Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Физико-механический институт

Работа допущена к защите
Должность руководителя
_____ М. Е. Фролов
«_____» _____ 202_ г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ АВТОМАТИЗАЦИЯ ПРОВЕДЕНИЯ ИССЛЕДОВАНИЙ С ИСПОЛЬЗОВАНИЕМ МУЛЬТИАГЕНТНЫХ СИСТЕМ

по направлению подготовки 01.04.02 Прикладная математика и информатика
Направленность (профиль) 01.04.02_02 Математические методы анализа и визуализации данных

Выполнил
студент гр. 5040102/30201 А.С. Сачук

Руководитель
Старший преподаватель,
кандидат ф.-м. наук, звание В.С. Чуканов

Консультант
кандидат ф.-м. наук Е.И. Пчицкая

Консультант
по нормоконтролю Л.А. Арефьева

Санкт-Петербург — 2025

РЕФЕРАТ

На 1 с., 10 рисунков, 1 таблицу, 0 приложений

КЛЮЧЕВЫЕ СЛОВА: МУЛЬТИАГЕНТНЫЕ СИСТЕМЫ, АВТОМАТИЗАЦИЯ ЗАДАЧ ЛАБОРАТОРИЙ, ПРИМЕНЕНИЕ ВНУТРЕННИХ ИНСТРУМЕНТОВ, ПРИМЕНЕНИЕ ИНСТРУМЕНТОВ РАЗЛИЧНЫХ ТИПОВ, ОПТИМИЗАЦИЯ КОНТЕКСТА, МУЛЬТИМОДАЛЬНЫЙ АНАЛИЗ.[1]

Тема выпускной квалификационной работы: «Автоматизация проведения исследований с использованием мультиагентных систем»[2].

В данной работе изложена сущность подхода к созданию динамического информационного портала на основе использования открытых технологий Apache, MySQL и PHP. Даны общие понятия и классификация IT-систем такого класса. Проведен анализ систем-прототипов. Изучена технология создания указанного класса информационных систем. Разработана конкретная программная реализация динамического информационного портала на примере портала выбранной тематики...[3]

В данной работе изложена сущность подхода к созданию динамического информационного портала на основе использования открытых технологий Apache, MySQL и PHP. Даны общие понятия и классификация IT-систем такого класса. Проведен анализ систем-прототипов. Изучена технология создания указанного класса информационных систем. Разработана конкретная программная реализация динамического информационного портала на примере портала выбранной тематики...

ABSTRACT

1 pages, 10 figures, 1 tables, 0 appendices

[1]Всего **слов**: от 3 до 15. Всего **слов и словосочетаний**: от 3 до 5. Оформляются в именительном падеже множественного числа (или в единственном числе, если нет другой формы), оформленных по правилам русского языка. *Внимание! Размещение сноски после точки является примером как запрещено оформлять сноски.*

[2]Реферат **должен содержать**: предмет, тему, цель ВКР; метод или методологию проведения ВКР; результаты ВКР; область применения результатов ВКР; выводы.

[3]ОТ 1000 ДО 1500 печатных знаков (ГОСТ Р 7.0.99-2018 СИБИД) на русский или английский текст. Текст реферата повторён дважды на русском и английском языке для демонстрации подхода к нумерации страниц.

KEYWORDS: MULTI-AGENT SYSTEMS, LABORATORY TASK AUTOMATION, APPLICATION OF INTERNAL TOOLS, APPLICATION OF DIFFERENT TYPES OF TOOLS, CONTEXT OPTIMIZATION, MULTIMODAL ANALYSIS.

The subject of the graduate qualification work is «Research automation using multi-agent systems».

In the given work the essence of the approach to creation of a dynamic information portal on the basis of use of open technologies Apache, MySQL and PHP is stated. The general concepts and classification of IT-systems of such class are given. The analysis of systems-prototypes is lead. The technology of creation of the specified class of information systems is investigated. Concrete program realization of a dynamic information portal on an example of a portal of the chosen subjects is developed...

In the given work the essence of the approach to creation of a dynamic information portal on the basis of use of open technologies Apache, MySQL and PHP is stated. The general concepts and classification of IT-systems of such class are given. The analysis of systems-prototypes is lead. The technology of creation of the specified class of information systems is investigated. Concrete program realization of a dynamic information portal on an example of a portal of the chosen subjects is developed...

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Первая половина 20-ых годов в мире машинного обучения уже ознаменовалось как веха больших фундаментальных моделей. Наиболее почетное место среди них занимают генеративные текстовые модели: с выходом моделей BERT и семейства GPT они значительно повлияли не только на исследования в области машинного обучения, но и на привычную бытовую жизнь, технологические процессы как в личных, так и в индустриальных подходах. Не обошло стороной это веяние и исследовательсие организации - действительно, способность моделей иметь некоторое, хоть местами и поверхностное, но представление о разных областях уже позволяет вовлекать GenAI в научные процессы.

Этому свидетельствует значительное число работ, посвященные интеграциям больших языковых моделей (или БЯМ, LLM) в различные виды деятельности ученых. Уже сейчас существуют направления исследований, направленных на *автоматизацию и упрощение различных процессов* лабораторий:

- автоматизация поиска статей и их анализа;
- консультация в формате диалога по поиску решений на вопросы узкоспециализированного или междисциплинарного профиля;
- развитие понимания узкоспециализированных областей;
- автоматическая генерация корректного научного текста;
- автоматизация поиска оптимального решения задач при помощи кодовых генераций;

Стоит отметить, что присутствуют и работы в области создания систем автоматизации полного исследовательского цикла, но они носят узкоспециализированный характер: они создаются только в рамках исследований методов машинного обучения и их непосредсвенных приложений.

Исследование и развитие автоматизаций процессов исследовательских групп имеет большую актуальность: автоматизация различных процессов, которые требуют наименьшей дополнительной валидации или не требуют её вообще, позволяют быстрее выполнять исследовательские задачи. Кроме того, автоматизация невозможна без обучения и снабжения LLM специфическими знаниями - это может привести к галлюцинациям и неожиданным в плохом смысле слова результатам.

Куда более обделенным направлением работ в исследовании автоматизаций является исследование интеграции БЯМ во внутренние ресурсы и процессы лабораторий, а также практическая разработка и тестирование таких систем. Дей-

ствительно, интеграция БЯМ во внутренние процессы сталкивает со следующими проблемами:

- Отсутствие единых стандартов работ в лабораторий: хотя многие молодые лаборатории стараются перенимать принципы организаций с индустрии, большинство исследовательских групп имеют собственное, зачастую непопулярное видение организаций различных процессов (хранения данных, логирование экспериментов и т.д), которые могут быть незнакомы БЯМ. Это приводит к малой универсальности предлагаемых решений;
- Наличие внутренних разработок: многие лаборатории используют свои внутренние специфические инструменты, библиотеки и фреймворки, информация о которых могла быть или не представлена в обучающих выборках LLM, или быть малорепрезентативна;
- Мультимодальность объектов исследования: не смотря на то, что объекты исследований лабораторий могут иметь различную модальность, создатели и исследователи систем автоматизаций на базе LLM предлагают решения, работающие исключительно с текстами, не оставляя в системах места для мультимодальных данных.

Целью данной работы ставится разработка системы с пользовательским интерфейсом, которая решает автоматизирует процессы пользователя, поддерживает различные форматы инструментов лабораторий, а также способна поддерживать интеграцию процессов работ с данными разнличной модальности.

ГЛАВА 1. ОПРЕДЕЛЕНИЯ И ПОСТАНОВКА ЗАДАЧИ

В данной главе представлено формальное определение решаемой задачи, которое поможет не только лучше понимать суть решаемой проблемы, но и определит возможные сценарии её решения. Однако невозможно сформулировать задачу, не понимая базовых понятий, из которых она состоит. С этих понятий и начнем.

1.1. Формальные определения

1.1.1. Определение агента

Агент - программа, самостоятельно выполняющую поставленную пользователем задачу, используя предоставленные ей инструменты и основываясь на состоянии окружения: инструменты для оценки состояния окружения принято называть *сенсорами*, а инструменты для работы и взимодействия со средой - *актуаторами* [1].

Стоит отметить, что в данное понятие не вкладываются критерии об “интеллектуальности” - хотя это важно: **рациональным (или интеллектуальным) агентом** принято называть агента выполняющий задачу “эффективно”, “наилучшим образом”. “Наилучшее” решение определяется уже вводимыми разработчиками метриками. В рамках текущей работы будем использовать эти два термина как синонимы.

Такое широкое определение позволяет формализовать понятие агента разными способами, каждое из которых остается не только верным, но и удобным для исследования в определенных прадигмах. Так, например, одним из самых популярных формальных определений агента, которое часто используется в области обучения с подкреплением (Reinforcment learning), описывается такой тройкой [2]:

$$Agent = \{\mathcal{P}, \mathcal{A}, \mathcal{S}\} \tag{1.1}$$

Определим понятия тройки 1.1:

- $Agent$ - непосредственно агент;
- $\mathcal{P} : (s_t \times ... \times s_{t-k}) \rightarrow a, s_i \in \mathcal{S}, a \in \mathcal{A}, k \in 0..t$ - политика (Policy), формирующие процесс принятия решения агентом;
- $\mathcal{A}$ - набор допустимых действий агента, актуаторы;

– $\mathcal{S}$ - окружение;

– s_t - t-ое состояние окружения: формируется после t действий агента.

На практике различные методы обучения политик подразумевает достаточно точную и строгую формулировку окружения, поскольку по ней выбирается оптимальная политика. В нашей задаче это свойство - недостаток, поскольку окружение в лабораториях может быть очень вариативным, что затрудняет формализацию.

В подобных случаях можно использовать LLM с указанием инструкций в качестве политики. Это понятно почему так: большие модели обучаются на больших массивов данных из Интернета и других источников, которые содержат в том числе и описательные инструкции как оптимально решать типовые задачи. Важным ещё является тот факт, что именно указание инструкций в виде входного текста формирует то, как именно LLM будет решать поставленные ей задачи [3—5], такое явление называется *“промптингом”*. Использование LLM-агента вместо строгоформулируемой политики позволяет отойти от строгого описания окружения - теперь окружение описывается в виде текста, подаваемого на вход модели. Такое описание окружения вместе с инструкцией принятия решений и доступными действиями для решения формируют *’контекст’ решаемой задачи*: именно через контекст модель понимает, как ей действовать и чем руководствоватся в решении переданной задачи.

Теперь осталось определить, что такое *мультиагентная сеть*.

1.1.2. Определение мультиагентная система

Мультиагентная система - система, состоящая из нескольких агентов, работающих в совокупности для достижения как глобальной, так и общей цели. Система определяется не только агентами, входящих в нее, но и регламентами их взаимодействия.

Формально мультиагентную сеть можно определить следующим образом:

$$MAS = \{Agent_i, i \in \mathcal{N}; C\} \tag{1.2}$$

где:

– MAS - multi-agent system, непосредственно определяемая система;

– $Agent_i$ - агенты-участники системы;

– $C : Agent_i \text{ x } Agent_j \rightarrow bool$ - политика взаимодействия: устанавливает наличие связи между агентами $Agent_i$ и $Agent_j$, $i, j \in \mathcal{N}$.

Иногда между агентами требуется не только наличие связи для взаимодействия, но и определить строго тип взаимодействий: в таком, более общем случае $bool$ заменяется на некоторый тип, который имеет свою область определения. В рамках работы мультиагентных систем на базе LLM таким типом является *текст ('string')* - конечная последовательность символов некоторого алфавита Σ, который, на момент написания работы, как правило состоит из символов латиницы, цифр, знаков препинания и спецсимволов.

Стоит уделить особое внимание популярному заблуждению, которое возникло за последнее время вместе с популяризацией агентов и мультиагентных систем на основе LLM: часто "агентами" называют программные решения, которые не позволяют моделям самостоятельно решать как именно действовать. Вместо этого такие системы используют генерацию больших языковых моделей в качестве некоторого промежуточного решения, которое уже используют в заранее определенном программно алгоритме [6]. В таких системах нет момента с "интеллектуальным" определением дальнейшего действия, и такие системы не могут называться "интеллектуальными агентами".

Дав определение необходимым терминам, мы можем приступить к постановке задачи.

1.2. Постановка задачи автоматизации процессов при помощи MAS

Необходимо реализовать программное решение, обеспечивающий возможность работы пользователя с мультиагентной системой MAS для автоматизации решения различных типовых задач, требующих как использование внтреннего инструментария лабораторий $\mathcal{A}_{lab}$, так и учитывающих состояния имеющегося окружения $\mathcal{S}_{lab}$, с которым этот инструентарий может взаимодействовать:

$$MAS = \{Agent_i, i \in \mathcal{N}; C\}; Agent_i = \{\mathcal{P}_i, \mathcal{A}_i, \mathcal{S}_i\} \tag{1.3}$$

где:

- $\mathcal{P}_i$ - LLM-политика агента $Agent_i$, сформированная видом модели и её контекстами;
- $\mathcal{A}_i$ - набор действий, доступный агенту $Agent_i$, притом важно, что $\mathcal{A}_{lab} = \bigcup_{i=1}^{N} \mathcal{A}_i$;
- $\mathcal{S}_i \in \mathcal{S}_{lab}$ - часть окружения $\mathcal{S}_{lab}$, с которым необходимо работать инструментарию $\mathcal{A}_i$.

Для постановки задачи также необходимо сформировать требования к инструментарию лабораторий. В рамках рассматриваемой задачи рассматриваются следующие возможные инструменты:

- функции для работы с текстом и числами: это обычные инструменты–функции, которые принимают на вход аргументы в виде текста или чисел, и которые возвращают также или текст, или число;
- функции, которые работают с данными отличные от текста и чисел;
- отдельные фрагменты программных кодов в виде функций, классов и модулей, объединенные в группы для совместной работы с целью решения опеделенной задачи (например, модуль для обработки специфических данных, класс для отрисовки данных и так далее);
- готовые целиковые библиотеки, фреймворки, пакеты: как внутренние, так и внешние.

Данный набор инструментов охватывает широкий спектр возможных существующих действий: они неограничены на тип входных и выходных данных, что позволяет более гибко взаимодействовать с окружением. Сами инструменты могут быть как отдельными функциями, так и частями кодовых баз, так и кодовыми базами в виде библиотек целиком. Могут функции работать как с локальной частью окружения, находящимися в той же среде, что и агент (например, с файлами), так и с удаленными окружениями посредством обращения через интерфейсы [4].

ГЛАВА 2. ОБЗОР ЛИТЕРАТУРЫ

Ввиду постановки задачи, текущая глава посвящена обзору существующих решений сразу двух направлений:

- инструментов в LLM-агентах;
- автоматизации решении задач.

Начнем главу с первого направления.

2.1. Обзор существующих реализаций инструментов

Прежде чем говорить о действиях, которые могут совершать LLM-агенты и MAS, нужно обозначить то, как выглядят эти действия. Стоит понимать, что

[4] возможно, стоит переписать

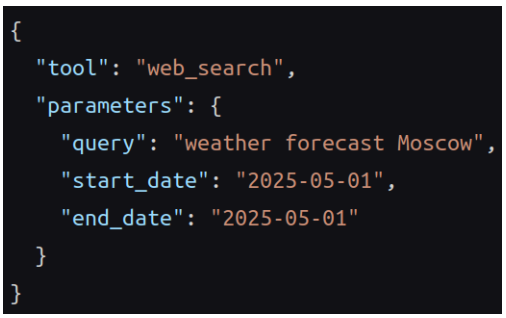

Рис.2.1. Пример вызова LLM при запросе узнать погоду в Москве: ’tool’ - название инструмента, ’parameters’ - аргументы инструмента.

все действия - это вызов некоторой функции. Но вот то, как именно происходят вызовы - будет далее изложенно в этой секции.

2.1.1. Определение классических вызовов инструментов

Вызов инструментов, ровно как и работа LLM в формате чат-модели - свойство, достигаемое обучению моделей на синтетических данных, которые внешне сильно отличаются от тех, которые можно встретить как в интернете, так и в различных специфических источниках (например, разметках веб-страниц): примером такой разметки служит ChatML [7]. В действительности, свойство модели “поддерживать” разговор в формате диалога, “вызывать” инструмент - некоторого рода договорённость, по которой и модель понимает, как ей генерировать более точно ответ, так и разработчики понимают, как обрабатывать результат. Именно эта договоренность позволяет использовать LLM в качестве чат-бота или агента.

Стоит отметить, что вид разметки вызова инструмента можно формировать при помощи дополнительного описания в контексте, однако одним из часто испльзуемых форматов все же остается JSON схема. Примером этого служит изображение рис.2.1. Это пример вызова инструмента и передаваемых параметров в специальной JSON разметке.

Такие модели, в отличие от обычных моделей, которые лучше понимают, как вызывать инструменты, как обрабатывать некоторого рода запросы, носят специальные названия: например, суффикс ’-instruct’. Это важно отметить ввиду того, что от выбора модели зависит качество работы всей мультиагентной сети в целом.

Как понятно из примера и теории выше, все аргументы и возвращаемые типы при таких вызовах функций - данные текстовопредставимые: строки и числа. Это накладывает ряд ограничений на инструменты: в такие функции нельзя передавать

в явном виде типы, которые в явном виде не представимы в виде строк (например, изображения, видео), ровно как и функции - должны возвращать строки. Но можно передавать их неявно: например, через пути к файлам - в некоторых сценариях это допустимо.

Стоит отметить, что попытки передать изображения и видео в явном виде функциям при помощи LLM (например, в формате строки байт-кода) провальна: как минимум из-за того, что символы, входящие в байт-код, могут не поддерживаться LLM. Максимально приближенным способом работы с данными в условиях, когда они не могут быть предствлены в виде текста, можно использовать различные большие фундаментальные модели, работающие с данными модальностями: например, VLM для работы с изображениями [8]. Такие модели, не смотря на некоторые неточности, успешно проявляют себя в решении ряда задач [9].

Последним, хоть и незначительным, но недостатком, является практическая особенность реализаций таких инструментов: хоть концептуально, вызовом действия может допускаться вызов любой функции, в большинстве случаев многие полезные функции, которые были бы актуальны для агента, невозможно вызвать напрямую ввиду разногласий в реализациях интерфейсов. В таких случаях можно воспользоваться одним из двух вариантов:

- реализовать адаптер для вызова метода с несовместимым для LLM интерфейсом;
- использовать готовый интерфейс коммуникации, например: MCP.

О последнем далее и пойдет речь.

2.1.2. Определение MCP-инструментов

Не смотря на недостатки классических LLM-инструментов, их свойств достаточно для их применения в широком круге задач. Этому соответствуют множество косвенных факторов, но одним из них является выпущенная Anthropic унифицированный протокол взаимодействия с инструментами ModelContextProtocol (или MCP) [10].

Вкратце, данный протокол позволяет модели обращаться и работать не только с внешними инструментами, но и с различными абстракциями в целом: это могут быть другие агенты и мультиагентные системы, источники знаний для генераций LLM-ответов с привлечением внешних источников (Retrieval Augmented

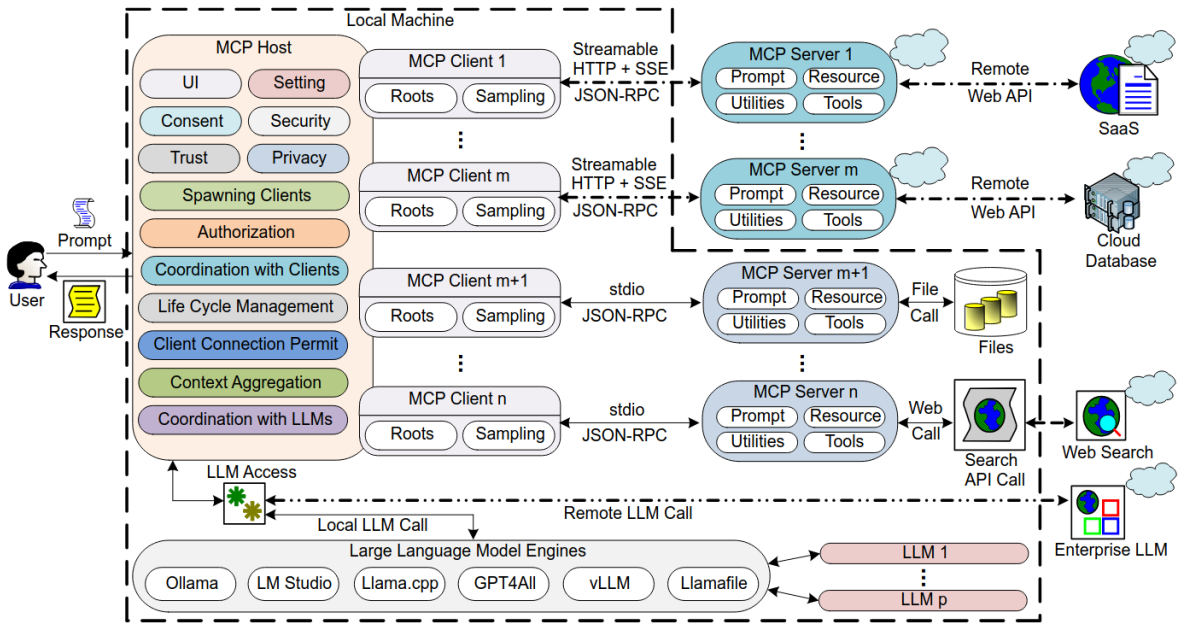

Рис.2.2. Полный цикл работы MCP протокола с LLM.

Generation или RAG) и многое другое. Нам же это интересно в первую очередь интерфейс из-за его возможности работать с инструментами.

На рис.2.2 из [11] представлен общий план агентной системы, работающей с инструментами MCP. Как видно из рисунка, для реализации подключения LLM-агента к некоторому инструментарию или окружению через интерфейсы MCP необходимо реализовать MCP-клиент и MCP-сервер.

Такая пара для каждого инструментария своя: она обеспечивает взаимодействие между LLM-агентом и инструментами. Все MCP-клиенты размещается на том же устройстве, на котором работает LLM-агент. MCP-сервера, напротив, не обязаны размещаться на том же устройстве, что и клиент: они могут быть как на локальном вычислительном устройстве и работать через stdio, так и удаленнымы, и работать через HTTP запросы. Это деталь важна для нас не с технической точки зрения, а с практической: MCP интерфейсы предлагают *простое и гибкое* взаимодействие с окружениями разного уровня доступности: с локальными и удаленными.

Список открытых инструментов постоянно пополняется и его можно наблюдать в открытых источниках: например, в официальном GitHub репозитории MCP [12].

Не взирая на все удобства MCP, данный интерфейс не устраняет недостатков для работы с данными других типов. Именно потому стоит рассмотреть другой вариант работы с инструментами.

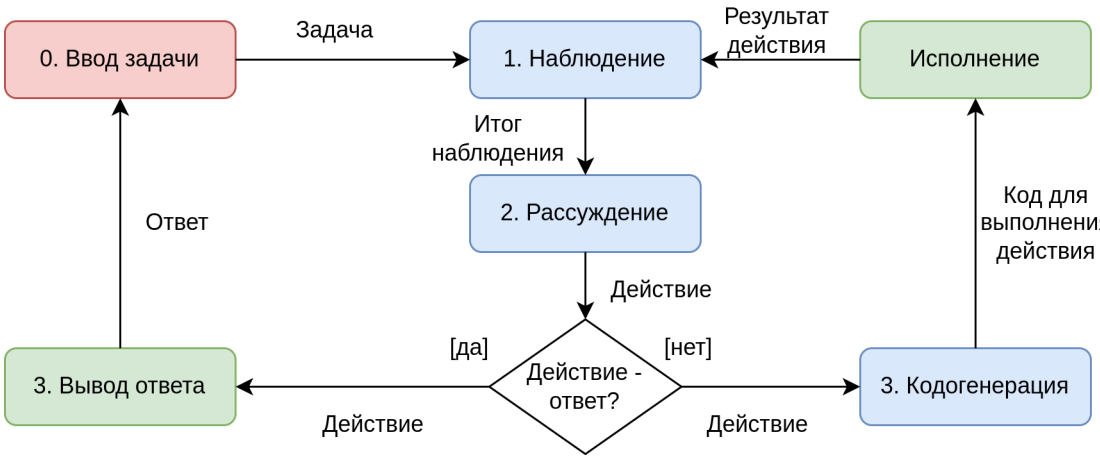

Рис.2.3. Принцип работы CodeAct: вызов инструмента (действия) происходит в формате кодогенерации на определенном языке программирования. Красные блоки - действия, выполняемые пользователем; синие - LLM; зеленые - кодом программы.

2.1.3. Определение кодовых инструментов

Для работы с инструментами, требующие в явном виде данные отличных от текста модальностей, можно воспользоваться свойством LLM-агентов генерировать код и пытаться исполнять его: такое решение предложено авторами в статье CodeAct [13], к которому еще вернемся. В данной работе предлагается генерировать вместо специфичного текста-разметки для выбора некоторого действия (или набора действий) сразу код, в котором могут присутствовать те же вызовы инструментов, но уже имеющие синтакс языка генераций. Процесс работы агента с кодогенерацией и инструментами представлен на рис рис.2.3.

Такой подход предоставляет сразу несколько плюсов:

– работа с различными типами данных: теперь диапазон допустимых инструментов не ограничивается модальностями аргуемнтов и возвращаемого значения;
– наличие логики в вызове инструментов: генерируемый код может иметь условные операторы, циклы, а также обрабатывать исключения - при условии если это все поддерживает язык генерации кода.

У такого метода есть два недостатка:

– *безопасность*: хоть и существуют исследования касательно того, как делать генерацию и исполнения кода более безопасной, нет гарантий, что код не окажется вредоносным;
– *мультимодальность*: такой подход позволяет вызывать инструменты, работающий с отличным от текстом типами данных, но в предлагаемом

решении кодогенерация всё еще обязана генерировать только текст в качестве результата работы кода (например: вывод в терминал).

2.1.4. Проблема большого количества инструментов

Одна из проблем, с которой можно столкнуться в процессе разработки агентов - большое количество инструментов. Эта проблема возникает из-за того, что всю подробную информацию о названиях, свойствах и решаемых задачах инструментов необходимо указывать при запросах каждом запросе больших языковых моделей. Это вызывает сразу две проблемы:

- увеличенный расход вычислительных мощностей: современные LLM основаны на базе архитектуры “трансформеров”, которые производят матричные перемножения, одна из размерностей которых фиксирована, а другая - имеет длину входного текста. Потому эти модели имеют квадратичную сложность $O(N^2)$ от длины входного запроса (эта сложность или временная [14], или по памяти за счет кэширования и оптимизаций[15] - зависит от реализации).
- ухудшение рассуждений: большое количество инструментов ухудшают рассуждение моделей, направленных на решение задачи.

Данная проблема решается различными способами. Самый действенный из них - управление контекстом модели при каждом запросе: в зависимости от типа запроса пользователя в контекст передается информация о тех инструментах, которые наиболее релевантны.

Отходя от темы выбора инструментов и рассуждая в общих практиках модерации контекста, семейство техник по управлению загрузки дополнительного контекста из источников называется retrieval augmented generation (RAG). Оно насчитывает разные способы оценки релевантности и процеды выбора информации для контекста, но стоит отметить два популярных из них:

- *оценка на основе косинусного сходства (cos-sim)* [16]: между запросом и всеми текстовыми сводками с дополнительной информацией считается метрика косинусальной близости, при этом k самых больших значений подгружается на вход в контекст модели;
- *оценка на основе предположения модели (HyDE)* [17]: перед расчетом метрик близости запроса с дополнительной информацией, предлагается обогатить запрос пользователя дополнительным гипотетическим отве-

том LLM. Данная идея позволяет местами улучшать точность выбора фрагментов за счет внутренних знаний и рассуждений LLM.

2.2. Обзор решений автоматизаций.

Перед обзором способов автоматизаций решений задач, необходимо вкратце рассмотреть одно из свойств мультиагентных систем. Очевидно, что мультиагентные системы могут поддаваться различным классификациям, но одна из них фигурирует наиболее часто в исследованиях, посвященных генерации MAS - классификация по принципу уровня автономности. Принято выделять два класса агентов и мультиагентных систем:

- полностью автономные (или agent-driven системы) - системы, в которых политики LLM-агентов целиком и полностью формируют принятие решений;
- частично автономные (или workflow-driven системы) - системы, в которых агенты только частично вовлечены в процесс решения поставленной задачи: другая часть процесса предусмотрена и алгоритмически определена разработчиком.

Оба метода имеют свои плюсы и минусы. Частично-автономные агенты имеют меньший диапазон решений и действий, в сравнении с полностью автономными LLM-агентами, однако это не является недостатком в случае, когда сценарии решении проблем можно обобщить и программно упростить. К тому же, таким методы легче контролировать. Полностью-автономные агенты конечно более сложные в тестировании и создании условий для схождения к успешным решениям, однако они имеют куда более высокую степень свободы, что особенно важно в случаях, когда спектр возможных задач трудно формулизуем. К таким случаям и относится поставленная задача.

Кроме того, как будет показано далее, все больше набирают популярность в исследованиях направления улучшения рассуждающих способностей моделей [18; 19]. Именно по этим причинам далее в большей степени будет отдаваться предпочтение *полностью автоматизированным MAS*.

2.2.1. Обзор вариантов agent-driven систем.

Одним из вариантов agent-driven ситемой является ReAct [20] (от англ. reasoning and acting). Данная система работает по принципу выполнения цикла “Рассуждай”/“Действуй”/“Наблюдай” (“Thought”/“Action”/“Observation”).

На этапе рассуждений модель использует внутренние выученные рассуждения того, как решать задачу и какой инструмент стоит применить: для извлечения рассуждений часто используется техника “Chains-of-thoughts”, которая на практике чаще приводит к достижению поставленной цели [4]. На этапе действия происходит вызов инструмента и добавление результата его работы. Уже, наконец, на последнем этапе наблюдений LLM-модель рефлексирует на предмет того, какую информацию она получила. Такой цикл заканчивается, когда модель принимает решение о готовности ответа.

Формально, данная система является агентом, а не MAS, но ничто не мешает использовать вместо инструментов других агентов, создавая иерархическую MAS.

ReAct позволяет успешно решать ряд частовстречаемых задач, что сделало метод популярным для многочисленных модификаций. Одной из таковых является уже упомянутая работа CodeAct [13]. В данной работе, как было продемонстрировано ранее, происходит все то же самое, только вызов инструментов производится при помощи кодогенераций.

Для решения куда более сложных задач был разработан метод PlanAndExecute и его различные модификации [21; 22]: данный метод использует два агента - “агента-планировщика” для генерации плана и “агента-исполнителя” для выполнения подзадач плана. Работа системы проста: в начале планировщик разбивает задачи на связанные подзадачи, после чего подзадачи шаг за шагом поступают исполнителю. В случае, если решить подзадачу не удалось, планировщик создает новый план, видоизменяя текущую подзадачу и все последующие от неё зависящие. Так повторяется, пока или не будет выполнена задача, или пока не будет истрачено количество генераций плана.

Данная система показывает себя лучше в сравнении с ReAct в ряде определенных задач, однако требует куда больше времени и ресурсов. Кроме того, на момент написания работы, такие системы в автоматическом режиме не умеют строить планы в долгосрочной перспективе, однако неплохо справляются в связке с человеком [23]. Это делает такие MAS на текущий момент больше “еще одним методом”, нежели бескомпромисной альтернативой.

Рассмотрим, как на практике автоматизируют различные процессы в лабораториях.

2.2.2. Обзор существующих решений поставленной задачи.

Одной из нашумевших работ является работа AI-Scientists [24], которая показывает возможность автоматизировать процесс исследования. Авторы приводят в пример процесс генерации идеи, переходящий в цикличную валидацию и развитие имеющегося кода, и заканчивающийся автоматической генерации отчета-статьи. Схожее решение представлено и в работе AgentLaboratory [25], которая

Идеи и решения, предеставленные в работах, могут звучать грандиозно, однако эти решения имеет целый ряд недостатков:

- наличие отправного решения: необходимо предоставлять системе начальный код (например, код нейросети), которое с течением итераций будет модифицироваться с целью извлечения наблюдений;
- workflow-driven решение без инструментов: само решение работает согласно предусмотренному порядку вызовов LLM;
- высокая доля стохастичности: само решение генерирует и пытается давать метрически оценки многим генерациям, не обосновывая принцип оценивания. В таких ситуациях сложно убедиться в достоверности таких оценок;
- узкое тестирование: тестирование проводилось на задач из области машинного обучения.

Существуют решения, охватывающие куда меньшую долю задач лаборатории, но предоставляющее более качественное и валидированные решения:

- *Data Interpreter* [26]: MAS для изучения наборов данных при помощи конструирования графа задач и подзадач с последующей кодогенерацией, что делает решение более контролируемым, интерпретируемым и воспроизводимым;
- [5];
- [6];

Все эти решения выполняют какую-то группу задач, являются agent-driven системами, но при этом все они заточены на узкое применение при помощи некоторого фиксированного набора инструментов или без инструментов вовсе.

[5] пример

[6] пример

Потому далее, на основе существующих решений, будет описано предлагаемое и реализованное решение, предоставляющее возможность не только гибко подключать различные инструменты лабораторий и выполнять более общие и рутинные задачи лабораторий, но и анлизировать мультимодальные данные.

ГЛАВА 3. ОБЗОР ПРЕДЛАГАЕМОГО РЕШЕНИЯ

Данная глава посвящена обзору предлагаемого решения, описания его реализации. Перед разбором деталей, стоит дополнительно рассмотреть потенциальный инструментарий лаборатории, который необходимо подключить к MAS.

Как показывает практика, язык программирования Python является одним из самых популярных языков программирования: для лабораторных исследовательских процессов - это не исключение. Используемые инструменты, будь то внутренние реализованные алгоритмы или специфические сторонние библиотеки, реализованы, как правило, именно на этом языке.

Потому перечисленные выше способы вызова функций LLM-агентами охватывают большинство инструментов, с которыми могут работать исследователи: для инструментов, реализованных на языке программирования Python, можно использовать JSON-вызовы или возможности кодогенерации. На рис.3.1 указано отображение инструментов лабораторий с интерфейсами, через которые они могут быть подключены к MAS для автоматизации решения задач.

Остальные, редкие, но возможные отличные сценарии использования решаются путем редуцирования их к перечисленным в обзоре литературы способов вызова инструментов: так, например, в случае, если существует удаленный инструментарий, работающий через запросы или написанный на отличном языке программирования, достаточно воспользоваться MCP-интерфейсом для создания MCP-сервера и подключения к предлагаемой MAS.

Теперь рассмотрим, как реализовывалось решение, работающее с указанным набором средств подключения инструментов.

3.1. ExCodeAgent-MM: агент кодогенерации.

Для работы с кодовыми инструментами можно оттолкнуться от решения, предлагаемого в работе [13], только модифицировав его для того, чтоб агент

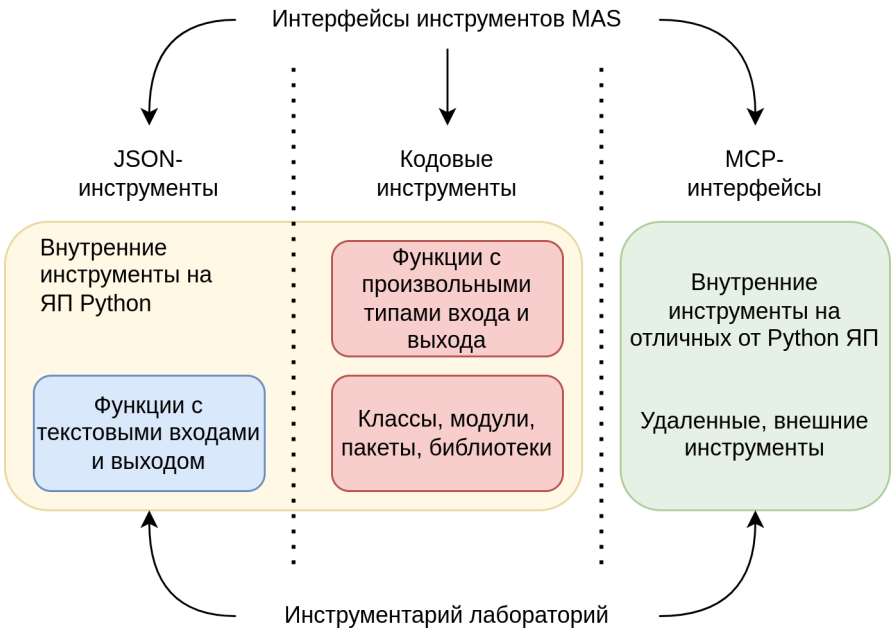

Рис.3.1. Соответствие между используемыми интерфейсами MAS и инструментами лабораторий.

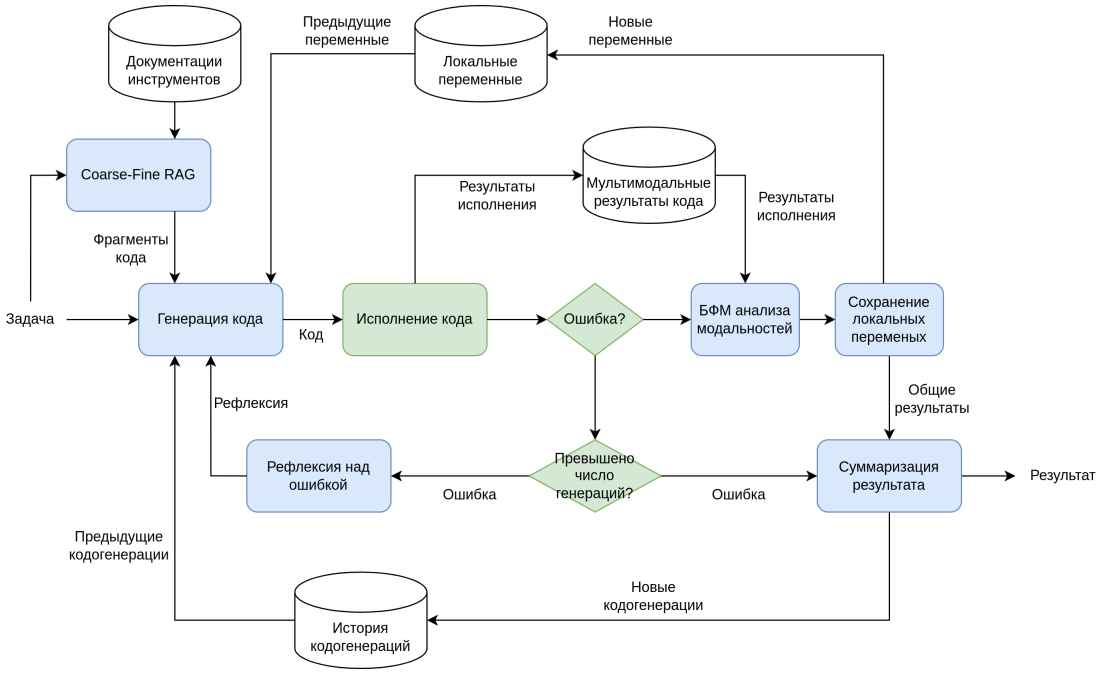

Рис.3.2. Общая структура агента кодогенерации.

кодогенерации, помимо непосредственной генерации и запуска кода, обладал еще двумя следующими свойствами:

A. работал с вариативным числом инструментов;

B. поддерживал работу с мультимодальными данными и их анализ;

С этими целями был построен агент кодогенерации, изображенный на рис.3.2. Принципы, по которому он устроен, подробно расписаны в секциях далее.

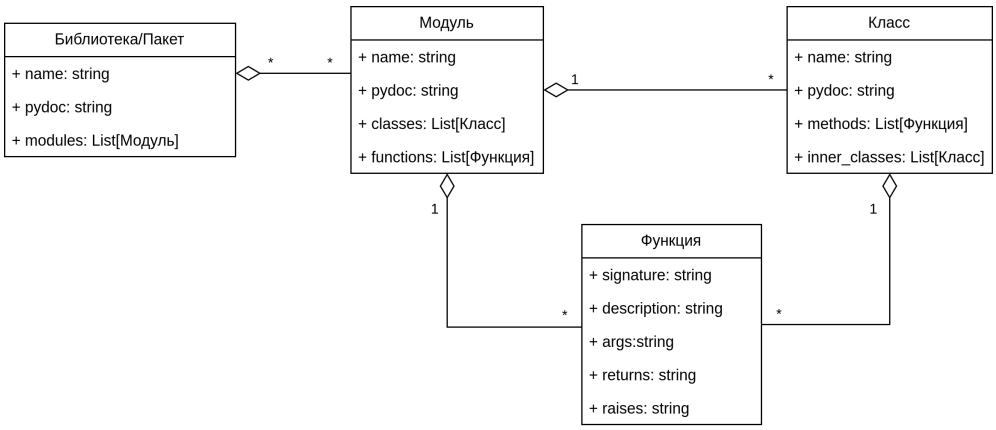

Рис.3.3. Диаграмма классов. Вложенность элементов кодовых инструментов и форма их документирования.

3.1.1. Coarse-Fine RAG: интеграция кодовых инструментов.

Для генерации кода с вариативными функциями, классами, модулями и библиотеками необходимо, как было рассказано ранее, не только добавлять информацию об этих элементах в контекст модели, но и делать это *эффективно*. Остановимся пока на том, как можно структурировать информацию.

На рис.3.3 представлена информация, как в общем виде структурируются пакеты, модули, классы, методы, как они относятся друг к другу, а также как они документируются. Как видно, у каждого элемента есть свое описание, притом у функций и методов - оно самое подробное. Самое примечательное, что это описание не требуется в каком-то специализированном для данного проекта формате или в отдельном документе - это обычные pydoc строки, которые используются для документирования и автодокументирования кода: они доступны у элементов посредством обращения к полю “.__doc__” и имеют общепризнанные стандарты [27]. Это свойство Python - чрезвычайно важно для формаировании информации об иструментах и её передачи в контекст LLM-модели для последующей генерации.

Процедура извлечения документации из кода и её структурирование происходит по принципу обхода элементов библиотек, модулей и классов в глубину: действительно, все эти элементы имеют строение дерева, по которому можно пройтись и для каждого элемента-узла сконструировать его документацию. После обхода, вся считанная документация сохраняется в упрощенной структуре, указанной на рис.3.4: имя и краткая документация задаются пользователями вручную, а полная структурированная документация в лице списка классов и функций - извлекается автоматически.

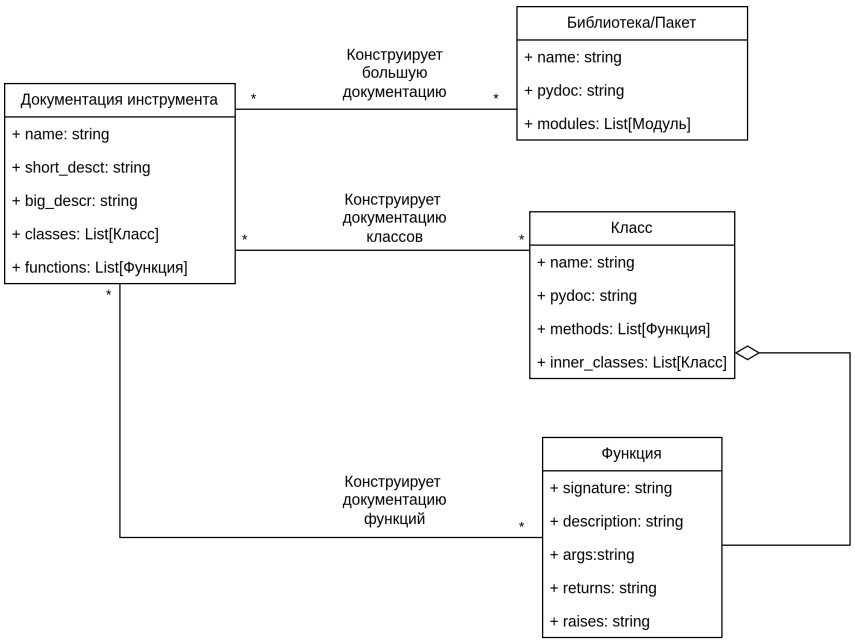

Рис.3.4. Диаграмма классов из класса-документации и классов, её формирующую.

После того, как документации всех переданных инструментов были загружены, необходимо передавать её в контекст LLM для генерации кода. Есть два варианта, как это можно сделать:

- загружать целиком всю документацию о всех инструментах в контекст в LLM вместе;
- выборочно подавать фрагменты документаций для последующей кодогенерации.

Первый вариант в условиях неограниченности ресурсов кажется выигрышным, однако LLM обладают спецификой “путаться” в выборе различных методов и классов, число которых, как правило, может быть очень большим даже при малом количестве переданных библиотек. Кроме того, не стоит и забывать, что ресурсы всё же ограничены, а модели имеют квадратичную сложность от длины контекста. Второй вариант при правильном выборе фрагментов документаций нивилирует эти проблемы: потому далее будет рассмотрен разработанный эффективный метод подгрузки фрагментов документации для кодогенераций. Он также продемонстрирован на [7].

Извлечение полезных фрагментов документаций в предлагаемом решении происходит в два этапа путём двойной фильтрации информации об инструментах:

А. *Грубая (“coarse”) фильтрация*: LLM на основе внутренних рассуждений и кратких пользовательских описаний инструментов выбирает, какие именно из инструментов будут актуальны для решения задач. По выбран-

[7] рис

ным инструментам создается “грубая” документация - документация инструмента без подробного описания функций в виде аргументов, возвращаемых типов и исключений: только их краткое описание и сигнатура.

В. *Точная (“fine”) фильтрация*: “Грубые” документации выбранных инструментов подгружается вновь в LLM для выбора нужных функций и методов для решения задачи. Выбранные функции используются для обогащения “грубой” документации подробной информацией об этих функций для последующей кодогенерации.

Такой процесс “coarse-fine” фильтрации можно рассматривать как двойное модифицированное использование HyDE, где используются такие же предположения и рассуждения модели: модификация в данном конкретном случае заключается в использовании структурированности данных в документации.

Стоит отметить, что попытки применения классических методов выбора фрагментов при помощи RAG на основе косинусной близости вместро предложенного алгоритмы могут оказаться малоэффективным: методы на основе косинусной близости работают только с векторными представлениями фрагментов и не учитывают то, *как структурированны* эти фрагменты и *что* описано во фрагменте в явном виде.

Теперь перейдем к вопросу учета и анализа мультимодальных данных.

3.1.2. Анализ мультимодльных данных

Оригинальная система CodeAct предлагает единственный способ анализа мультимодальных данных: только посредством вывода информации о данных в поток вывода. Предлагаемое решение позволяет анализировать мультимодальные двумя способами:

- посредством текстовой информации в потоке вывода, полученной при исполнении кода;
- посредством использования БФМ для анализа дополнительных модальностей.

На фоне роста числа как общих БФМ (например, для анализа инфографиков), так и специализированных БФМ [28], а также невозможности описать данные одними только аналитическими метриками, дополнительная возможность исследовать данные является скорее полезной, чем избыточной.

Эта возможность достигается за счет реализации трех элементов:

- подключения БФМ для анализа дополнительной модальности;
- создания callback-функции, которая захватывает данные из сгенерированного и сохраняет их;
- описания этой самой callback-функции, а также других дополнительных инструкций в системном промпте для кодогенерации.

Общий принцип анализа дополнительных модальностей таков: LLM-агент кодогенерации генерирует код, в котором согласно прописанным инструкциям используются callback-функции с передачей данных специальной модальности в качетве аргумента. При вызове callback-функции происходит копирование данных специальной модальности для последующей передачи БФМ модели. В случае успешного выполнения всего сгенерируемого кода - описание входной задачи, текстовый выход в потоке вывода, а также скопированные данные специальной модальности отправляются на вход БФМ для генерации текстового описания данных.

3.1.3. Генерация, выполнение и валидация кода

Генерация кода для выполнения задачи происходит с учетом переданных из Coarse-Fine RAG фрагментов необходимой документации, а также переданных пользователем списке рекомендованных популярных библиотек (например, “numpy” или “matplotlib”).

Исполнение сгенерированного кода происходит в несколько этапов:

- *проверка импортирования модулей*: посредством отдельного исполнения строк кода со словами “import” и “from” происходит проверка досягаемости используемых кодовых инструментов, а также заранее рекомендованных библиотек и фреймворков. В случае ошибки возвращается соответствующая ошибка для рефлексии и выборе других библиотек;
- *исполнение кода*: при помощи функции “exec(code, locals)” с передачей сгенерированного кода и ранее полученных при предыдущем исполнении локальных переменных происходит выполнение нового кода;
- *повторная генерация кода*: происходит в случае ошибки при исполнении. LLM передается задача, необходимые фргменты документации, последняя попытка кодогенерации с рассуждениями, а также ошибка при исполнении - всё это с инструкцией “обдумать” ошубику и попытаться выявить её причину. После рефлексии - происходит повторная генерация; Результат

рефлексии вместе со всем перечисленным выше подается в качестве входа для повторной кодогенерации с целью исправить ошибку;

– *сохранение локальных переменных и результатов кода в специальные переменные-хранилища*: в сучае успешной генерации кода новые переменные и мультимодальные результаты сохраняются для последующего использования.

Выделение специальных хранилищ необходимо по двум причинам:

– *последовательность задач кодогенерации*: поскольку задачи для кодогенерации направлены на решение общей главной задачи и все они поступают друг за другом, для решения вновь пришедшего задания необходимо учитывать контекст решения предыдуших;
– переиспользование разных модальностей: помимо анализа модальностей полученные результаты используются в других целях - в выводе в графическом интерфейсе и в формировании Jupyter Notebook'ов со сгенерированными командами и сохраненными результатами выводом.

Генерация Jupyter Notebook'ов с сгенерированными кодами и результатами исполнения в формате изображений и текста в поток вывода - разработанная в рамках задачи опция, позволяющая переиспользовать сгенерированный код для решения типовых задач, а также формирования графических документов для анализа данных.

3.2. ExCodeAct: конечная мультиагентная система для различных инструментов.

С целью создать рефлексирующую и рассуждающую модель для работы не только с кодовыми инструментами лаборатории, но и с прочими другими указанными на рис.3.1, было создана сеть, сочетающая оригинальное решение ReAct с разработанным агентом кодогенерации ExCodeAgent-MM, а также MCP инструментами. Изображение разработанной системы, названной ExCodeAct, представлено на рис.3.5.

Предлагаемая система циклично "рассуждает" над выбором инструментов для решения подзадач, вызывает инструменты, а затем оценивает результаты и выбирает новые инструменты.

Данная структура выбрана неспроста: система ReAct способна не только автоматизировать исполнение тривиальных рутинных задачи, она обеспечивая

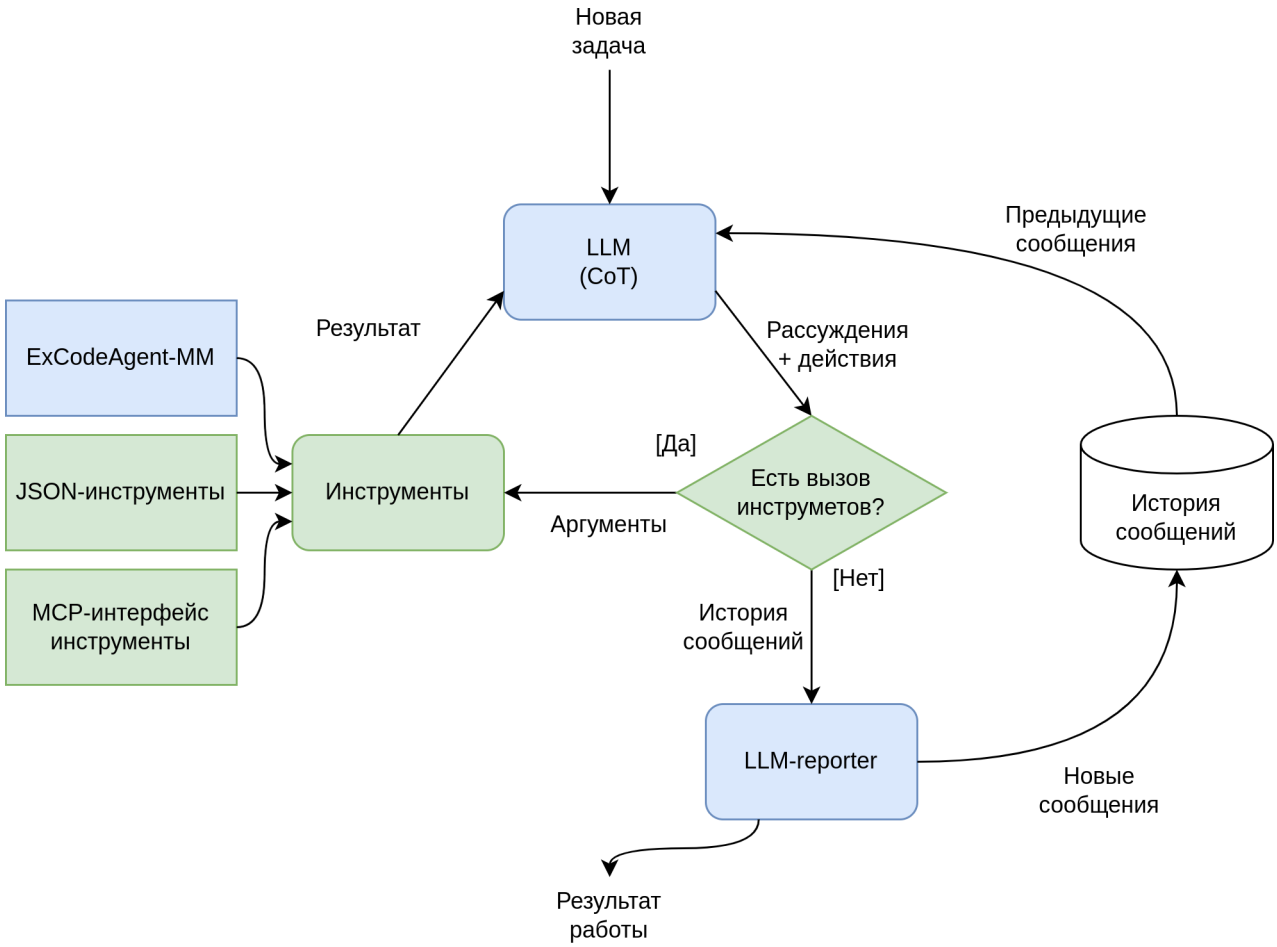

Рис.3.5. Структура полной мультиагентной системы ExCodeAct.

гибкость в используемых инструментах, которые вызываются через узел “Инструменты”, что увеличивает спектр решаемых задач. Кроме того, данная система естественно встраивается в формат диалога с пользователем, который может дополнительно передавать новые задачи, связанные с предыдущими. Заключительной причиной в выборе базовой системы для модификации послужил тренд в развитии качества рассуждений больших языковых моделей, что позволит в будущем решать более сложные задачи куда менее усложненными системами.

Используемые данные об инструментах, будь то кодовые инструменты или MCP-интерфейсы с JSON-инструментами, ровно как и системные инструкции к различным агентам сети, задаются в файле конфигурации в формате YAML - формат имеет низкий порог к требуемым знаниям для заполнения исследователями.

Все специальные хранилища, которые используются в процессе работы, являются простыми переменными: хранилища для сообщений - списки, хранилище для переменных - словарь (хэш-таблица), где ключами служат названия переменных, а хранилище для pydoc документаций - словарь, в котором ключами служат имена кодовых инструментов.

3.3. Детали реализации

Предлагаемое решение было реализовано при помощи фреймворков разработки мультиагентных систем на базе LLM LangChain и LangGraph Для тестирования различных вариаций систем и выбора конечного решения использовался среда LangSmith [29—31].

Графический интерфейс предлагаемого решения реализован при помощи фреймворка Streamlit, предлагающего возможность быстро и удобно разворачивать LLM-агентов и MAS, работающих на базе LangGraph [32].

ГЛАВА 4. АПРОБАЦИЯ РЕШЕНИЯ

4.1. Оценка ExCodeAgent-MM

“Ядром” предлагаемого решения ExCodeAct является агент кодогенерации ExCodeAgent-MM: именно он в апробации использует бо’льшую часть спектра предлагаемых инструментов.

Ввиду специфики задачи (а именно создание MAS для работы *с внутренними* инструментами), использование публичных бенчмарков для тестирования является в определенном смысле нецелесообразным: важно оценить качество агента именно в применении внутренних инструментов, информация о которых была малоизвестная LLM-моделям.

Потому ниже будет представлен полное изложение анализа предлагаемого решения и сравнения его с аналогами в кодогенерации с использованием внешних источников.

4.1.1. ClearML-bench - система корректности работы с окружением

В рамках тестирования корректности работы системы был создан собственный бенчмарк ClearML-bench, содержащий 15 вопросов-сценариев для взаимодействие с окружением в различных вариациях: работа с датасетами (создание, загрузка и выгрузка, анализ содержмого), работа с данными (преобразования,

визуализации и сохранение) и прочее. Полный список вопросов представлен в Приложении №[8].

Принцип, по которому происходит тестирование, описать просто: каждое задание направлено на определенный набор взаимодействий с окружением (будь то файлы на локальном хранилище, датасеты в системе ClearML). Сначала, перед запуском агента кодогенерации, производится оценка окружения - так формируется начальное, t_0 описание окружения. Описание окружения состоит из содержимого ожидаемой рабочей директорией агента, с которой предполагается работа (файлы, директории), а также из содержимого самого ClearML: какие датасеты и задания на текущий момент там находятся. После этого производится запуск агента кодогенерации с последующим снятием конечного описания окружения t_1. Поскольку все представленные выше элементы описания окружения можно представить в виде множеств, над которым определена операция разности, можно посчитать разность $\delta t = t_1 \backslash t_0$ и сравнить её с тем, какой результат ожидался. В случае совпадения ожидания задание засчитывается и ставится 1 балл, в противном случае - 0 баллов. Краткий график с работой представлен на [9].

Хоть бенчмарк и назван в честь инструмента менеджмента данных и моделей ClearML, но тестирование охватывает сценарии с применением *различных* инструментов: ClearML носит больше описательную характеристику окружения, с которым инструменты отчасти иногда взаимодействуют.

4.1.2. Используемые инструменты

В ходе тестирования принимало участие 4 внутренних кодовых инструмента–пакетов, выполняющие следующие роли:

- “cloudml_manager” - модификация Python SDK ClearML для автоматического структурирования артефактов деятельности лаборатории (датасетов, заданий, моделей) по проектам;
- “deconvolution_module” - модуль с классами и статическими методами для работы с трехмерными изображениями и их улучшения при помощи алгоритма деконволюции;
- “denoising_module” - модуль с классами и статическими методами для работы с трехмерными изображениями и их улучшения при помощи алгоритма денойзинга;

[8] добавить приложение

[9] рисуночек тут с ClearML-bench

– “biobert_module” - мокап-модуль с классами и статическими методами для работы с сигналами активности мозга; использовался как дополнительный функционал в рамках тестирования для оценки длины контекста и качества выбора нужного инструмента;
– “spinetool_module” - мокап-модуль с классами и статическими методами для работы с трехмерными моделями дендритных шипов; использовался как дополнительный функционал в рамках тестирования для оценки длины контекста и качества выбора нужного инструмента.

Полный список используемых инструментов лаборатории представлен в Приложении №[10].

4.1.3. Результаты кодогенерации

Тестирование ExCodeAgent-MM производилось в полном описании, которое приводилось в главе 3: с использованием Coarse-Fine RAG и кодовой LLM Codestral-latest. Данная модель больше обучена для генерации и работы с кодом, потому её использование здесь вполне логично.

Кроме того, для сравнения было проведено тестирование как на различных ретриверах, работающих по принципу косинусального сходства (с семантическим и рекурсивным разбиением документации на блоки),а также с различными LLM-моделями, список который ниже:

– Mistral-Large-instruct - большая модель семейства Mistral общего назначения;
– Mistral-Small-instruct - маленькая модель семейства Mistral на 8B параметров;
– Mistral-Mini-instruct - самая маленькая модель семейства Mistral на 3B параметров;

Все данные модели были запущены на удаленных вычислительных мощностях, предоставляемые самим Mistral по API.

В качестве измеряемых параметров, помимо числа баллов в бенчмарке ClearML-bench, были еще время работы и количество использованных токенов.

На рисунке рис.4.1 представлены результаты работы ExCodeAgent-MM на различных ретриверах. Кроме обычных ретриверов также приведены результаты работы в случае, когда использована подгрузка документации всех инструментов

[10]сделать таблицу с инструментами

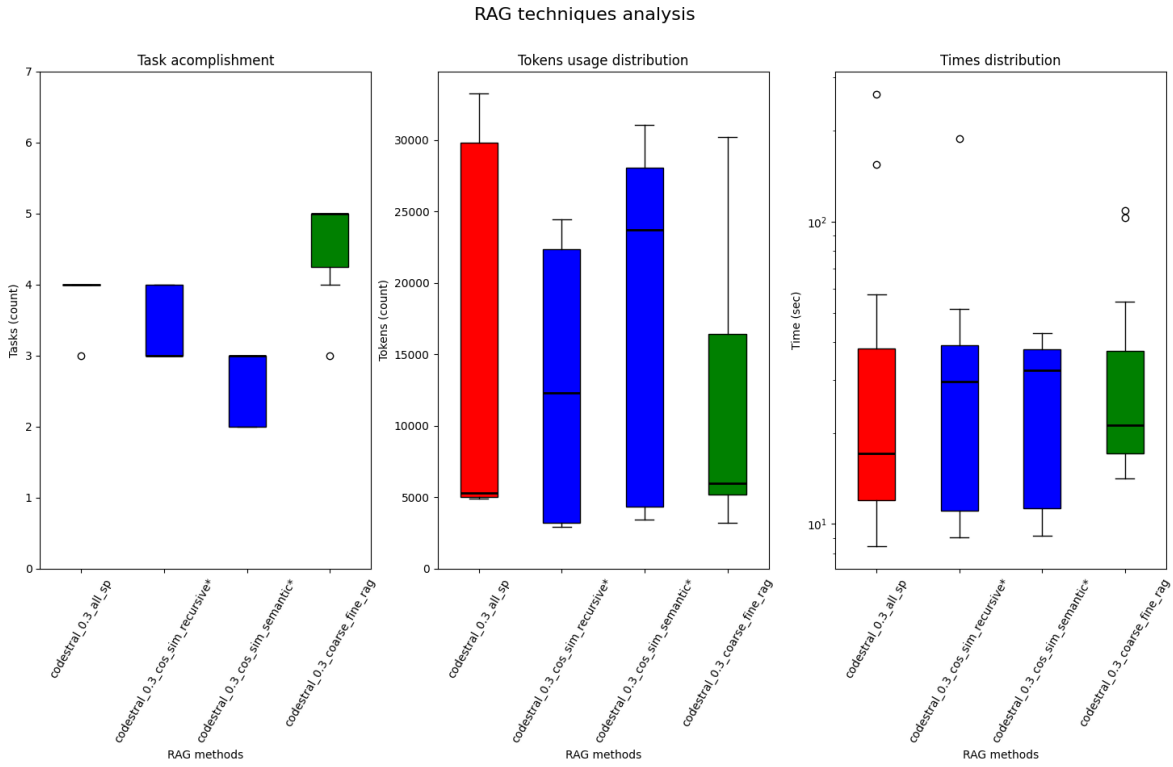

Рис.4.1. Результаты работы ExCodeAgent-MM на различных ретриверах.

в контекст модели. Как видно, количество токенов в случае “coarse-fine RAG” наименьшее, а число выполненных задач - наибольшее. Это свидетельствует о том, что вся полезная информация для кодогенераций подгружается лучше в условиях структурированной организации, которые классические ретриверы не обеспечивают.

На рисунке рис.4.2 представлены результаты работы ExCodeAgent-MM на различных больших языковых моделях. Из графико видно, что использование как специализированной, так и большой модели общего назначения, практически одинаково сказываются на числе решенных задач. Также стоит заметить, что число решенных задач по мере уменьшения числа обучаемых параметров модели также падают - малыми моделями решаются только простые задачи, в которых требуется вызов нескольких методов без построения логики программы на основе циклов и условий.

Писать про важность работы с контекстами, даже для публичных тулов но которые редко используют: привести кодогенерации где что то забывается и тд и тп. Можно вынести в приложения. В приложения также можно вынести список тулов (таблицы), примеры промптов (как типа код-снипет). Писать про экстраполяцию на другие методы (опционально - если успеются графики на другие LLM семейства).

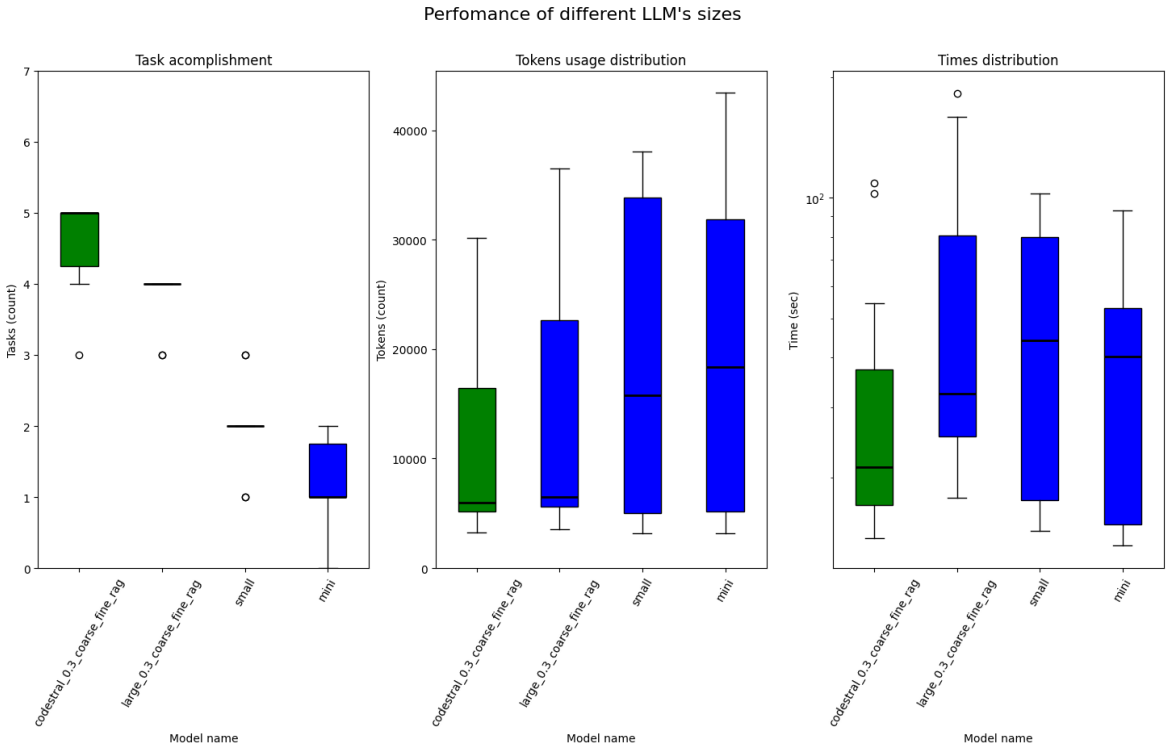

Рис.4.2. Результаты работы ExCodeAgent-MM на различных моделях.

4.1.4. Оценка количества наблюдений с текстовой и с мультимодальностью

Здесь можно привести примеры на тестовых задачах Kaggle в вопросах анализа табличных данных - сколько выводов извлекается из только тестовой модальности при автозапуске и сколько при мультимодальной (с анализом "чартов"). Очевидно, что будет не меньше - но насколько больше - лучше показать. (Опционально - если успеются цифры/графики).

4.2. Прочие качественные результаты

Здесь секция просто про то, что нельзя померить, но что просто есть и что стоит упомянуть.

Написать про оставшиеся свойства - про сохранение контекста, про создание юпитер ноутбуков про сохранение графиков в них и все такое.

ЗАКЛЮЧЕНИЕ

Заключение (2 – 5 страниц) обязательно содержит выводы по теме работы, *конкретные предложения и рекомендации* по исследуемым вопросам. Количество общих выводов должно вытекать из количества задач, сформулированных во введении выпускной квалификационной работы.

Предложения и рекомендации должны быть органически увязаны с выводами и направлены на улучшение функционирования исследуемого объекта. При разработке предложений и рекомендаций обращается внимание на их обоснованность, реальность и практическую приемлемость.

Заключение не должно содержать новой информации, положений, выводов и т. д., которые до этого не рассматривались в выпускной квалификационной работе. Рекомендуется писать заключение в виде тезисов.

Выражаю благодарность своему научному руководителю за визионерскую и профессиональную помощь в реализации предлагаемой работы. Выражаю также особую благодарность консультанту от Лаборатории Анализа Биомедицинских Изображений и Данных (ЛАБИД)

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

БЯМ Большая языковая модель.

LLM Large language model.

БФМ Большая фундаментальная модель.

MAS Multi-agent system.

RAG Retrieval-augmented generation.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Russell S.*, *Norvig P.* Artificial Intelligence: A Modern Approach. — 3rd. — USA: Prentice Hall Press, 2009.

2. *Sutton R. S.*, *Barto A. G.* Reinforcement Learning: An Introduction. — Second. — The MIT Press, 2018. — URL: http://incompleteideas.net/book/the-book-2nd.html.

3. Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models / L. Wang [и др.]. — 2023. — arXiv: 2305.04091 [cs.CL]. — URL: https://arxiv.org/abs/2305.04091.

4. Large Language Models are Zero-Shot Reasoners / T. Kojima [и др.]. — 2023. — arXiv: 2205.11916 [cs.CL]. — URL: https://arxiv.org/abs/2205.11916.

5. Tree of Thoughts: Deliberate Problem Solving with Large Language Models / S. Yao [и др.] // Advances in Neural Information Processing Systems. Т. 36 / под ред. A. Oh [и др.]. — Curran Associates, Inc., 2023. — С. 11809—11822. — URL: https://proceedings.neurips.cc/paper_files/paper/2023/file/271db9922b8d1f4dd7aaef84ed5ac703-Paper-Conference.pdf.

6. AIDE: AI-Driven Exploration in the Space of Code / Z. Jiang [и др.]. — 2025. — arXiv: 2502.13138 [cs.AI]. — URL: https://arxiv.org/abs/2502.13138.

7. Chat markup language chatml (preview). — URL: https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/chat-markup-language (visited on 30.04.2025).

8. Pixtral 12B / P. Agrawal [и др.] // ArXiv. — 2024. — Т. abs/2410.07073. — URL: https://api.semanticscholar.org/CorpusID:273229118.

9. Unraveling the Truth: Do VLMs really Understand Charts? A Deep Dive into Consistency and Robustness / S. Mukhopadhyay [и др.]. — 2024. — arXiv: 2407.11229 [cs.CL]. — URL: https://arxiv.org/abs/2407.11229.

10. ModelContextProtocol - official documentation. — URL: https://modelcontextprotocol.io/introduction (visited on 30.04.2025).

11. *Ray P. P.* A Survey on Model Context Protocol: Architecture, State-of-the-art, Challenges and Future Directions. — 2025. — DOI 10.36227/techrxiv.174495492.22752319/v1. — URL: http://dx.doi.org/10.36227/techrxiv.174495492.22752319/v1.

12. ModelContextProtocol - available public servers. — URL: https://github.com/modelcontextprotocol/servers (visited on 30.04.2025).

13. Executable Code Actions Elicit Better LLM Agents / X. Wang [и др.] // ArXiv. — 2024. — Т. abs/2402.01030. — URL: https://api.semanticscholar.org/CorpusID:267406155.

14. Attention is All you Need / A. Vaswani [и др.] // Advances in Neural Information Processing Systems. Т. 30 / под ред. I. Guyon [и др.]. — Curran Associates, Inc., 2017. — URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

15. Keep the Cost Down: A Review on Methods to Optimize LLM' s KV-Cache Consumption / L. Shi [и др.]. — 2024. — arXiv: 2407.18003 [cs.CL]. — URL: https://arxiv.org/abs/2407.18003.

16. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks / P. Lewis [и др.]. — 2021. — arXiv: 2005.11401 [cs.CL]. — URL: https://arxiv.org/abs/2005.11401.

17. Precise Zero-Shot Dense Retrieval without Relevance Labels / L. Gao [и др.]. — 2022. — arXiv: 2212.10496 [cs.IR]. — URL: https://arxiv.org/abs/2212.10496.

18. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning / DeepSeek-AI [и др.]. — 2025. — arXiv: 2501.12948 [cs.CL]. — URL: https://arxiv.org/abs/2501.12948.

19. RL-STaR: Theoretical Analysis of Reinforcement Learning Frameworks for Self-Taught Reasoner / F.-C. Chang [и др.]. — 2025. — arXiv: 2410.23912 [cs.AI]. — URL: https://arxiv.org/abs/2410.23912.

20. ReAct: Synergizing Reasoning and Acting in Language Models / S. Yao [и др.] // ArXiv. — 2022. — Т. abs/2210.03629. — URL: https://api.semanticscholar.org/CorpusID:252762395.

21. Plan and execute - implementation on LangGraph. — URL: https://langchain-ai.github.io/langgraph/tutorials/plan-and-execute/plan-and-execute/ (visited on 30.04.2025).

22. *He G.*, *Demartini G.*, *Gadiraju U.* Plan-Then-Execute: An Empirical Study of User Trust and Team Performance When Using LLM Agents As A Daily Assistant // Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems. — ACM, 2025. — С. 1—22. — (Сер.: CHI '25). — DOI 10.1145/3706598.3713218. — URL: http://dx.doi.org/10.1145/3706598.3713218.

23. On the Planning Abilities of Large Language Models (A Critical Investigation with a Proposed Benchmark) / K. Valmeekam [и др.]. — 2023. — arXiv: 2302.06706 [cs.AI]. — URL: https://arxiv.org/abs/2302.06706.

24. The AI Scientist: Towards Fully Automated Open-Ended Scientific Discovery / C. Lu [и др.]. — 2024. — arXiv: 2408.06292 [cs.AI]. — URL: https://arxiv.org/abs/2408.06292.

25. Agent Laboratory: Using LLM Agents as Research Assistants / S. Schmidgall [и др.]. — 2025. — arXiv: 2501.04227 [cs.HC]. — URL: https://arxiv.org/abs/2501.04227.

26. Data Interpreter: An LLM Agent For Data Science / S. Hong [и др.]. — 2024. — arXiv: 2402.18679 [cs.AI]. — URL: https://arxiv.org/abs/2402.18679.

27. Google Python Style Guide. — URL: https://google.github.io/styleguide/pyguide.html (visited on 01.05.2025).

28. Accurate structure prediction of biomolecular interactions with AlphaFold 3 / J. Abramson [и др.] // Nature. — 2024. — Т. 630. — С. 493—500. — DOI 10.1038/s41586-024-07487-w.

29. LangChain - documentation. — URL: https://python.langchain.com/docs/introduction/ (visited on 02.05.2025).

30. LangGraph - documentation. — URL: https://langchain-ai.github.io/langgraph/tutorials/introduction/ (visited on 02.05.2025).

31. LangSmith - documentation. — URL: https://docs.smith.langchain.com/ (visited on 02.05.2025).

32. Streamlit - documentation. — URL: https://docs.streamlit.io/ (visited on 02.05.2025).