Министерство науки и высшего образования Российской Федерации

|  |  |  |
| --- | --- | --- |
| УТВЕРЖДАЮ |  | УТВЕРЖДАЮ |
| Руководитель офиса операционной и проектной деятельности Ассоциации «Цифровые технологии в промышленности» |  | Научный руководитель ИЦ СИИП Университета ИТМО |
| \_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Погосова |  | \_\_\_\_\_\_\_\_\_\_\_\_\_ А.В. Бухановский |
| \_\_\_ \_\_\_\_\_\_\_\_ 2024 г. |  | \_\_\_ \_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

**ФРЕЙМВОРК БЫСТРОГО ПРОТОТИПИРОВАНИЯ ПРИЛОЖЕНИЙ НА ОСНОВЕ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (БЯМ)**

РУКОВОДСТВО ПРОГРАММИСТА

ЛИСТ УТВЕРЖДЕНИЯ

*Инв.№ подл.* GGJLKподп.

*Подп. и дата*

*Инв.№ дубл.*

*Подп. и дата*

*Взам.инв.№*

RU.СНАБ.00855-01 33 01-ЛУ

|  |
| --- |
| Представители  Организации-разработчика |
| Руководитель разработки |
| \_\_\_\_\_\_\_\_\_\_\_ А. В. Калюжная |
| \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |
| Нормоконтролер |
| \_\_\_\_\_\_\_\_\_\_\_ А. В. Киреева |
| \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

УТВЕРЖДЕН

RU.СНАБ.00855-01 33 01-ЛУ

**ФРЕЙМВОРК БЫСТРОГО ПРОТОТИПИРОВАНИЯ ПРИЛОЖЕНИЙ НА ОСНОВЕ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ (БЯМ)**

РУКОВОДСТВО ПРОГРАММИСТА

RU.СНАБ.00855-01 33 01

*Инв.№ подл.* GGJLKподп.

*Подп. и дата*

*Инв.№ дубл.*

*Подп. и дата*

*Взам.инв.№*

Листов 95

2024

**АННОТАЦИЯ**

Документ содержит руководство программиста для использования фреймворка быстрого прототипирования приложений на основе больших языковых моделей (БЯМ).

В число задач, решаемых фреймворком, входят упрощение и ускорение прототипирования систем с БЯМ на основе RAG, упрощение и ускорение прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов, повышение эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий создания сложных синтетических примеров для дообучения БЯМ. ProtoLLM разработан в рамках выполнения пункта 1.1.2 плана деятельности ИЦ СИИП по Соглашению с АНО «Аналитический центр при Правительстве Российской Федерации» № 70–2021–00141 (ИГК 000000D730324P540002).

СОДЕРЖАНИЕ

[1. ОБЩИЕ СВЕДЕНИЯ 6](#_Toc185708516)

[2. НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ 7](#_Toc185708517)

[2.1. Назначение программы 7](#_Toc185708518)

[2.2. Область применения 7](#_Toc185708519)

[2.3. Функциональные условия применения 8](#_Toc185708520)

[2.4. Технические условия применения 9](#_Toc185708521)

[3. ОПИСАНИЕ ПРИКЛАДНЫХ ЗАДАЧ 10](#_Toc185708522)

[3.1. Классы решаемых задач 10](#_Toc185708523)

[3.2. Примеры решения задач 10](#_Toc185708524)

[**Классификация данных энергопотребления** 10](#_Toc185708525)

[В данном примере будет рассмотрен набор данных ItalyPowerDemand [https://www.timeseriesclassification.com/description.php?Dataset=ItalyPowerDemand], взятый, аналогично предыдущему, из архива UCR/UEA. Датасет представляет собой данные о спросе на электроэнергию в Италии, где задача классификации заключается в отличии дней с октября по март и с апреля по сентябрь включительно. Объем обучающей и тестовой выборок составляют 60 и 61 образец соответственно, длина временного ряда равна 637 измерениям, количество классов – 2. На рисунке 3 приведены примеры классов, присутствующих в наборе данных. 10](#_Toc185708526)

[**3.2.1.** Создание новых лекарственных молекул для химической отрасли на одноагентной реализации (Жидковская) 13](#_Toc185708527)

[Химическая агентная LLM система является AI-помощником для специалистов в области химии по генерации новых лекарственных молекул исходя из содержания запроса пользователя. 13](#_Toc185708528)

[**3.2.2.** Ассистент для ответов на вопросы о городской застройке Санкт-Петербурга (Луценко) 14](#_Toc185708529)

[Урбанистическая агентная БЯМ система является умным ассистентом, который помогает управлять крупными урбанизированными территориями. Система дает комплексную информацию о городе, его состоянии, проблемах и планах развития на основе различных источников данных. 15](#_Toc185708530)

[Есть два основных источника данных: векторная база данных и сервисы урбанистической платформы, которые доступны через REST API. В векторной БД доступны документы о стратегическом развитии города. Через API платформы можно получить данные об обеспеченности сервисами и транспортом, планировании постройки новых домов, сервисов и данные о благоустройстве города. 15](#_Toc185708531)

[Предлагаемая система является мульти-агентной, её схема представлена на рисунке 3.2.2.1. Для генерации ответов на запросы пользователя система использует несколько агентов: для маршрутизации вопросов, валидации маршрутизации, подбора оптимальной функции API, извлечения контекста и суммаризации окончательных ответов. 15](#_Toc185708532)

[**3.2.3.** Создание новых лекарственных молекул для химической отрасли на мультиагентной реализации (Жидковская) 18](#_Toc185708533)

[Химическая мульти-агентная LLM система является AI-помощником для специалистов в области химии по генерации новых лекарственных молекул исходя из содержания запроса пользователя. 18](#_Toc185708534)

[**3.2.4.** Генерация синтетических валидационных данных для градостроительной отрасли (Ходорченко) 19](#_Toc185708535)

[4. ХАРАКТЕРИСТИКИ ПРОГРАММЫ 20](#_Toc185708536)

[4.1. Режимы работы программы (Луценко) 20](#_Toc185708537)

[4.2. Порядок оценки качества результатов программы (Луценко) 20](#_Toc185708538)

[4.2.1. Оценка точности маршрутизации запроса системой на основе БЯМ к подключенному плагину или сервису 20](#_Toc185708539)

[4.2.2. Оценка корректности ответа системы на основе БЯМ 21](#_Toc185708540)

[Для оценки корректности ответов на вопросы были использованы метрики G-Eval и Answer Relevancy (AR) из фреймворка DeepEval. 21](#_Toc185708541)

[Метрика G-Eval превосходит другие современные метрики и даёт оценку максимально приближённую к человеческой. Она использует БЯМ с техникой chain-of-thoughts (CoT) [1], чтобы оценивать ответы от приложений с БЯМ на основе заданных критериев. Для оценки результатов работы системы, были разработаны критерии, которые строго проверяют фактическую точность ответов. 21](#_Toc185708542)

[5. ОБРАЩЕНИЕ К ПРОГРАММЕ 25](#_Toc185708543)

[5.1. Вызов модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов 25](#_Toc185708544)

[5.2. Вызов модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG 26](#_Toc185708545)

[5.3. Вызов модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий 27](#_Toc185708546)

[5.4. Вызов модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ 34](#_Toc185708547)

[6. ПРОВЕРКА ПРОГРАММЫ 36](#_Toc185708548)

[7. ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ 41](#_Toc185708549)

[7.1. Состав и структура входных данных 41](#_Toc185708550)

[*7.1.1.* *Состав и структура входных данных для модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов* 41](#_Toc185708551)

[7.1.2. *Состав и структура входных данных для модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG* 41](#_Toc185708552)

[*7.1.3.* *Состав и структура входных данных для модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий* 42](#_Toc185708553)

[*7.1.4.* *Состав и структура входных данных для модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ* 43](#_Toc185708554)

[7.2. Подготовка входных данных 44](#_Toc185708555)

[*7.2.1.* *Подготовка входных данных для модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов* 44](#_Toc185708556)

[*7.2.2.* *Подготовка входных данных для модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG* 45](#_Toc185708557)

[7.2.3. *Подготовка входных данных для модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий* 47](#_Toc185708558)

[*7.2.4.* *Подготовка входных данных для модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ* 47](#_Toc185708559)

[7.3. Состав и структура выходных данных 48](#_Toc185708560)

[*7.3.1.* *Состав и структура выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов* 48](#_Toc185708561)

[*7.3.2.* *Состав и структура выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG* 48](#_Toc185708562)

[7.3.3. *Состав и структура выходных данных модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий* 49](#_Toc185708563)

[*7.3.4.* *Состав и структура выходных данных модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ* 49](#_Toc185708564)

[7.4. Интерпретация выходных данных 50](#_Toc185708565)

[*7.4.1.* *Интерпретация выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов* 50](#_Toc185708566)

[7.4.2. *Интерпретация выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG* 50](#_Toc185708567)

[7.4.3. *Интерпретация выходных данных модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий* 50](#_Toc185708568)

[7.4.4. *Интерпретация выходных данных модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ* 52](#_Toc185708569)

[8. СООБЩЕНИЯ 53](#_Toc185708570)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 54](#_Toc185708571)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 55](#_Toc185708572)

# **ОБЩИЕ СВЕДЕНИЯ**

Программный комплекс – фреймворк быстрого прототипирования приложений на основе больших языковых моделей (БЯМ) (далее – ProtoLLM) RU.СНАБ.00855-01 разработан в рамках выполнения пункта 1.1.2 плана деятельности ИЦ СИИП по Соглашению с АНО «Аналитический центр при Правительстве Российской Федерации» № 70–2021–00141 (ИГК 000000D730324P540002). Фреймворк предназначен для решения задач быстрого прототипирования приложений на основе БЯМ.

Разработанный фреймворк является кроссплатформенным программным обеспечением, поэтому системные программные средства, используемые фреймворком могут быть представлены локализованной версией любой операционной системы для ПК (Windows, Unix). Основным инструментом работы с фреймворком служит среда разработки (PyCharm).

Компонент разработан на языке Python 3.10 с использованием библиотек langchain, pydantic, chromadb, tornado, pypdf, redis, celery и размещен в репозиториях:

* https://gitlab.actcognitive.org/itmo-sai-code/ProtoLLM (версия с русскоязычным описанием)
* https://github.com/aimclub/ProtoLLM;

# **НАЗНАЧЕНИЕ И УСЛОВИЯ ПРИМЕНЕНИЯ**

* 1. **Назначение программы**

Разрабатываемый фреймворк предназначен поддержки технологических операций и типовых пайплайнов работы с БЯМ, включая дообучение моделей, работу с дополненной выборкой, ансамблирование и промптинг. Он позволяет ускорить разработку приложений и систем ИИ на основе БЯМ.

Функциональное назначение разрабатываемого фреймворка:

1. Быстрое прототипирование RAG-систем за счет предварительно настроенных компонентов и шаблонов для улучшения точности и релевантности ответов.
2. Быстрое прототипирование агентных систем за счет настроенных компонентов и шаблонов, а также настроенных БЯМ и  механизмов координации и взаимодействия между агентами.
3. Автоматический подбор и управление ансамблями БЯМ для оптимизации производительности и качества результатов.
4. Создание сложных, контекстно-зависимых примеров для дообучения и тестирования БЯМ.
5. Упрощение интеграции БЯМ и других инструментов в единую систему, за счет унифицированного интерфейс для подключения внешних API, баз данных и моделей.
6. Поддержка исследований и экспериментов:
   * Предоставление инструментов для проведения экспериментов с различными конфигурациями моделей и агентов.
   * Обеспечение воспроизводимости результатов исследований.
   1. **Область применения**

Областью применения разрабатываемого фреймворка является автоматизация создания отраслевых интеллектуальных систем на основе больших языковых моделей (БЯМ) и мультиагентных технологий для решения широкого спектра задач. Фреймворк может применяться как для создания полноценных приложений, так и в качестве компонента более крупных систем искусственного интеллекта.

Основные направления применения фреймворка включают:

1. Разработка отраслевых систем диалогового взаимодействия и виртуальных ассистентов, способных понимать сложные контексты и решать многоэтапные задачи.
2. Реализация отраслевых интеллектуальных систем поддержки принятия решений, основанных на глубоком анализе текстовой информации и мультиагентном подходе.
3. Разработка систем для автоматизации сложных бизнес-процессов, требующих понимания и генерации естественного языка.
4. Создание платформ для проведения экспериментов и исследований в области искусственного интеллекта и обработки естественного языка.
5. Фреймворк фокусируется на решении двух ключевых проблем:
6. Эффективное комбинирование и оркестрация различных языковых моделей и агентов для решения комплексных задач. Это требует разработки гибкой архитектуры, позволяющей интегрировать различные типы моделей и агентов, а также механизмов их взаимодействия и координации.
7. Автоматизация процесса создания и валидации синтетических массивов данных, которые могут использоваться

Для решения первой проблемы фреймворк предоставляет унифицированный интерфейс для работы с различными БЯМ и агентными системами, а также инструменты для создания сложных композитных структур, объединяющих различные компоненты.

Для решения второй проблемы реализуются алгоритмы оптимизации, способные работать с абстрактными представлениями мультиагентных систем и адаптироваться к специфике конкретных задач. Это включает методы автоматического подбора архитектуры системы, выбора оптимальных моделей и настройки их параметров.

Фреймворк также может применяться в составе более крупных платформ искусственного интеллекта для решения экспериментальных задач и тестирования новых подходов в области обработки естественного языка и мультиагентных систем. Это позволяет исследователям и разработчикам быстро прототипировать и оценивать эффективность инновационных решений, которые впоследствии могут быть интегрированы в промышленные системы.

* 1. **Функциональные условия применения**

Фреймворк не предназначен для решения задач взаимодействия с мульти-модальными БЯМ (поддерживается только текстовая модальность). Кроме того, не предусмотрена работа с большим числом агентов (более 20), не решается задача оптимизации графа взаимодействий между агентами. Не предоставляются готовые инструменты для интеграции БЯМ с данными из сети Интернет.

* 1. **Технические условия применения**

Для применения фреймворка необходимо наличие на компьютере интерпретатора python версии 3.10 и доступ к вычислительной системе под управлением OC Windows/Linux со следующими характеристиками:

1. Процессор (CPU):
   1. Минимальная конфигурация: 8-ядерный процессор (например, Intel Core i7 или AMD Ryzen 7).
   2. Рекомендуемая конфигурация: 16+ ядер, серверные процессоры типа Intel Xeon или AMD EPYC для работы с большими объемами данных и множеством параллельных процессов.
2. Оперативная память (RAM):
   1. Минимальная конфигурация: 32 ГБ.
   2. Рекомендуемая конфигурация: 64-128 ГБ или более, в зависимости от размера моделей и объема обрабатываемых данных.
3. Графический процессор (GPU):
   1. Минимальная конфигурация: NVIDIA GPU с 8 ГБ VRAM (например, GTX 1070 или выше).
   2. Рекомендуемая конфигурация: NVIDIA Tesla V100, A100 или аналогичные профессиональные GPU с 16-32 ГБ VRAM или более.
4. Хранилище:
   1. Минимальная конфигурация: HDD 500 ГБ.
   2. Рекомендуемая конфигурация: SSD 1-2 ТБ для быстрого доступа к данным и моделям.
5. Сеть: высокоскоростное подключение к интернету (1 Гбит/с или выше) для загрузки моделей и работы с облачными сервисами.

# **В ОПИСАНИЕ ПРИКЛАДНЫХ ЗАДАЧ**

* 1. **Классы решаемых задач**

Предметной областью применения разработанного фреймворка являются все отрасли промышленности в части процессов, требующих внедрения систем поддержки принятия решений и интеллектуальных ассистентов. Практические примеры применения с соответствующими постановками задачи в терминах машинного обучения приведены в таблице 3.1.

Таблица 3.1 – Классы и примеры задач, решаемых фреймворком

|  |  |
| --- | --- |
| Класс решаемой задачи | Практическая реализация задачи |
| Прототипирование систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов | Создание новых лекарственных молекул для химической отрасли на одноагентной реализации |
| Прототипирование систем с БЯМ на основе RAG | Ассистент для ответов на вопросы о стратегии развития Санкт-Петербурга |
| Повышение эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий | Создание новых лекарственных молекул для химической отрасли на мультиагентной реализации |
| Генерация синтетических отраслевых даннных для тестирования ИИ-ассистентов и СППР | Генерация синтетических валидационных данных для градостроительной отрасли |

* 1. **Примеры решения задач**

ПРИМЕР

**Классификация данных энергопотребления**

В данном примере будет рассмотрен набор данных ItalyPowerDemand [https://www.timeseriesclassification.com/description.php?Dataset=ItalyPowerDemand], взятый, аналогично предыдущему, из архива UCR/UEA. Датасет представляет собой данные о спросе на электроэнергию в Италии, где задача классификации заключается в отличии дней с октября по март и с апреля по сентябрь включительно. Объем обучающей и тестовой выборок составляют 60 и 61 образец соответственно, длина временного ряда равна 637 измерениям, количество классов – 2. На рисунке 3 приведены примеры классов, присутствующих в наборе данных.

Идет вставка изображения...

Рисунок 4 – Примеры классов из набора данных ItalyPowerDemand

Для запуска эксперимента необходимо воспользоваться скриптом из листинга Листинг 5, приведённого ниже.

Листинг 5 – Пример запуска эксперимента

|  |
| --- |
| from fedot\_ind.api.main import FedotIndustrial  from fedot\_ind.tools.loader import DataLoader  data\_loader = DataLoader(dataset='ItalyPowerDemand',  folder=’local\_folder\_path’)  X\_test, y\_test, X\_train, y\_train = data\_loader.load\_data()  model = FedotIndustrial(task='ts\_classification', timeout=5,  strategy='topological', n\_jobs=-1, window\_size=15)  model.fit(features=X\_train, target=y\_train)  prediction = model.predict(features=X\_test)  metrics = model.get\_metrics(target=y\_test) |

Для данного примера был выбран топологический метод извлечения признаков, принцип работы которого показан на рисунке Рисунок 5. Данный метод генерации признаков может использоваться как самостоятельно, так и в составе другого метода. Идея заключается в формировании вектора признаков для каждого временного ряда, извлекаемых из персистентной диаграммы. Затем набор извлечённых признаков и вектор классов используются для обучения классификационной модели средствами алгоритмического ядра Fedot.

Изображение выглядит как текст, снимок экрана, графический дизайн

Автоматически созданное описание

Рисунок 5 – Топологический алгоритм извлечения признаков

В результате эксперимента была получена следующая структура композитной модели классификации на топологических признаках (рисунок Рисунок 6):

Изображение выглядит как диаграмма, линия, круг, дизайн

Автоматически созданное описание

Рисунок 6 – Композитная модель классификации на топологических признаках

Результаты сравнения метрики, полученной с использованием разработанного алгоритма, с метриками лучших SOTA решений приведены в таблицe Таблица 5.

Таблица 5 – Результаты эксперимента с данными ItalyPowerDemand

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Алгоритм | Fedot\_Ind | HIVE-COTE v1.0 | TS-CHIEF | InceptionTime |
| Roc-Auc | 0.994 | 0.991 | 0.992 | 0.988 |

Согласно приведённым в таблице 5 результатам, решение задачи классификации временных рядов энергопотребления наилучшим образом обеспечивается с использованием разработанного фреймворка Fedot.Industrial.

* + 1. Создание новых лекарственных молекул для химической отрасли на одноагентной реализации (Жидковская)

Химическая агентная LLM система является AI-помощником для специалистов в области химии по генерации новых лекарственных молекул исходя из содержания запроса пользователя.

Система состоит из одного агента. У агента есть доступ к инструментам для достижения максимально возможной достоверности ответа. Инструменты задействованные в системе:

*gen\_mols(num: int),*

*gen\_mols\_alzheimer(num: int),*

*gen\_mols\_multiple\_sclerosis(num: int),*

*gen\_mols\_dyslipidemia(num: int),*

*gen\_mols\_acquired\_drug\_resistance(num: int),*

*gen\_mols\_lung\_cancer(num: int),*

*gen\_mols\_parkinson(num: int).*

Все из перечисленных инструментов отправляют запрос по API к генеративным моделям с необходимыми аргументами для генерации молекулы со свойствами, которые ожидает пользователь. Каждый инструмент отвечает за генерацию молекулы для конкретной задачи, а именно для следующих болезней: Альцгеймер, Паркинсон, Дислипидемия, Лекарственная устойчивость, Рак легких, Склероз.

В качестве коннектора был использован класс Llama31ChatModel из ProtoLLM, который является универсальной связкой между моделью и модулями библиотеки LangChain. Коннектор позволяет использовать обертки от LangChain, что уменьшает необходимость написания дополнительного кода.

Для оценки эффективности системы был использован датасет состоящий из 300 запросов, в которых были просьбы о генерации молекулы с определенными заранее свойствами. В одном запросе могло содержаться от 1 до 3 различных просьб. Пример запросов для генерации лекарства от болезни Альцгеймер:

*“Generate GSK-3beta inhibitors with high docking score”,*

*“Generate molecules which can bind to GSK-3beta protein”.*

Примеры по генерации лекарства от болезни Паркинсон:

*“Generate me 2 molecules that could help in the treatment of Parkinson's disease, focusing on compounds that support the regulation of dopamine levels and protect neurons from oxidative stress and mitochondrial dysfunction. It is important that these molecules do not cause severe side effects such as hallucinations, dyskinesia, or cardiovascular issues.”,*

*“Create molecules using a fragment-based approach, combining elements of natural alkaloids.”*

Эффективность системы можно считать высокой. При использовании языковой модели Llama 3.1 70b кол-во верных ответов (общая точность системы) достигло 71,42%. При этом процент верно определенных инструментов агентом составил 82%, согласно Таблице 3.2.1.1. Такие результаты означают, что система допускает ошибки при генерации итогового ответа пользователю.

Таблица 3.2.1.1 — Метрики оценки эффективности агентной системы

|  |  |
| --- | --- |
| Вид оценки | Accuracy |
| Определение инструментов | 80% |
| Итоговый ответ системы | 72% |

* + 1. Ассистент для ответов на вопросы о городской застройке Санкт-Петербурга (Луценко)

Урбанистическая агентная БЯМ система является умным ассистентом, который помогает управлять крупными урбанизированными территориями. Система дает комплексную информацию о городе, его состоянии, проблемах и планах развития на основе различных источников данных.

В данной реализации были использованы классы LLMAPI и OuterLLMAPI из библиотеки ProtoLLM для обеспечения возможности работы с БЯМ из различных источников, а также модуль ProtoLLM.rags для работы с векторной базой данных ChromaDB. Модуль позволяет создавать и удалять коллекции, загружать данные в коллекции, а также извлекать информацию, наиболее подходящую для ответа на вопрос пользователя.

* + 1. Создание новых лекарственных молекул для химической отрасли на мультиагентной реализации (Жидковская)

Химическая мульти-агентная LLM система является AI-помощником для специалистов в области химии по генерации новых лекарственных молекул исходя из содержания запроса пользователя.

Система состоит из двух агентов: Decomposer, Conductor. Агент Decomposer принимает на вход запрос пользователя и определяет задачи исходя из содержания. Агент может определить от 1 до 5 задач в рамках одного запроса.

Пример запроса пользователя и результат работы агента Decomposer. Запрос пользователя:

*“Generate GSK-3beta inhibitors with high docking score and low brain-blood barrier permeability. Generate inhibitors of KRAS protein with G12C mutation. The inhibitors should be selective, meaning they should not bind with HRAS and NRAS proteins.”.*

Ответ агента Decomposer:

*“[“Generate GSK-3beta inhibitors with high docking score and low brain-blood barrier permeability.”, “Generate inhibitors of KRAS protein with G12C mutation.The inhibitors should be selective, meaning they should not bind with HRAS and NRAS proteins.”]”.*

Далее агент Conductor обрабатывает каждую из переданных задач от агента Decomposer в отдельности, определяет нужную функцию (функции совпадают с указанными в разделе 3.2.1), вызывает, используя возможности класса AgentExecutor из LangChain, рефлексирует над результатом, при необходимости запускает процесс определения нужного инструмента заново, возвращает ответ на каждую из задач пользователю.

Итоговый ответ возвращается пользователю по шагам, для каждой задачи в отдельности.

Для оценки эффективности была произведена валидация на датасете из 300 примеров, каждый из которых состоял из 1-3 задач.

По результатам проверки было выяснено, что реализация из двух агентов справляется лучше реализации из одного агента. Метрики качества можно наблюдать в таблице 3.2.3.1. Значение точности определения инструментов совпадает с итоговой точностью в связи с тем, что ответы выдаются пользователю в процессе их составления без составления итогового сообщения, содержащего информацию об ответах на каждую из задач.

Таблица 3.2.3.1 — Метрики оценки эффективности мульти-агентной системы

|  |  |
| --- | --- |
| Вид оценки | Accuracy |
| Определение инструментов | 91,5% |
| Итоговый ответ системы | 91,5% |

В данной реализации был использован класс Llama31ChatModel из библиотеки ProtoLLM для обеспечения корректной работы БЯМ с фреймворком LangChain.

* + 1. Генерация синтетических валидационных данных для градостроительной отрасли (Ходорченко)

# **ХАРАКТЕРИСТИКИ ПРОГРАММЫ**

* 1. **Режимы работы программы**

Фреймворк ProtoLLM может использоваться в нескольких режимах:

1. Работа БЯМ без RAG и без использования внешних сервисов в качестве инструментов (автономный режим работы модели). Пример использования приведен в листинге 4.1.1.1.
2. Работа БЯМ только с RAG. Пример использования приведен в листинге 4.1.1.2.
3. Работа БЯМ с вызовом внешних сервисами. Пример использования приведен в листинге 4.1.1.3.
4. Работа БЯМ и с RAG, и с внешними сервисами. Пример использования приведен в листинге 4.1.1.4.

Кроме того, каждый режим может быть модифицирован с помощью выбора модели (в том числе переключения между локальной и «облачной» моделью) – её название передается в параметре model. Примеры кода для каждого из режимов приведены в Листингах 4.1.1.1-4.1.14.

Листинг 4.1.1.1 – Использование ProtoLLM для вызова БЯМ в автономном режиме

|  |
| --- |
| llmapi = OuterLLMAPI(model=”llama-3.1-70binstruct”, key)  llm\_request = PromptModel(job\_id=str(uuid.uuid4()),  meta=meta,  content=question)  res = llmapi.inference(llm\_request)  return res.content |

Листинг 4.1.1.2 – Использование ProtoLLM для вызова БЯМ с RAG

|  |
| --- |
| # Функция для запросов к БД  def proto\_view(  query: str,  collection: str,  k: int = 1,  embedding\_function: HuggingFaceHubEmbeddings = None,  ) -> list:  # Возвращает k ближайших к запросу фрагментов  embedding\_host = os.environ.get("EMBEDDING\_HOST")  embedding\_function = HuggingFaceHubEmbeddings(model=embedding\_host)  chroma\_client = init\_chroma\_client()  docs\_searcher\_models = DocsSearcherModels(embedding\_model=embedding\_function, chroma\_client=chroma\_client)  retriever = DocRetriever(top\_k=k,  docs\_searcher\_models=docs\_searcher\_models,  )  return retriever.retrieve\_top(collection\_name=collection, query=query)  context = proto\_view(question, collection\_name)  context = f'Вопрос: {question} Контекст: {context[0].page\_content}'  # Получение ответа от БЯМ  print(f'Ответ VseGPT LLM: \n {outer\_llm(context, meta, key, model)}') |

Листинг 4.1.1.3 – Использование ProtoLLM для вызова БЯМ с внешними сервисами

|  |
| --- |
| @tool  def call\_data\_service (json\_data: dict, params: dict) -> str:  self.\_check\_params(params)  url, params = self.\_parse\_url\_params(params)  result = self.\_execute\_request(url, params=params, json\_data=json\_data)  if result.status\_code != 200:  logger.error(f"url: {url}")  logger.error(f"params: {params}")  logger.error(f"json\_data: {json\_data}")  raise RequestException(result.status\_code)  return result.json()  context = call\_data\_service(input\_data, params)  context = f'Вопрос: {question} Ответ сервиса: {context[0].page\_content}'  # Получение ответа от БЯМ  print(f'Ответ VseGPT LLM: \n {outer\_llm(context, meta, key, model)}') |

Листинг 4.1.1.4 – Использование ProtoLLM для вызова БЯМ с RAG и внешними сервисами

|  |
| --- |
| context = call\_data\_service(input\_data)  context = f'Вопрос: {question} Ответ сервиса: {context[0].page\_content}'  context = proto\_view(question, collection\_name)  context = context += f'Ответ сериса: {context[0].page\_content}'  # Получение ответа от БЯМ  print(f'Ответ VseGPT LLM: \n {outer\_llm(context, meta, key, model)}') |

В качестве примера эффекта от использования различных режимов рассмотрим урбанистическую мульти-агентную БЯМ-систему, созданную с помощью ProtoLLM, которая должна предоставлять комплексную информацию о городе, его проблемах, состоянии на основе различных источников данных. Для этого в ней использованы различные режимы работы ProtoLLM, которые обеспечивают извлечение нужного контекста из векторной базы данных и API урбанистической платформы. Также в системе предусмотрена возможность использовать для расчетов различные виды БЯМ.

Чтобы корректно ответить на вопрос пользователя, необходимо правильно маршрутизировать его запрос и использовать нужный режим. Прежде всего необходимо выбрать сервис (БД или API). Затем из БД необходимо подобрать нужные документы, а из API нужные функции для вызова вместе с параметрами.

* 1. **Порядок оценки качества результатов программы** (Луценко)
     1. Оценка качества маршрутизации запроса системой на основе БЯМ к подключенному плагину или сервису

Для оценки качества маршрутизации запроса системой на основе БЯМ к подключенному плагину или сервису используется метрика Accuracy (точность, в %).

Значения метрики точности выбора режима работы ProtoLLM с помощью БЯМ представлены в таблице 4.2.1.

Метрика в таблице 4.2.1 оценены на примере системы, включающей в себя несколько агентов, которые маршрутизируют запросы с помощью механизма Function Calling. Один агент отвечает за выбор правильного источника данных: БД или API. Второй агент выбирает необходимые функции для извлечения контекста из API. Метрика точности могут быть оценены на различных БЯМ, чтобы проверить качество работы агентов.

Таблица 4.2.1.1 — Оценка точности маршрутизации запроса

|  |  |  |
| --- | --- | --- |
| Модель | Выбор источника  данных, % | Выбор функции  для API, % |
| gpt-4o-2024-08-06 | 97.3 | 80.0 |
| mixtral-8x22binstruct | 94.0 | 94.7 |
| llama-3.1-70binstruct | 99.3 | 96.0 |
| llama-3.1-70binstruct-int4 | 98.7 | 93.3 |

В первой колонке результаты подбора источника данных. Все модели показывают высокий результат от 94% до 99%. Лучший результат показала неквантизованная Llama 3.1-70b-instruct. Во второй колонке результаты подбора функции API. В этом случае БЯМ нужно было выбирать между большим числом функций, это привело к снижению качества. У трёх моделей результаты снизились на несколько процентов до 93% - 96%. Худшие результаты у модели gpt-4o-2024-08-06 - 80%. Лучший результат опять показала неквантизованная Llama 3.1-70b-instruct.

Таким образом, мы можем заключить, что агенты с оптимально подобранными моделями и достаточно подробными описаниями функций обеспечивают успешную маршрутизацию запросов пользователя.

* + 1. Оценка корректности ответа системы на основе БЯМ

В рамках эксперимента оценивалась точность выбора режима работы ProtoLLM с помощью БЯМ. Для оценки корректности ответов на вопросы были использованы метрики G-Eval и Answer Relevancy (AR) из фреймворка DeepEval.

Метрика G-Eval даёт оценку максимально приближённую к человеческой. Она использует БЯМ с техникой chain-of-thoughts (CoT) [1], чтобы оценивать ответы от приложений с БЯМ на основе заданных критериев. Для оценки результатов работы системы, были разработаны критерии, которые строго проверяют фактическую точность ответов.

Метрика AR также основана на использовании БЯМ. Она оценивает, насколько ответ от системы с БЯМ соответствует правильному ответу от эксперта. Метрика AR намного менее жёсткая чем G-Eval [2].

Были вычислены метрики для различных БЯМ на разных режимах работы системы: БЯМ без контекста, БЯМ с контекстом только из векторной БД, БЯМ с контекстом только из API и БЯМ с контекстом из БД и API одновременно. Средние значения обеих метрик на всех вопросах представлены в таблице 4.2.2.1.

Таблица 4.2.2.1 — Оценка корректности ответа

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Модель | БЯМ | | БЯМ + ДБ | | БЯМ + API | | БЯМ + ДБ + API | |
| G-Eval | AR | G-Eval | AR | G-Eval | AR | G-Eval | AR |
| gpt-4o-2024-08-06 | 0.30 | 0.93 | 0.38 | 0.65 | 0.52 | 0.66 | 0.71 | 0.94 |
| mixtral-8x22binstruct | 0.38 | 0.90 | 0.43 | 0.70 | 0.51 | 0.73 | 0.68 | 0.91 |
| llama-3.1-70binstruct | 0.38 | 0.95 | 0.41 | 0.74 | 0.54 | 0.71 | 0.74 | 0.95 |
| llama-3.1-70binstruct-int4 | 0.38 | 0.98 | 0.39 | 0.71 | 0.50 | 0.69 | 0.71 | 0.92 |

В первой колонке находятся данные для БЯМ без контекста. Результаты G-Eval для всех моделей примерно одинаковые 0.30-0.38. Показатели достаточно низкие, это свидетельствует о том, что БЯМ не могут полноценно отвечать на вопросы без предоставления дополнительной информации. При этом показатели AR высокие 0.90-0.98. БЯМ хорошо справляются с задачей генерации текста на заданную тему, но часто в процессе выдумывают информацию. Это объясняет высокие показатели AR и более низкие показатели G-Eval.

Во второй колонке находятся результаты БЯМ с контекстом из БД. С добавлением контекста значения метрик G-Eval повысились на 13%. Прирост не очень высокий, потому что вопросы на тему стратегии развития города достаточно общие, с некоторыми БЯМ хорошо справляется без контекста. Показатели AR при этом понизились до 0.65-0.74. Система была разработана таким образом, чтобы все ответы давались исключительно на основе предоставленного контекста. Если нужной информации нет, система возвращает сообщение, что данные для ответа не предоставлены. Это снижает показатели метрики AR.

В третьей колонке находятся результаты БЯМ с контекстом из API. В этом случае после добавления контекста значения метрик G-Eval повысились на 45%. В вопросах по этой теме часто нужны точные числовые показатели. БЯМ с релевантным контекстом отлично справляется с такой задачей. БЯМ без контекста почти никогда не может дать правильный ответ. Показатели AR в этом случае очень близки к AR для БЯМ с контекстом из БД.

В четвертой колонке представлены результаты для БЯМ с контекстом из обоих источников. Значения метрик G-Eval значительно повысились для всех моделей: до 0.68 - 0.74. Метрика G-Eval очень строгая и требует полного соответствия ответа БЯМ правильному ответу. Часто в правильном ответе от экспертов присутствует дополнительная информация, которая не требуется в ответе от БЯМ, чтобы он считался верным. G-Eval очень чувствительна к эти отличиям, это приводит к снижению метрики. Таким образом, можно считать, что значение G-Eval выше 0.8 показатель верного ответа. Значения AR в этом режиме также высокие 0.91 - 0.95. Это говорит о том, что система эффективно подбирает источник контекста.

Таким образом, мы можем заключить, что при использовании БЯМ выбор подходящего источника данных и последующее извлечение правильного контекста играют решающую роль в обработке сложных запросов.

Описание G-eval и AR из статьи и примеры таблицы с их оценкой из статьи

* + 1. Оценка времени выполнения запросов системой на основе БЯМ

Было измерено время работы системы в разных режимах. Результаты представлены в таблице 4.2.3.1.

Таблица 4.2.3.1 — Оценка времени выполнения запросов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Модель | БЯМ, с | БЯМ + ДБ, с | БЯМ + API, с | БЯМ + ДБ + API, с |
| gpt-4o-2024-08-06 | 2.6 | 6.1 | 17.4 | 12.1 |
| mixtral-8x22binstruct | 3.8 | 4.2 | 17.1 | 10.5 |
| llama-3.1-70binstruct | 7.4 | 10.0 | 21.8 | 14.6 |
| llama-3.1-70binstruct-int4 | 2.5 | 3.3 | 15.2 | 9.2 |

БЯМ без контекста работает достаточно быстро, среднее время на обработку одного запроса 3-7 секунд. БЯМ с контекстом из векторной БД в среднем работает на 2 секунды дольше, а БЯМ с контекстом из API на 14 секунд дольше.

Система в режиме БЯМ с контекстом из БД и API в среднем работает на 8 секунд больше чем БЯМ без контекста. Дополнительное время необходимо для подбора нужного источника данных (дополнительные запросы к БЯМ) и извлечения контекста из БД и API.

1. **ОБРАЩЕНИЕ К ПРОГРАММЕ**
   1. **Вызов модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов**

В листинге Листинг 5.1.1 приведен пример использования модуля для упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов.

Листинг 5.1.1 – Фрагмент кода для создания агента с помощью модуля прототипирования систем с БЯМ.

|  |
| --- |
| from langchain.agents import (  create\_structured\_chat\_agent,  AgentExecutor,  tool,  )  from langchain.prompts import ChatPromptTemplate, MessagesPlaceholder  from langchain.tools.render import render\_text\_description\_and\_args  from langchain\_core.prompts import SystemMessagePromptTemplate, HumanMessagePromptTemplate  from protollm.agents.llama31\_agents.llama31\_agent import Llama31ChatModel  system\_message = SystemMessagePromptTemplate.from\_template(  “some  “some system prompt”,  input\_variables=["tools", "tool\_names"],  )  human\_message = HumanMessagePromptTemplate.from\_template(  “some  “some human prompt”,  input\_variables=["input", "agent\_scratchpad"],  )  prompt = ChatPromptTemplate.from\_messages([  [  system\_message,  MessagesPlaceholder(variable\_name="chat\_history", optional=True),  human\_message]),])  llm = Llama31ChatModel(api\_key="API\_KEY", base\_url="BASE\_URL",  model="MODEL")  agent = create\_structured\_chat\_agent(llm=llm, tools=tools,  prompt=prompt, stop\_sequence=True)  agent\_executor = AgentExecutor.from\_agent\_and\_tools(  agent=agent, tools=tools,  verbose=True, return\_intermediate\_steps=True, output\_keys=["output"],  )  user\_question = "What is the sum and product of 15 and 27?"  response = agent\_executor.invoke({"input": user\_question})  final\_answer = response["output"]  print(f"Agent's Response: \n {final\_answer}") |

Класс Llama31ChatModel является langchain-совместимым [3]. Для его работы необходимо применнить его любом классическом сценарии langchain для создания системы на основе БЯМ, которая может использовать сервисы, приложения и плагины в виде инструментов.

* 1. **Вызов модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG**

В листинге 5.2.1 ниже приводится пример вызова модуля для сценария упрощения и ускорения прототипирования систем с БЯМ на основе RAG.

Листинг 5.2.1 – Фрагмент скрипта вызова модуля для генерации ответа на пользовательский вопрос на естественном языке с использованием RAG.

|  |
| --- |
| import uuid  from protollm\_sdk.jobs.utility import construct\_job\_context  from protollm\_sdk.utils.reddis import get\_reddis\_wrapper, load\_result  from protollm\_sdk.rags.jobs import RAGJob  # Шаг 1. Инициализация уникального номера идентификации  job\_id = str(uuid.uuid4())  # Шаг 2. Инициализация переменных доступа к БД и SDK  job\_name = "fast\_validation"  ctx = construct\_job\_context(job\_name)  # Шаг 3. Запуск поиска релевантных документов  RAGJob().run(job\_id, ctx, user\_prompt='Какой бывает арматура железобетонных конструкций?', do\_reranking=False)  # Шаг 4. Получение ответа модели из базы данных.  rd = get\_reddis\_wrapper()  result = load\_result(rd, job\_id, job\_name) |

Перед запуском процесса с использованием модуля требуется задать несколько переменных: id и название запуска, а также объект класса JobContext, который конструируется с помощью функции construct\_job\_context. Класс JobContext открывает возможности использования всех функций и сервисов, задекларированных внутри SDK проекта.

Объект класса RAGJob позволяет запустить процесс с помощью встроенного метода run(). Этот метод принимает на вход id запуска, экземпляр класса JobContext, вопрос, ответ на который требуется найти, а также параметр use\_advanced\_rag, который отвечает за настройку аугментации запроса.

* 1. **Вызов модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий**

В листинге 5.3.1 приводится пример запуска в отладочном режиме API компоненты модуля повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий.

Листинг 5.3.1 – Фрагмент скрипта старта сервера API компоненты модуля повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий.

|  |
| --- |
| import logging  from protollm\_agents.entrypoint import Entrypoint  from protollm\_agents.sdk.agents import StreamingAgent  from protollm\_agents.sdk.models import CompletionModel, ChatModel, TokenizerModel, EmbeddingAPIModel  from protollm\_agents.sdk.vector\_stores import ChromaVectorStore  from pipelines.rag\_agent import RAGAgent  logging.basicConfig(  level=logging.DEBUG,  format="%(asctime)s - %(name)s - %(levelname)s - %(filename)s:%(lineno)d - %(message)s",  handlers=[logging.StreamHandler()],  )  logger = logging.getLogger(\_\_name\_\_)  HOST = "localhost"  if \_\_name\_\_ == "\_\_main\_\_":  epoint = Entrypoint(  models = [  CompletionModel(  name="planner\_llm",  model="/model",  temperature=0.01,  top\_p=0.95,  streaming=False,  url=f"http://{HOST}:8001/v1",  api\_key="token",  ),  CompletionModel(  name="generator\_llm",  model="/model",  temperature=0.01,  top\_p=0.95,  streaming=True,  url=f"http://{HOST}:8001/v1",  api\_key="token",  ),  ChatModel(  name="router\_llm",  model="/model",  temperature=0.01,  top\_p=0.95,  streaming=True,  url=f"http://{HOST}:8001/v1",  api\_key="token",  ),  TokenizerModel(  name="qwen\_2.5",  path\_or\_repo\_id="Qwen/Qwen2.5-7B-Instruct",  ),  EmbeddingAPIModel(  name="e5-mistral-7b-instruct",  model="/models/e5-mistral-7b-instruct",  url=f"http://{HOST}:58891/v1",  api\_key="token",  check\_embedding\_ctx\_length=False,  tiktoken\_enabled=False,  ),  ],  agents = [  RAGAgent(  name="rag\_domain1",  description="Поиск по базе докуметов домена 1",  arguments=RAGAgent.Arguments(  max\_input\_tokens=6144,  max\_chat\_history\_token\_length=24576,  retrieving\_top\_k=2,  generator\_context\_top\_k=2,  include\_original\_question\_in\_queries=True,  planner\_model\_name="planner\_llm",  generator\_model\_name="generator\_llm",  tokenizer\_name="qwen\_2.5",  store\_name="chroma\_domain1",  ),  ),  RAGAgent(  name="rag\_domain2",  description="Поиск по базе докуметов домена 2",  arguments=RAGAgent.Arguments(  max\_input\_tokens=6144,  max\_chat\_history\_token\_length=24576,  retrieving\_top\_k=2,  generator\_context\_top\_k=2,  include\_original\_question\_in\_queries=True,  planner\_model\_name="planner\_llm",  generator\_model\_name="generator\_llm",  tokenizer\_name="qwen\_2.5",  store\_name="chroma\_domain2",  ),  ),  RAGAgent(  name="rag\_domain3",  description="Поиск по базе докуметов домена 3",  arguments=RAGAgent.Arguments(  max\_input\_tokens=6144,  max\_chat\_history\_token\_length=24576,  retrieving\_top\_k=2,  generator\_context\_top\_k=2,  include\_original\_question\_in\_queries=True,  planner\_model\_name="planner\_llm",  generator\_model\_name="generator\_llm",  tokenizer\_name="qwen\_2.5",  store\_name="chroma\_domain3",  ),  ),  ],  vector\_stores = [  ChromaVectorStore(  name="chroma\_domain1",  description="Chroma vector store",  embeddings\_model\_name='e5-mistral-7b-instruct',  host=HOST,  port=57777,  collection\_name='domain1',  ),  ChromaVectorStore(  name="chroma\_domain2",  description="Chroma vector store",  embeddings\_mel\_name='e5-mistral-7b-instruct',  host=HOST,  port=57777,  collection\_name='domain2',  ),  ChromaVectorStore(  name="chroma\_domain3",  description="Chroma vector store",  embeddings\_mel\_name='e5-mistral-7b-instruct',  host=HOST,  port=57777,  collection\_name='domain3',  ),  ]  )  epoint.run() |

Перед запуском API компоненты фреймворка необходимо задать основные объекты системы. Пул моделей задается при помощи объектов CompletionModel, ChatModel, TokenizerModel, EmbeddingAPIModel, которые позволяют добавить модели langchain-совместимых типов VLLMOpenAI, ChatOpenAI, AutoTokenizer и OpenAIEmbeddings, соответственно. Объекты класса RAGAgent, наследуемого от StreamingAgent класса обладают методами stream и ivoke, которые позволяют вызывать отдельных агентов по вебсокету через agent\_id, или использовать их в роутинговом и ансамблевом агентах как инструменты, получаемые методом to\_tool. Реализация агента RAGAgent полагается на использование пайплайнов Langchain. Отдельно добавляется пул векторных хранилищ типа ChromaVectorStore. Для указания модели эмбеддингов используются наименования (атрибут `name`) пула моделей типа EmbeddingAPIModel, для указания у агентов используемых векторных хранилищ используются наименования (атрибут `name`) пула векторных хранилищ.

В листинге 5.3.2 приводится пример запуска в экплуатационном режиме API компоненты модуля повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий.

Листинг 5.3.2 – Фрагмент скрипта старта сервера API компоненты модуля повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий с помощью конфигурационного файла.

|  |
| --- |
| import logging  from protollm\_agents.entrypoint import Entrypoint  logging.basicConfig(  level=logging.DEBUG,  format="%(asctime)s - %(name)s - %(levelname)s - %(filename)s:%(lineno)d - %(message)s",  handlers=[logging.StreamHandler()],  )  logger = logging.getLogger(\_\_name\_\_)  if \_\_name\_\_ == "\_\_main\_\_":  epoint = Entrypoint(config\_path="./examples/admin-config.yml")  epoint.run() |

Листинг 5.3.3 – Конфигурационный файл API компоненты модуля повышения эффективности БЯМ за счет использования ансамблей моделей.

|  |
| --- |
| app\_port: 8000  app\_host: 0.0.0.0  redis\_host: localhost  redis\_port: 55531  postgres\_host: localhost  postgres\_port: 55530  postgres\_user: user  postgres\_password: password  postgres\_db: agents  agents:  - name: rag\_domain1  description: Поиск по базе докуметов домена 1  class\_path: pipelines.rag\_agent.RAGAgent  default\_params:  max\_input\_tokens: 6144  max\_chat\_history\_token\_length: 24576  retrieving\_top\_k: 2  generator\_context\_top\_k: 2  include\_original\_question\_in\_queries: True  planner\_model\_name: planner\_llm  generator\_model\_name: generator\_llm  tokenizer\_name: qwen\_2.5  store\_name: chroma\_domain1  - name: rag\_domain2  description: Поиск по документам домена 2  class\_path: pipelines.rag\_agent.RAGAgent  default\_params:  max\_input\_tokens: 6144  max\_chat\_history\_token\_length: 24576  retrieving\_top\_k: 2  generator\_context\_top\_k: 2  include\_original\_question\_in\_queries: True  planner\_model\_name: planner\_llm  generator\_model\_name: generator\_llm  tokenizer\_name: qwen\_2.5  store\_name: chroma\_domain2  - name: rag\_domain3  description: Поиск по документам домена 3  class\_path: pipelines.rag\_agent.RAGAgent  default\_params:  max\_input\_tokens: 6144  max\_chat\_history\_token\_length: 24576  retrieving\_top\_k: 2  generator\_context\_top\_k: 2  include\_original\_question\_in\_queries: True  planner\_model\_name: planner\_llm  generator\_model\_name: generator\_llm  tokenizer\_name: qwen\_2.5  store\_name: chroma\_domain3  models:  - type: completion  params:  model: /model  temperature: 0.01  top\_p: 0.95  streaming: false  name: planner\_llm  url: <http://localhost:8001/v1>  api\_key: token  - type: completion  params:  model: /model  temperature: 0.01  top\_p: 0.95  streaming: true  name: generator\_llm  url: <http://localhost:8001/v1>  api\_key: token  - type: chat  params:  model: /model  temperature: 0.01  top\_p: 0.95  streaming: true  name: router\_llm  url: <http://localhost:8001/v1>  api\_key: token  - type: embedding  params:  name: e5-mistral-7b-instruct  url: <http://localhost:58891/v1>  api\_key: token  model: /models/e5-mistral-7b-instruct  check\_embedding\_ctx\_length: false  tiktoken\_enabled: false  - type: tokenizer  params:  name: qwen\_2.5  path\_or\_repo\_id: Qwen/Qwen2.5-7B-Instruct  vector\_stores:  - type: chroma  params:  name: chroma\_domain1  description: vector\_store\_description  host: localhost  port: 57777  collection\_name: domain1  embeddings\_model\_name: e5-mistral-7b-instruct  - type: chroma  params:  name: chroma\_domain2  description: vector\_store\_description  host: localhost  port: 57777  collection\_name: domain2  embeddings\_model\_name: e5-mistral-7b-instruct  - type: chroma  params:  name: chroma\_domain3  description: vector\_store\_description  host: localhost  port: 57777  collection\_name: domain3  embeddings\_model\_name: e5-mistral-7b-instruct |

Аналогичная представленной в Листинге 5.3.1 конфигурация пула ресурсов представлена в конфигурационном файле Листинга 5.3.3. Дополнительно передаются параметры баз данных Redis и Postgresql для хранения агентов и результатов их вызовов. Объект Entrypoint инициализируется из YAML файла, как показано в Листинге 5.3.2.

После инициализации Entrypoint объекта, для запуска API компоненты необходимо вызвать метод run(), который запустит uvicorn сервис с приложением FastAPI на http://<host>:<port>. По адресу http://<host>:<port>/docs доступны REST эндпойнты OpenAPI, включая методы листинга и просмотра агентов по идентификатору, а также эндпойнты для старта задач агентов типа BackgroundAgent, доступного для наследования из SDK. Также по URL сервиса доступен набор Websocket эндпойнтов: /agent, /router, /ensemble, которые позволяют вызывать отдельного стримингового агента, роутер-агента и ансамблирующего агента, соответственно.

* 1. **Вызов модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ**

Пример запуска модуля для генерации синтетических примеров с помощью больших языковых моделей приведен в Листинге 5.4.1.

Листинг 5.4.1 – Фрагмент скрипта вызова синтетической генерации примеров

|  |
| --- |
| from protollm.synthetics.synthetic\_pipelines.chains import SummarisationChain  from protollm.synthetics.utils import VLLMChatOpenAI, Dataset  qwen2vl\_api\_key = os.environ.get("QWEN2VL\_OPENAI\_API\_KEY")  qwen2vl\_api\_base = os.environ.get("QWEN2VL\_OPENAI\_API\_BASE")  llm=VLLMChatOpenAI(  api\_key=qwen2vl\_api\_key,  base\_url=qwen2vl\_api\_base,  model="/model",  max\_tokens=2048,  )  dataset = Dataset(path="data/sample\_summarization\_dataset.csv", labels=True)  summarisation\_chain = SummarisationChain(llm=llm)  summarisation\_chain.run(dataset, n\_examples=100)  summarisation\_chain.save\_chain\_output("data/sample\_summarization\_chain\_output.csv") |

В листинге показан пример явного указания модели с openai-совместимым интерфейсом и вызова имеющегося шаблона выполнения суммаризации.

1. **ПРОВЕРКА ПРОГРАММЫ**

Запуск модульных и интеграционных тестов может быть выполнен различными способами:

1. Локально развернуть конфигурации фреймворка, установить зависимости из файла requirements.txt для тестируемого модуля (pip install requirements.txt). Перейти в каталог test, запустить команду pytest –s.
2. На платформе github выполнить обновление основной ветки кода – это приведет к автоматическому запуску модульных текстов. Тесты располагаются в репозитории по адресу: <https://github.com/aimclub/ProtoLLM/tree/main/tests>, <https://github.com/aimclub/ProtoLLM/tree/main/protollm_tools/sdk/tests> и <https://github.com/aimclub/ProtoLLM/tree/main/protollm_tools/sdk/tests>. Для каждого теста используется собственный набор зависимостей.
   1. **Работа с агентными БЯМ**

Тестирование класса для работы с БЯМ в режиме агента приведено в Листинге 6.1.1.

Входные данные: пользовательский запрос

Проверка: корректность выводимых значений.

Ожидаемый результат: final\_answer – объект типа String с не-пустым значением.

Листинг 6.1.1– Код тестирования интерфейса для агентных БЯМ

|  |
| --- |
| from langchain.agents import (  create\_structured\_chat\_agent,  AgentExecutor,  tool, ) from langchain.prompts import ChatPromptTemplate, MessagesPlaceholder from langchain\_core.prompts import SystemMessagePromptTemplate, HumanMessagePromptTemplate  from protollm.tests.mock\_chat\_model import MockChatModel   def test\_from\_prompt\_model():   @tool  def add\_numbers(a: int, b: int) -> int:  """Adds two numbers."""  return a + b   @tool  def multiply\_numbers(a: int, b: int) -> int:  """Multiplies two numbers."""  return a \* b   # List of tools  tools = [add\_numbers, multiply\_numbers]   # Create the system and human prompts  system\_prompt = '''Respond to the human as helpfully and accurately as possible. You have access to the following tools:    {tools}    Use a JSON blob to specify a tool by providing an "action" key (tool name) and an "action\_input" key (tool input).    Valid "action" values: "Final Answer" or {tool\_names}    Provide only ONE action per JSON blob, as shown:    {{ "action": $TOOL\_NAME, "action\_input": $INPUT }}    Follow this format:    Question: input question to answer  Thought: consider previous and subsequent steps  Action: $JSON\_BLOB    Observation: action result  ... (repeat Thought/Action/Observation N times)  Thought: I know what to respond  Action: {{ "action": "Final Answer", "action\_input": "Final response to human" }}      Begin! Reminder to ALWAYS respond with a valid JSON blob of a single action. Use tools if necessary.   Respond directly if appropriate. Format is Action:```$JSON\_BLOB``` then Observation'''   human\_prompt = '''{input}  {agent\_scratchpad}  (Reminder to respond in a JSON blob no matter what)'''   system\_message = SystemMessagePromptTemplate.from\_template(  system\_prompt,  input\_variables=["tools", "tool\_names"],  )  human\_message = HumanMessagePromptTemplate.from\_template(  human\_prompt,  input\_variables=["input", "agent\_scratchpad"],  )   prompt = ChatPromptTemplate.from\_messages(  [  system\_message,  MessagesPlaceholder(variable\_name="chat\_history", optional=True),  human\_message,  ]  )   llm = MockChatModel(  api\_key="API\_KEY",  base\_url="BASE\_URL",  model="MODEL",  temperature=0.5,  max\_tokens=3000,  )   # Create the structured chat agent  agent = create\_structured\_chat\_agent(  llm=llm,  tools=tools,  prompt=prompt,  stop\_sequence=True,  )   # Create the AgentExecutor  agent\_executor = AgentExecutor.from\_agent\_and\_tools(  agent=agent,  tools=tools,  verbose=True,  return\_intermediate\_steps=True, # Set to True if you want intermediate steps  output\_keys=["output"],  )   user\_question = "What is the sum and product of 15 and 27?"   response = agent\_executor.invoke({"input": user\_question})   final\_answer = response["output"]   assert final\_answer is not None |

* 1. **Проверка работы с коннектором к БЯМ**

Тестирование коннектора для работы с БЯМ в режиме чата приведено в Листинге 6.1.2.

Входные данные: название модели

Проверка: корректность создания чат-модели

Ожидаемый результат: chat– объект типа ChatRESTServer с не-пустым значением.

Листинг 6.1.2 – Код тестирования коннектора к БЯМ

|  |
| --- |
| from langchain\_core.messages import HumanMessage from protollm.connectors.rest\_server import ChatRESTServer  def test\_connector():  conn = ChatRESTServer()  conn.base\_url = 'mock'  chat = conn.create\_chat(messages=[HumanMessage('M1'), HumanMessage('M2'), HumanMessage('M3')])  assert chat is not None |

* 1. **Создание модели-агента с заданным промптом**

Тестирование класса для создание модели-агента приведено в Листинге 6.1.1.

Входные данные: пользовательский запрос

Проверка: корректность значений полей модели

Ожидаемый результат:поля – объекты типа String с ожидаемым значением.

Листинг 6.1.3 – Код тестирования интерфейса для создание модели-агента с заданным промптом

|  |
| --- |
| from protollm\_sdk.models.job\_context\_models import PromptModel, PromptMeta, ChatCompletionModel  def test\_from\_prompt\_model(): prompt\_model = PromptModel( job\_id="test\_job\_123", meta=PromptMeta( temperature=0.5, tokens\_limit=100, stop\_words=["stop", "words"], model="gpt-3" ), content="This is a test prompt" )  chat\_completion = ChatCompletionModel.from\_prompt\_model(prompt\_model)  assert chat\_completion.job\_id == prompt\_model.job\_id assert chat\_completion.meta == prompt\_model.meta  assert len(chat\_completion.messages) == 1  assert chat\_completion.messages[0].role == "user" assert chat\_completion.messages[0].content == prompt\_model.content  assert chat\_completion.meta.temperature == 0.5 assert chat\_completion.meta.tokens\_limit == 100 assert chat\_completion.meta.stop\_words == ["stop", "words"] assert chat\_completion.meta.model == "gpt-3" |

1. **ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ**
   1. **Состав и структура входных данных**
      1. ***Состав и структура входных данных для модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов***

При использовании систем на основе БЯМ с подключением внешних сервисов необходимо подать инструкции, подключаемые сервисы и запрос.

В таблице 7.1.1 указаны входные данные, необходимые для корректной работы модуля.

Таблица 7.1.1 – Входные данные модуля для упрощения и ускорения прототипирования систем БЯМ с возможностью подключения внешних сервисов

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование данных | Структура данных | Способ ввода данных | Ограничения |
| Системные инструкции | Объект класса SystemMessagePromptTemplatee | Ручной ввод по шаблону | - |
| Пользовательские инструкции | Объект класса HumanMessagePromptTemplate | Ручной ввод по шаблону | - |
| Запрос | Строка | Ручной ввод | - |
| Список подключаемых сервисов | Список объектов класса StructuredTool | Ручной ввод | - |

* + 1. ***Состав и структура входных данных для модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG***

При решении задач генерации ответа БЯМ с учетом релевантных знаний, извлеченных из базы знаний, входными данными является запрос, ответ на который требуется сгенерировать.

В таблице 7.1.2 указаны типы, структура, а также способы ввода и ограничения входных данных.

Таблица 7.1.2 – Входные данные модуля для упрощения и ускорения прототипирования систем с БЯМ на основе RAG.

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование данных | Структура данных | Способ ввода данных | Ограничения |
| Уникальный номер идентификации запроса | Строка | Программное задание | Уникальность |
| Контекст | Объект класса JobContext | Программное задание | - |
| Запрос | Строка | Ручной ввод / программное задание | - |
| Настройка стратегии аугментации | Логическое значение | Ручной ввод / программное задание | - |
| Путь к файлу с настройками процесса обработки документов | Строка | Ручной ввод / программное задание | - |
| Путь к файлу с переменными окружения | Строка | Ручной ввод / программное задание | - |

* + 1. ***Состав и структура входных данных для модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий***

Для проведения запросов к отдельным агентам посредством Websocket эндпойнта необходимо передать запрос в формате JSON, содержащий:

* идентификатор сессии,
* идентификатор агента, доступный на эндпойнте листинга агентов или заданный на этапе конфигурирования компоненты,
* историю сессии (если существует),
* запрос, ответ на который требуется сгенерировать,
* параметры агента (при необходимости проведения запроса с параметрами, отличающимися от параметров, заданных на этапе конфигурирования компоненты).

Проведение запросов к агентам типа RouterAgent и EnsembleAgent посредством Websocket эндпойнта необходимо передать запрос в формате JSON, содержащий:

* идентификатор сессии,
* историю сессии (если существует),
* запрос, ответ на который требуется сгенерировать

В таблице Таблица 7.1.3 указаны типы, структура, а также способы ввода и ограничения входных данных.

Таблица 1.3 – Входные данные для модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование данных | Структура данных | Способ ввода данных | Ограничения |
| Идентификатор сессии | UUID4 | Значение поля “dialogue\_id” на эндпойнтах /agent, /router, /ensemble | - |
| Идентификатор агента | UUID4 | Значение поля “agent\_id” на эндпойнте /agent | - |
| История сессии | Список словарей | Значение поля “chat\_history” на эндпойнтах /agent, /router, /ensemble | Не более 50 объектов |
| Запрос | Строка | Значение поля “query” на эндпойнтах /agent, /router, /ensemble |  |
| Параметры агента в рамках запроса | Словарь | Значение поля “run\_params” на эндпойнте /agent | Соответствие схеме модели параметров агента |

* + 1. ***Состав и структура входных данных*** *для* ***модуля*** *на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ*

Таблица 7.1.4 – Входные данные для вызова обработки данных для задачи генерации синтетических примеров

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование данных | Структура данных | Способ ввода данных | Ограничения |
| Путь до датасета | Строка | Ручной ввод | - |
| Тип решаемой задачи | Строка | Ручной ввод | - |
| Инструкция | Строка | Ручной ввод | - |
| Лейблы | Строка | Ручной ввод | - |
| Количество генерируемых примерос | Целочисленное значение | Ручной ввод | - |

* 1. **Подготовка входных данных**
     1. ***Подготовка входных данных для модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов***

Подготовка данных для модуля осуществляется с помощью различных классов. Подготовка системных и пользовательских инструкций осуществляется с помощью классов SystemMessagePromptTemplate и HumanMessagePromptTemplate, которые преобразуют системные инструкции из строки в специальный класс (Листинг 7.2.1).

Листинг 7.2.1 – Пример использования классов SystemMessagePromptTemplate и HumanMessagePromptTemplate

|  |
| --- |
| from langchain\_core.prompts import SystemMessagePromptTemplate, HumanMessagePromptTemplate  system\_message = SystemMessagePromptTemplate.from\_template(system\_prompt,  input\_variables=["tools", "tool\_names"])  human\_message = HumanMessagePromptTemplate.from\_template(human\_prompt,  input\_variables=["input", "agent\_scratchpad"])  prompt = ChatPromptTemplate.from\_messages(  [  system\_message,  MessagesPlaceholder(variable\_name="chat\_history", optional=True),  human\_message,  ]  ) |

Кроме того, алгоритм требует подачу сервисов, приложений и плагинов в качестве функций с декоратором @tool или объектов класса StructuredTool (Листинг 7.2.2).

Листинг 7.2.2 – Пример использования классов SystemMessagePromptTemplate и HumanMessagePromptTemplate

|  |
| --- |
| from langchain.agents import (  create\_structured\_chat\_agent,  AgentExecutor,  tool,  )  @tool  def add\_numbers(a: int, b: int) -> int:  """Adds two numbers."""  return a + b  @tool  def multiply\_numbers(a: int, b: int) -> int:  """Multiplies two numbers."""  return a \* b  tools = [add\_numbers, multiply\_numbers]  llm = Llama31ChatModel(api\_key="API\_KEY", base\_url="BASE\_URL",  model="MODEL", temperature=0.5, max\_tokens=3000)  agent = create\_structured\_chat\_agent(llm=llm, tools=tools,  prompt=prompt, stop\_sequence=True) |

* + 1. ***Подготовка входных данных для модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG***

В рамках подготовки данных для модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG [4] необходимо создать экземпляр класса JobContext, который предоставляет возможности использования всех функций и сервисов, задекларированных внутри библиотеки ProtoLLM, а также сгенерировать уникальную строку номера идентификации с помощью класса uuid. Пример использования класса JobContext и библиотеки uuid представлен в листинге 7.2.3.

Листинг 7.2.3 – Пример использования класса JobContext и библиотеки uuid для подготовки данных

|  |
| --- |
| import uuid  from protollm\_sdk.jobs.utility import construct\_job\_context  # Шаг 1. Инициализация уникального номера идентификации  job\_id = str(uuid.uuid4())  # Шаг 2. Инициализация требуемых переменных для доступа к SDK проекта  job\_name = "fast\_validation"  ctx = construct\_job\_context(job\_name) |

Инструкция с параметрами для процесса обработки и загрузки документов описывается в файле с названием docs\_processing\_config.yaml (см. листинг 7.2.4). Его необходимо поместить в директорию /config\_files. В этом файле требуется прописать все необходимые настройки, касающиеся чтения и сбора информации из файлов – это указывается под ключевым словом “loader”, а также процесса разбивания файла на части заданной длины. Настройки этого процесса прописываются под ключевым словом “splitter”. Дополнительно указывается модель, которая выполняет процесс лексического анализа – разбора входной последовательности символов на распознавание групп – лексем – с целью получения на выходе идентифицированных последовательностей, под ключевым словом “tokenizer”.

Листинг 7.2.4 – Скрипт файла для настройки процесса сбора и обработки информации из документов

|  |
| --- |
| loader:  loader\_name: 'PDFLoader'  parsing\_params:  parsing\_scheme: 'paragraphs'  extract\_images: False  extract\_tables: False  parse\_formulas: False  remove\_service\_info: True  handle\_converting\_error: False splitter:  splitter\_name: 'hierarchical\_merger'  splitter\_params:  chunk\_size: 510  chunk\_overlap: 0  separators:  - '\n\n'  - '\n'  - '. '  - ', '  - '.'  - ','  - ' '  - ''  keep\_separator: False  add\_start\_index: False  strip\_whitespace: True  apply\_chunks\_merge: True tokenizer: 'intfloat/multilingual-e5-large' |

Также необходимо описать файл с переменными окружения с названием chroma.env (см. листинг 7.2.5). В нем требуется прописать ip адрес и порт базы данных ChromaDB, критерий, по которому будут сравниваться документы, название коллекций, где будут лежать загруженные далее документы, а также название и URL адрес модели токенизации.

Листинг 7.2.5 – Скрипт файла с переменными окружениями

|  |
| --- |
| CHROMA\_HOST='10.32.15.30' CHROMA\_PORT=9941 ALLOW\_RESET=False COLLECTION\_NAME='rag\_main\_collection' COLLECTION\_NAMES\_FOR\_ADVANCE=["rag\_main\_collection\_file\_name", "rag\_main\_collection\_keywords", "rag\_main\_collection\_content"] EMBEDDING\_NAME='intfloat/multilingual-e5-large' EMBEDDING\_HOST='http://10.32.15.30:9942/embed' DISTANCE\_FN='cosine' |

Передача пользовательского запроса в модуль осуществляется в виде строки на естественном языке, дополнительная подготовка не требуется.

* + 1. ***Подготовка входных данных для модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий***

Подготовка входных данных для использования API сервиса компонента повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий не требуется.

* + 1. ***Подготовка входных данных для модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ***

Подготовка документов аналогична подготовке данных, описанных в 7.2.2, если пользователь не воспользовался готовым датасетом или не произвел обработку своими средствами. При использовании функций protollm необходимо отметить, что в листинге 7.2.4 необходимо внести небольшие изменения – убрать секции чанкования и токенизации [5] и сохранить полученные документы в формате json. Также необходимо добавить поле «source\_uuid» определяющее наименование файла или его уникальный идентификатор. Входной файл может быть формата csv или json с обязательными колонками: «source\_uuid», «content» и необязательной колонкой с примерами ожидаемого результата для «content».

* 1. **Состав и структура выходных данных**
     1. ***Состав и структура выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов***

После успешного выполнения цикла, модуль возвращает словарь с изначальным вопросом, ответом и промежуточными шагами.

Таблица 7.3.1 - Выходные данные модуля прототипирования систем с БЯМ с возможностью подключения внешних сервисов, приложений, моделей в виде плагинов

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование данных | Структура данных | Способ вывода данных | Ограничения |
| Результат работы модуля | Словарь | Запись в переменную | - |

* + 1. ***Состав и структура выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG***

В таблице 7.3.2 указано описание выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования с БЯМ на основе RAG.

После окончания работы модуля возвращается результат генерации ответа на запрос с учетом релевантной информации, который отправляется на запись в базу данных.

Таблица 7.3.2 – Выходные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование данных | Структура данных | Способ вывода данных | Ограничения |
| Результат генерации алгоритма | Строка | Запись в базу данных | Длина контекста в зависимости от выбранной БЯМ |

* + 1. ***Состав и структура выходных данных модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий***

В Таблице 7.3.3 указано описание выходных данных компоненты модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий.

По мере исполнения запроса агентом (агентами в случае использования агента-роутера или ансамблевого агента) с Websocket приходят промежуточные результаты, а также конечный результат генерации.

Таблица 7.3.3 – Выходные данные API сервиса компоненты модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование данных | Структура данных | Способ вывода данных | Ограничения |
| Промежуточный результат, финальный результат | Словарь | Сообщение Websocket | - |

* + 1. ***Состав и структура выходных данных модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ***

В таблице 7.3.4 указано описание выходных данных модуля на основе алгоритма создания сложных синтетических примеров.

После окончания работы модуля возвращается набор сгенерированных примеров в соответствии с запущенным типом генерации. Результат можно как сохранить в переменную для дальнейшей работы, так и записать в файл.

Таблица 7.3.4 – Выходные данные модуля генерации синтетических примеров

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование данных | Структура данных | Способ вывода данных | Ограничения |
| Набор синтетических примеров | Словарь | Запись в переменную / запись в json файл | - |

* 1. **Интерпретация выходных данных**
     1. ***Интерпретация выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов***

Структура выходного словаря представлена в листинге 7.4.1. Финальный ответ и промежуточные шаги представляют собой ответ на естественном языке.

Листинг 7.4.1 – Использование модуля сбора данных результатов эксперимента

|  |
| --- |
| {  'input': <user\_query\_string>,  'output': <final\_response\_string>,  'intermediate\_steps': [  (<AgentAction object>, <optional result or observation>, <step\_number>)  ]  } |

* + 1. ***Интерпретация выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG***

Результат работы модуля записывается в базу данных. Этот результат может быть выгружен из базы данных с помощью скрипта из листинга 7.4.2. Для этой операции требуется знать номер идентификации запроса, а также название запроса. После выгрузки выходные данные представляют собой ответ на пользовательский запрос на естественном языке.

Листинг 7.4.2 – Получение результата работы алгоритма RAG

|  |
| --- |
| from protollm\_sdk.utils.reddis import get\_reddis\_wrapper, load\_result  rd = get\_reddis\_wrapper()  result = load\_result(rd, job\_id, job\_name) |

* + 1. ***Интерпретация выходных данных модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий***

Структура выходного словаря представлена в листинге 7.4.3. Финальный ответ и промежуточные шаги представляют собой ответ на естественном языке или описание промежуточных результатов, например, словари документов с метаданными и содержанием, которые используются в качестве источников для извлечения ответов агента RAG.

Листинг 7.4.3 – Промежуточный результат (is\_eos=False) модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий

|  |
| --- |
| {  "event\_id": <уникальный идентификатор события>,  "agent\_id": <уникальный идентификатор агента в системе>,  "description": <описание полученного события>,  "is\_eos": false,  "name": <наименование события>,  "result": [  {  "id": <идентификатор части документа>,  "metadata": {  "chapter": <наименование главы документа>,  "file\_name": <наименование документа>,  "original\_file\_name": <наименование документа без корректировки кодирования>,  "source": <источник документа>,  },  "page\_content": <текст части документа>,  "type": "Document"  },  ...  ]  } |

Листинг 7.4.4 – Финальный результат (is\_eos=True) модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий

|  |
| --- |
| {  "event\_id": <уникальный идентификатор события>,  "agent\_id": <уникальный идентификатор агента в системе>,  "description": <описание полученного события>,  "is\_eos": true,  "name": <наименование события>,  "result": <результат генерации ответа на запрос>  } |

Все возвращаемые сообщения содержат поля `event\_id`, `agent\_id`, `description`, `is\_eos`, `name`, `result`. Промежуточные результаты имеют значение поля is\_eos – False. По окончании генерации приходит результат с is\_eos – True.

* + 1. ***Интерпретация выходных данных модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ***

После запуска цепочки генерации синтетических примеров, как показано в листинге 7.4.5, можно забрать результат в виде json-объекта или сохранить на устройство. Возвращаемые поля показаны в листинге 7.4.6, где content указывает на конкретные части документов, которые использовалась при генерации примера.

Листинг 7.4.5 – Часть скрипта с вызовом и сохранением результата для SummarisationChain

summarisation\_chain = SummarisationChain(llm=llm)

result = summarisation\_chain.run(dataset, n\_examples=100)

summarisation\_chain.save\_chain\_output("data/sample\_summarization\_chain\_output.json")

Листинг 7.4.6 – Формат результата работы синтетической генерации

|  |
| --- |
| {  "generated\_example": <сгенерированный пример по данным>,  "content": [  "chunk": <данные использованные для генерации примера>,  "source\_uuid": <идентификатор источника данных>  ]  } |

1. **СООБЩЕНИЯ**

Процесс решения любой задачи средствами фреймворка сопровождается рядом сообщений от модуля логгирования. Примеры и расшифровки таких сообщений приведены в таблице 8.1.

Таблица 8.1 – Примеры сообщений, получаемых во время работы фреймворка

|  |  |
| --- | --- |
| Сообщение | Расшифровка |
| Agent's Response: … | Ответ от агента |
| Some fields in the Prompts dataclass are still None | Некорректная структура запроса |
| Task with id {task\_id} not found | Задача не найдена в кэше |
| Invoking '{job\_name}' with task id'{task\_id}'. | Начат вызов задачи |
| "Error in task '{task\_id}'. Unknown job class: '{task\_class}'." | Задан некорректный тип задач |
| The response generation has been interrupted. Error: {ex}. | Не удалось сгенерировать ответ с помощью БЯМ |
| The length of retrievers and collection\_names must match | Некорректная конфигурация RAG |
| Saving the result with the {key} prefix has been interrupted. Error: {ex}. | Ошибка при сохранении результата |
| API request failed: | Ошибка при запросе к API модели |

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Feng G. et al. Towards revealing the mystery behind chain of thought: a theoretical perspective //Advances in Neural Information Processing Systems. – 2024. – Т. 36.

[2] Liu Y. et al. G-eval: Nlg evaluation using gpt-4 with better human alignment //arXiv preprint arXiv:2303.16634. – 2023.

[3] 2. Chase, H. (2022). LangChain [Computer software]. https://github.com/langchain-ai/langchain

[4] Li X. A Review of Prominent Paradigms for LLM-Based Agents: Tool Use (Including RAG), Planning, and Feedback Learning //arXiv preprint arXiv:2406.05804. – 2024.

[5] Vijayarani S. et al. Text mining: open source tokenization tools-an analysis //Advanced Computational Intelligence: An International Journal (ACII). – 2016. – Т. 3. – №. 1. – С. 37-47.

# **ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в докум. | № документа | Входящий № сопроводительного докум. и дата | Подп. | Дата |
| Изменённых | Заменённых | Новых | Аннулированных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |