая директория содержит главный файл приложения `main.py`, файлы конфигурации и директории с различными компонентами системы: `api`, `core`, `db`, `features`, `services`.

Точка входа и конфигурация приложения. Файл `main.py` является точкой входа в приложение. В нем создается экземпляр класса `FastAPI`, настраиваются middleware-компоненты (включая `CORSMiddleware` для обработки кросс-доменных запросов), монтируются директории для статических файлов (`/static`) и загрузок (`/uploads`), а также подключаются все API-маршруты из модуля `api`. Важной частью является функция с жизненным циклом (`lifespan`), которая при старте сервера создает необходимые директории, например, для аватаров пользователей.

Модуль аутентификации и управления пользователями. Данная функциональность реализована в директории `features/users/`.

1. **Модель данных (`models.py`)**. Описана модель `User` с использованием SQLAlchemy ORM. Она содержит поля для хранения псевдонима, хэшированного пароля, URL аватара и флагов статуса пользователя (активен, суперпользователь, стандартный аватар).
2. **Операции с данными (`crud.py`)**. Реализован класс `CRUDUser`, который инкапсулирует всю логику прямого взаимодействия с базой данных: создание пользователя, поиск по имени, аутентификация, обновление. Для хэширования паролей используется библиотека `passlib` с алгоритмом bcrypt, что обеспечивает безопасное хранение учетных данных.
3. **API эндпоинты (`api.py`)**. Определены все публичные маршруты для работы с пользователями:

3.1. `/register`: Регистрация нового пользователя с валидацией длины и формата псевдонима и пароля.

3.2. `/login/access-token`: Аутентификация пользователя и выдача JWT-токена доступа, который используется для авторизации всех последующих запросов.

3.3. `/me`, `/me/username`, `/me/password`: Маршруты для получения и обновления данных текущего пользователя.

3.4. `/me/avatar`: Эндпоинты для загрузки и удаления пользовательского аватара.

4. **Сервис генерации аватаров (`services/avatar_service.py`)**. Выделенный сервис, который при регистрации или удалении кастомного аватара генерирует стандартное изображение с первой буквой псевдонима пользователя на цветном фоне, созданном на основе хэша от имени. Для генерации используется библиотека `Pillow`.

Модуль управления таблицами данных. Аналогично пользователям, логика работы с таблицами вынесена в модуль `features/tables/`.

1. **Модель данных (`models.py`)**. Описана модель `Table`, связанная с моделью `User` отношением «один-ко-многим». Хранит имя таблицы, оригинальное имя файла, путь к файлу на сервере и ID владельца.
2. **Сервис обработки таблиц (`services/table_service.py`)**. Этот сервисный слой содержит основную бизнес-логику. Функция `process_and_save_table` отвечает за получение загруженного файла, его валидацию (проверка расширения на `.csv` или `.xlsx`), очистку имени, проверку на дубликаты, сохранение файла на диск с уникальным именем (с помощью `uuid`) и, наконец, вызов CRUD-функции для создания записи в БД. Функция `get_table_preview` использует библиотеку `pandas` для чтения файла и возвращает первые 5 строк, названия столбцов и общее количество строк.
3. **API эндпоинты (`api.py`)**. Предоставляют интерфейс для фронтенда: загрузка файла (`/upload`), получение списка таблиц пользователя (`/`) и удаление таблицы (`/{table_id}`). Все маршруты защищены и требуют наличия токена аутентификации.

Модуль обработки NL-запросов (Организатор и Эмулятор). Это центральный модуль, реализующий основную функцию системы. В текущем прототипе он состоит из двух частей:

1. **Организатор:** Его роль выполняет обработчик эндпоинта `/api/query/`. Он получает от клиента текст вопроса и ID таблицы, извлекает из БД метаданные, формирует из них M-Schema и передает их в ядро NLIDB.
2. **Ядро NLIDB (Эмулятор):** Реализовано в виде функции-заглушки `convert_text_to_sql` в файле `services/text_to_sql_service.py`. Эта функция имитирует работу

настоящего ядра XiYan-SQL. Для демонстрационных целей в ней реализована простая логика, которая возвращает заранее заготовленные SQL-запросы для нескольких ключевых слов.

Пример реализации эмулятора приведен в листинге Б.5. Такая архитектура с эмуляцией позволяет полностью протестировать сквозное взаимодействие всех компонентов системы, отложив сложную интеграцию с реальным MCP-сервером на следующие этапы разработки.

После получения SQL-запроса от эмулятора, Организатор использует библиотеку `pandasql` для его выполнения над DataFrame, полученным из пользовательского CSV-файла, и возвращает результат на клиентскую часть.

Заключение

В заключении в тезисной форме необходимо отразить результаты работы:

– аналитические (что изучено/проанализировано);

– теоретические;

– инженерные (что спроектировано);

– практические (что реализовано/внедрено).

Примерная формула такая: по каждому указанному пункту приводится по 3-5 результатов, каждый результат излагается в объеме до 5 фраз или предложений.

Также есть смысл привести предполагаемые направления для будущей работы.

Общий объем заключения не должен превышать 1,5 страниц (1 страницы для УИРов).

Список литературы

1. *Савоськин И. В.*, *Фирсов А. О.* Исследование способов применения NoSQL и реляционных баз данных // E-Scio. — Россия, Саранск, 2019. — 6 (33), 6 (33). — С. 101—108. — URL: https://cyberleninka.ru/article/n/issledovanie-sposobov-primeneniya-nosql-i-relyatsionnyh-baz-dannyh (дата обр. 12.06.2025).
2. *Karimi S.*, *Rasel A. A.*, *Abdullah M. S.* Natural language query and control interface for database using afghan language // 2022 International Conference on INnovations in Intelligent SysTems and Applications (INISTA). — IEEE, 2022. — P. 1–8. — URL: https://ieeexplore.ieee.org/abstract/document/9894168/ (visited on 06/12/2025).
3. *Болябкин М. В.*, *Овчинников Ф. В.*, *Башмуров Н. А.* Интеллектуальная система для преобразования запросов на естественном языке в SQL и их выполнения // Международный журнал гуманитарных и естественных наук. — Россия, Новосибирск, 2021. — Вып. 12—1, № 12—1. — С. 134—138. — ISSN 2500-1000. — URL: https://cyberleninka.ru/article/n/intellektualnaya-sistema-dlya-preobrazovaniya-zaprosov-na-estestvennom-yazyke-v-sql-i-ih-vypolneniya (дата обр. 15.05.2025).
4. *Бураков П. В.*, *Петров В. Ю.* Введение в системы баз данных: Учебное пособие // СПб: СПбГУ ИТМО. — 2010. — С. 29.
5. *Корягин Д. А.* Модели баз данных // Редакционный совет. — 2020. — С. 985. — URL: https://alley-science.ru/domains_data/files/Collection_of_journals/1_tom_May_2020.pdf#page=985 (дата обр. 12.06.2025).
6. *Smallcombe M.* SQL vs NoSQL: 5 Critical Differences / Integrate.io. — URL: https://www.integrate.io/blog/the-sql-vs-nosql-difference/ (visited on 06/12/2025).
7. *Клейменов К. С.*, *Скуднев Д. М.* NoSQL и реляционные базы данных // Актуальные проблемы естественных, математических, технических наук и их преподавания: сборник научных трудов. — 2022. — С. 60. — URL: https://elibrary.ru/download/elibrary_49224984_58821542.pdf#page=61 (дата обр. 12.06.2025).

8. *Maran M. M.*, *Paniavin N. A.*, *Poliushkin I. A.* Alternative Approaches to Data Storing and Processing // 2020 V International Conference on Information Technologies in Engineering Education (Inforino) (2020 V International Conference on Information Technologies in Engineering Education (Inforino)). — Moscow, Russia : IEEE, 04/2020. — P. 1–4. — ISBN 978-1-7281-4810-6. — DOI: 10.1109/Inforino48376.2020.9111708. — URL: https://ieeexplore.ieee.org/document/9111708/ (visited on 06/12/2025).

9. *Иванов В. А.*, *Лебедева М. Ю.* Обзор современных NoSQL баз данных // Информационно-вычислительные технологии и их приложения: сборник статей XXIII Международной научно-технической конференции, Пенза, 13–14 июня 2019 года. — Пенза : Пензенский государственный аграрный университет, 2019. — С. 86—90. — URL: https://elibrary.ru/item.asp?id=39149615 (дата обр. 12.06.2025).

10. *Якушин А. Ю.*, *Муковозов А. М.*, *Исмоилов М.* Сравнительный анализ реляционной базы данных и документоориентированной NoSQL базы данных в разрезе их применения при создании локального чата/мессенджера // Инновационная наука. — Россия, Уфа, 2018. — Вып. 4, № 4. — С. 73—83. — ISSN 2410-6070. — URL: https://cyberleninka.ru/article/n/sravnitelnyy-analiz-relyatsionnoy-bazy-dannyh-i-dokumentoorientirovannoy-nosql-bazy-dannyh-v-razreze-ih-primeneniya-pri-sozdanii (дата обр. 13.06.2025).

11. *Zloof M. M.* Query-by-example: A data base language // IBM systems Journal. — 1977. — Vol. 16, no. 4. — P. 324–343. — URL: https://ieeexplore.ieee.org/abstract/document/5388055/ (visited on 06/13/2025).

12. Visual Query Systems for Databases: A Survey / T. Catarci [et al.] // Journal of Visual Languages & Computing. — 1997. — Apr. 1. — Vol. 8, no. 2. — P. 215–260. — ISSN 1045-926X. — DOI: 10.1006/jvlc.1997.0037. — URL: https://www.sciencedirect.com/science/article/pii/S1045926X97900379 (visited on 06/13/2025).

13. *Пирогов В. Ю.* Некоторые особенности преподавания языка управления базами данных // Мир науки. Педагогика и психология. — Россия, Москва, 2018. — Т. 6, вып. 6, № 6. — С. 55. — URL: https://cyberleninka.ru/article/n/nekotorye-osobennosti-prepodavaniya-yazyka-upravleniya-bazami-dannyh (дата обр. 13.06.2025).

14. *Catarci T.*, *Costabile M. F.* Visual Query Systems // Journal of Visual Languages & Computing. — 1996. — 1 сент. — Т. 7, № 3. — С. 243—245. — ISSN 1045-926X. — DOI: 10.1006/jvlc.1996.0013. — URL: https://www.sciencedirect.com/science/article/pii/S1045926X96900130 (дата обр. 13.06.2025).

15. *Baskaran S.*, *Baskaran S.*, *Gopalarathinam S.* Syntatic and Semantic Visual Query Building Using Natural Language Processing // International Journal of Engineering Research. — 2013. — Vol. 2, no. 9.

16. *Soumis D.*, *Stamoulis G.*, *Koubarakis M.* A Visual Query Builder for DBpedia // Web Information Systems Engineering – WISE 2024 PhD Symposium, Demos and Workshops. Vol. 15463 / ed. by M. Barhamgi [et al.]. — Singapore : Springer Nature Singapore, 2025. — P. 243–248. — ISBN 978-981-96-1482-0 978-981-96-1483-7. — DOI: 10.1007/978-981-96-1483-7_20. — URL: https://link.springer.com/10.1007/978-981-96-1483-7_20 (visited on 06/14/2025).

17. *Stigeborn P.*, *Strömgren J.* Query builder for database system / Stigeborn Patrik, Strömgren Jonathan. — 2015. — URL: https://www.diva-portal.org/smash/record.jsf?pid=diva2:860417 (visited on 06/14/2025).

18. VIREX and VRXQuery: interactive approach for visual querying of relational databases to produce XML / A. Lo [et al.] // J Intell Inf Syst. — 2010. — Aug. 1. — Vol. 35, no. 1. — P. 21–49. — ISSN 1573-7675. — DOI: 10.1007/s10844-009-0087-6. — URL: https://doi.org/10.1007/s10844-009-0087-6 (visited on 06/14/2025).

19. *Anisyah A.*, *Widagdo T. E.*, *Nur Azizah F.* Natural Language Interface to Database (NLIDB) for Decision Support Queries // 2019 International Conference on Data and Software Engineering (ICoDSE) (2019 International Conference on Data and Software Engineering (ICoDSE)). — 11/2019. — P. 1–6. — DOI: 10.1109/ICoDSE48700.2019.9092769. — URL: https://ieeexplore.ieee.org/document/9092769 (visited on 06/14/2025).

20. *Бородин Д. С.*, *Строганов Ю. В.* К задаче составления запросов к базам данных на естественном языке // Новые информационные технологии в автоматизированных системах. — 2016. — № 19. — С. 119—126. — URL: https://cyberleninka.ru/article/n/k-zadache-sostavleniya-zaprosov-k-bazam-dannyh-na-estestvennom-yazyke (дата обр. 15.05.2025).

21. *Desai H.*, *Patil A.* Natural Language Interface to the Database Management System using NLTK in Python // 2023 International Conference on Networking and Communications (ICNWC) (2023 International Conference on Networking and Communications (ICNWC)). — 04/2023. — P. 1–7. — DOI: 10.1109/ICNWC57852.2023.10127509. — URL: https://ieeexplore.ieee.org/document/10127509 (visited on 06/14/2025).

22. Enhancing Relational Database Interaction through Open AI and Stanford Core NLP-Based on Natural Language Interface / S. Kumar [et al.] // 2024 5th International Conference on Recent Trends in Computer Science and Technology (ICRTCST) (2024 5th International Conference on Recent Trends in Computer Science and Technology (ICRTCST)). — 04/2024. — P. 589–602. — DOI: 10.1109/ICRTCST61793.2024.10578418. — URL: https://ieeexplore.ieee.org/document/10578418 (visited on 06/14/2025).

23. Natural Language to SQL: Where Are We Today? / H. Kim [et al.] // Proc. VLDB Endow. — 2020. — June. — Vol. 13, no. 10. — P. 1737–1750. — ISSN 2150-8097. — DOI: 10.14778/3401960.3401970. — URL: https://dl.acm.org/doi/10.14778/3401960.3401970 (visited on 05/15/2025).

24. *Liu M.*, *Xu J.* NLI4DB: A Systematic Review of Natural Language Interfaces for Databases. — 03/04/2025. — DOI: 10.48550/arXiv.2503.02435. — arXiv: 2503.02435 [cs]. — URL: http://arxiv.org/abs/2503.02435 (visited on 06/14/2025). — Pre-published.

25. Large Language Model Enhanced Text-to-SQL Generation: A Survey / X. Zhu [et al.]. — 10/08/2024. — DOI: 10.48550/arXiv.2410.06011. — arXiv: 2410.06011 [cs]. — URL: http://arxiv.org/abs/2410.06011 (visited on 06/14/2025). — Pre-published.

26. *Poetra D. A.*, *Esterina Widagdo T.*, *Azizah F. N.* Natural Language Interface to Database (NLIDB) for Query with Temporal Aspect // 2019 International Conference on Data and Software Engineering (ICoDSE) (2019 International Conference on Data and Software Engineering (ICoDSE)). — 11/2019. — P. 1–6. — DOI: 10.1109/ICoDSE48700.2019.9092618. — URL: https://ieeexplore.ieee.org/document/9092618 (visited on 06/14/2025).

27. *Hamaz K.*, *Benchikha F.* A novel method for providing relational databases with rich semantics and natural language processing // JEIM. — 2017. — Apr. 10. — Vol. 30, no. 3. —

P. 503–525. — ISSN 1741-0398. — DOI: 10.1108/JEIM-01-2015-0005. — URL: https://www.emerald.com/insight/content/doi/10.1108/JEIM-01-2015-0005/full/html (visited on 06/14/2025).

28. Comparative study on the customization of natural language interfaces to databases / R. A. Pazos R. [et al.] // Springerplus. — 2016. — Apr. 30. — Vol. 5. — P. 553. — ISSN 2193-1801. — DOI: 10.1186/s40064-016-2164-y. — PMID: 27190752. — URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4851672/ (visited on 06/14/2025).

29. Dataset for a Neural Natural Language Interface for Databases (NNLIDB) / F. Brad [et al.]. — 07/11/2017. — DOI: 10.48550/arXiv.1707.03172. — arXiv: 1707.03172 [cs]. — URL: http://arxiv.org/abs/1707.03172 (visited on 06/14/2025). — Pre-published.

30. On Modern Text-to-SQL Semantic Parsing Methodologies for Natural Language Interface to Databases: A Comparative Study / M. Visperas [et al.] // 2023 International Conference on Artificial Intelligence in Information and Communication (ICAIIC) (2023 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)). — 02/2023. — P. 390–396. — DOI: 10.1109/ICAIIC57133.2023.10067134. — URL: https://ieeexplore.ieee.org/document/10067134 (visited on 06/14/2025).

31. CodeS: Towards Building Open-source Language Models for Text-to-SQL / H. Li [et al.] // Proceedings of the ACM on Management of Data. — 2024. — May 29. — DOI: 10.1145/3654930. — URL: https://dl.acm.org/doi/10.1145/3654930 (visited on 06/14/2025).

32. *Baik C.*, *Jagadish H. V.*, *Li Y.* Bridging the Semantic Gap with SQL Query Logs in Natural Language Interfaces to Databases // 2019 IEEE 35th International Conference on Data Engineering (ICDE). — 04/2019. — P. 374–385. — DOI: 10.1109/ICDE.2019.00041. — arXiv: 1902.00031 [cs]. — URL: http://arxiv.org/abs/1902.00031 (visited on 06/14/2025).

33. *Verma P.*, *Arora S.*, *Batra K.* Punjabi Language Interface to Database: a brief review // ArXiv. — 2013. — June 18. — URL: https://www.semanticscholar.org/paper/Punjabi-Language-Interface-to-Database%3A-a-brief-Verma-Arora/97c402b62376257839c96268e6feaead9d0f3fd1 (visited on 06/14/2025).

34. *Сидоров М. В.* Естественные и искусственные языки: проблема перевода // Ноосферные исследования. — Россия, Иваново, 2024. — Вып. 4, № 4. — С. 89—98. — URL: https://cyberleninka.ru/article/n/estestvennye-i-iskusstvennye-yazyki-problema-perevoda (дата обр. 15.05.2025).

35. *Karimi S.*, *Rasel A. A.*, *Abdullah M. S.* Non-English Natural Language Interface to Databases: A Systematic Review // 2022 IEEE 13th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON) (2022 IEEE 13th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)). — 10/2022. — P. 0391–0397. — DOI: 10.1109/IEMCON56893.2022.9946569. — URL: https://ieeexplore.ieee.org/document/9946569 (visited on 06/14/2025).

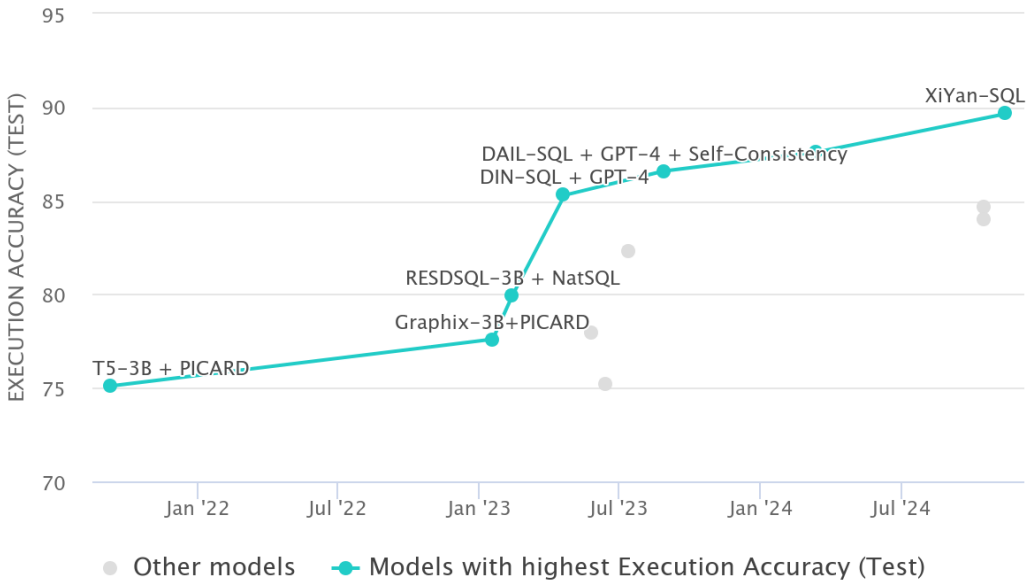
36. *Mohammadjafari A.*, *Maida A. S.*, *Gottumukkala R.* From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems. — 02/04/2025. — DOI: 10.48550/arXiv.2410.01066. — arXiv: 2410.01066 [cs]. — URL: http://arxiv.org/abs/2410.01066 (visited on 06/14/2025). — Pre-published.

37. Exploring the Landscape of Text-to-SQL with Large Language Models: Progresses, Challenges and Opportunities / Y. Huang [и др.]. — Вер. 1. — 28.05.2025. — DOI: 10.48550/arXiv.2505.23838. — arXiv: 2505.23838 [cs]. — URL: http://arxiv.org/abs/2505.23838 (дата обр. 15.06.2025). — Пред. пуб.

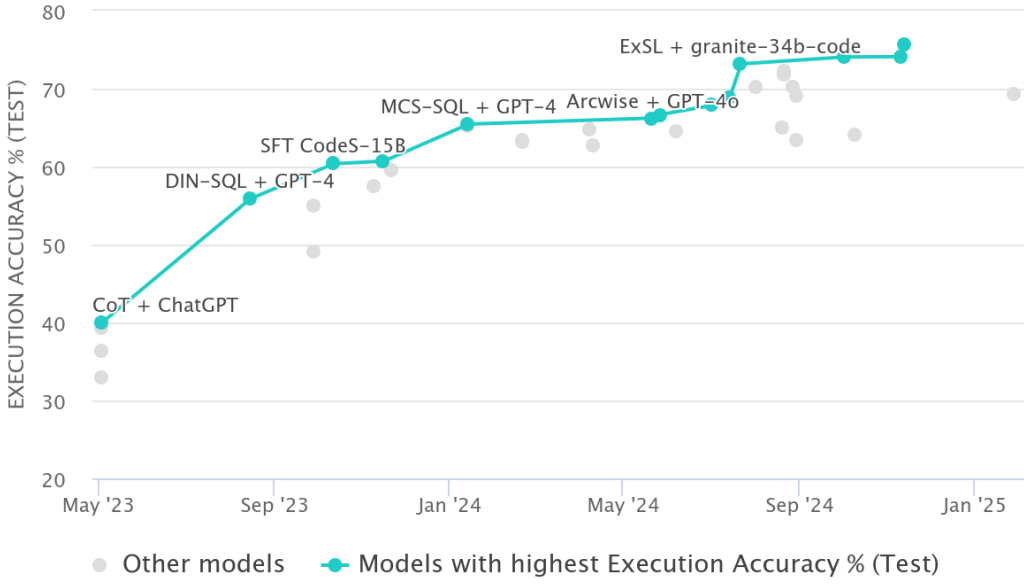
38. A Preview of XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL / Y. Gao [и др.]. — Вер. 3. — 10.02.2025. — DOI: 10.48550/arXiv.2411.08599. — arXiv: 2411.08599 [cs]. — URL: http://arxiv.org/abs/2411.08599 (дата обр. 15.06.2025). — Пред. пуб.

Приложения

Приложение А. Показатели XiYan-SQL в ключевых бенчмарках задачи Text-to-SQL



(а) Бенчмарк Spider



(в) Бенчмарк BIRD

Рисунок А.1 – Графики лидеров на бенчмарках Spider и BIRD

Приложение Б. Фрагменты программного кода

Листинг Б.1 – Точка входа в приложение (`main.py`)

```
from fastapi import FastAPI, Request, HTTPException
from fastapi.staticfiles import StaticFiles
from fastapi.responses import FileResponse, RedirectResponse, HTMLResponse
from fastapi.middleware.cors import CORSMiddleware
from contextlib import asynccontextmanager
import os

from api.v1.api import api_router
from core.middleware import CacheBustingMiddleware
from core.config import settings
from db.base import Base
from db.session import engine


def create_tables():
    print("Creating tables...")
    Base.metadata.create_all(bind=engine)
    print("Tables created.")


@asynccontextmanager
async def lifespan(app: FastAPI):
    # Code to run on startup
    avatars_dir = os.path.join(settings.UPLOADS_DIR, "avatars")
    os.makedirs(avatars_dir, exist_ok=True)
    # create_tables()
    yield
    # Code to run on shutdown


def create_app() -> FastAPI:
    app = FastAPI(title="Text-to-SQL Service", lifespan=lifespan)

    app.add_middleware(CacheBustingMiddleware)

    app.add_middleware(
        CORSMiddleware,
        allow_origins=["*"],
        allow_credentials=True,
        allow_methods=["*"],
        allow_headers=["*"],
    )


   app.mount("/static", StaticFiles(directory="static"), name="static")

   app.mount("/uploads", StaticFiles(directory="uploads"), name="uploads")
```

```
    @app.get("/uploads/avatars/{filename:path}")
    async def get_avatar(filename: str):
        file_path = f"uploads/avatars/{filename}"
        if not os.path.exists(file_path):

    raise HTTPException(status_code=404, detail="File not found")

        headers = {
            "Cache-Control": "no-cache, no-store, must-revalidate",
            "Pragma": "no-cache",
            "Expires": "0",
        }
        return FileResponse(file_path, headers=headers)

    app.include_router(api_router, prefix="/api/v1")

    @app.get("/login")
    async def read_login():
        return FileResponse("templates/login.html")

    @app.get("/queries", response_class=HTMLResponse)
    async def queries_page(request: Request):

    return HTMLResponse(content=open("templates/index.html").read())

    @app.get("/help", response_class=HTMLResponse)
    async def help_page(request: Request):
        return HTMLResponse(content=open("templates/help.html").read())

    @app.get("/settings", response_class=HTMLResponse)
    async def settings_page(request: Request):

    return HTMLResponse(content=open("templates/settings.html").read())

    @app.get("/")
    async def read_root():
        return RedirectResponse(url="/queries")

    return app


app = create_app()
```

Листинг Б.2 – Реализация эндпоинта регистрации пользователя (features/users/api.py)

```
from fastapi import APIRouter, Depends, HTTPException, status
from sqlalchemy.orm import Session
from typing import Any
import re

from features.users import crud
from features.users.schemas import User, UserCreate
from core.deps import get_db

router = APIRouter()

@router.post("/register", response_model=User)
def register(
    *,
    db: Session = Depends(get_db),
    user_in: UserCreate,
) -> Any:
    """
    Create new user and return it.
    """
    # Basic validation for username and password length
    if not (3 <= len(user_in.username) <= 20):
        raise HTTPException(
            status_code=422,
            detail="Username must be between 3 and 20 characters.",
        )
    if len(user_in.password) < 8:
        raise HTTPException(
            status_code=422,
            detail="Password must be at least 8 characters.",
        )
    if not re.match(r"^[a-zA-Z0-9_]+$", user_in.username):
        raise HTTPException(
            status_code=422,

   detail="Username can only contain alphanumeric characters and " \
            "underscores.",
        )

    user = crud.user.get_by_username(db, username=user_in.username)
    if user:
        raise HTTPException(
            status_code=400,

   detail="The user with this username already exists in the system.",
        )

    user = crud.user.create(db, obj_in=user_in)
    return user
```

Листинг Б.3 – Реализация эндпоинта загрузки таблицы (features/tables/api.py)

```
from fastapi import APIRouter, Depends, UploadFile, File, status
from sqlalchemy.orm import Session

from core.deps import get_db, get_current_active_user
from features.users.models import User
from features.tables.schemas import Table
from services import table_service

router = APIRouter()

@router.post("/upload", response_model=Table,
             status_code=status.HTTP_201_CREATED,
             )
async def upload_table_file(
    db: Session = Depends(get_db),
    file: UploadFile = File(...),
    current_user: User = Depends(get_current_active_user),
):
    """
    Handle the upload of a .csv or .xlsx file, save it, and create a
    corresponding entry in the database for the user's table.
    """
    return await table_service.process_and_save_table(
        db=db, file=file, user=current_user
    )
```

Листинг Б.4 – Реализация сервисной функции для получения превью таблицы (services/table_service.py)

```
import pandas as pd
from fastapi import HTTPException, status
from sqlalchemy.orm import Session

from features.tables import crud

def get_table_preview(db: Session, table_id: int, user_id: int) -> dict:
    table = (
        db.query(crud.Table)

   .filter(crud.Table.id == table_id, crud.Table.user_id == user_id)
        .first()
    )
    if not table:
        raise HTTPException(

   status_code=status.HTTP_404_NOT_FOUND, detail="Table not found"
        )

    try:
        if table.file_path.endswith(".csv"):
            df = pd.read_csv(table.file_path)
        elif table.file_path.endswith(".xlsx"):
            df = pd.read_excel(table.file_path)
        else:
            return {"error": "Unsupported file format for preview."}

        preview = df.head(5).to_dict(orient="records")
        columns = df.columns.tolist()

   return {"preview": preview, "columns": columns, "total_rows": len(df)}

    except Exception as e:
        raise HTTPException(status_code=500,
                            detail=f"Error reading table file: {e}")
```

Листинг Б.5 – Реализация модуля-эмулятора ядра NLIDB (services/text_to_sql_service.py)

```
def convert_text_to_sql(natural_language_query: str) -> str:
    """
    Placeholder function to convert a natural language query to SQL.
    In the future, this will be replaced with a call to the real
    XiYan-SQL model.
    """
    print(f"Received query: {natural_language_query}")

    # Simple hardcoded logic for demonstration
    if "users" in natural_language_query.lower():
        return "SELECT * FROM users;"
    elif (
        "products" in natural_language_query.lower()
        and "count" in natural_language_query.lower()
    ):
        return "SELECT COUNT(*) FROM products;"
    else:
        # A default, more complex query to show potential
        return \
            """
            SELECT name, price
            FROM products
            WHERE category = 'electronics'
            ORDER BY price DESC;
            """
```