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)**

*Инв.№ подл.* GGJLKподп.

*Подп. и дата*

*Инв.№ дубл.*

*Подп. и дата*

*Взам.инв.№*

ОПИСАНИЕ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯ

RU.СНАБ.00855-01 13 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 13 01-ЛУ

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

ОПИСАНИЕ ПРОГРАММЫ

RU.СНАБ.00855-01 13 01

*Инв.№ подл.* GGJLKподп.

*Подп. и дата*

*Инв.№ дубл.*

*Подп. и дата*

*Взам.инв.№*

Листов 54

2024

АННОТАЦИЯ

В данном программном документе приведено описание фреймворка быстрого прототипирования приложений на основе больших языковых моделей (БЯМ) Proto.LLM. Для функционирования фреймворка необходимо наличие интерпретатора Python 3.10, среды разработки (опционально), а также ряда предустановленных библиотек, основные из которых - langchain, pydantic, chromadb, tornado, pypdf, redis, celery.

Основной задачей фреймворка является повышение эффективности прототипирования приложений на основе больших языковых моделей (БЯМ).

В документе представлены общие сведения о фреймворке ProtoLLM для упрощения и ускорения прототипирования систем с БЯМ на основе RAG (поисковой дополненной генерации), упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов, повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий создания сложных синтетических примеров для дообучения БЯМ.

СОДЕРЖАНИЕ

[АННОТАЦИЯ 2](#_Toc185703841)

[СОДЕРЖАНИЕ 3](#_Toc185703842)

[1. ОБЩИЕ СВЕДЕНИЯ 6](#_Toc185703843)

[2. ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ 7](#_Toc185703844)

[2.1. Назначение программного компонента 7](#_Toc185703845)

[2.2. Классы решаемых задач 7](#_Toc185703846)

[2.3. Функциональные ограничения на применение 8](#_Toc185703847)

[2.4. Область применения 8](#_Toc185703848)

[3. ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ 10](#_Toc185703849)

[3.1. Алгоритм упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов 10](#_Toc185703850)

[3.2. Алгоритм упрощения и ускорения прототипирования систем с БЯМ на основе RAG 12](#_Toc185703851)

[3.3. Алгоритм повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий 15](#_Toc185703852)

[3.4. Алгоритм создания сложных синтетических примеров для дообучения БЯМ 16](#_Toc185703853)

[3.5. Структура фреймворка 18](#_Toc185703854)

[4. ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА 26](#_Toc185703855)

[5. ВЫЗОВ И ЗАГРУЗКА 28](#_Toc185703856)

[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 28](#_Toc185703857)

[5.2. Вызов модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG 29](#_Toc185703858)

[5.3. Вызов модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий 30](#_Toc185703859)

[5.4. Вызов модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ 36](#_Toc185703860)

[6. ВХОДНЫЕ ДАННЫЕ 38](#_Toc185703861)

[6.1. Состав и структура входных данных 38](#_Toc185703862)

[6.1.1. 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 38](#_Toc185703863)

[6.1.2. Состав и структура входных данных для модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG 38](#_Toc185703864)

[6.1.3. Состав и структура входных данных для модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий 39](#_Toc185703865)

[6.1.4 *Состав и структура входных данных для модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ* 40](#_Toc185703866)

[6.2. Подготовка входных данных 41](#_Toc185703867)

[6.2.1. 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 41](#_Toc185703868)

[6.2.2. Подготовка входных данных для модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG 42](#_Toc185703869)

[6.2.3. Подготовка входных данных для модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий 45](#_Toc185703870)

[6.2.4. Подготовка входных данных для модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ 45](#_Toc185703871)

[7. ВЫХОДНЫЕ ДАННЫЕ 46](#_Toc185703872)

[7.1. Состав и структура выходных данных 46](#_Toc185703873)

[7.1.1. 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 46](#_Toc185703874)

[7.1.2. Состав и структура выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG 46](#_Toc185703875)

[7.1.3. Состав и структура выходных данных модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий 47](#_Toc185703876)

[7.1.4. Состав и структура выходных данных модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ 47](#_Toc185703877)

[7.2. Интерпретация выходных данных 47](#_Toc185703878)

[7.2.1. 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 47](#_Toc185703879)

[7.2.2. Интерпретация выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG 48](#_Toc185703880)

[7.2.3. Интерпретация выходных данных модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий 48](#_Toc185703881)

[7.2.4. Интерпретация выходных данных модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ 49](#_Toc185703882)

[Список использованной литературы 51](#_Toc185703883)

[ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ 52](#_Toc185703884)

# ОБЩИЕ СВЕДЕНИЯ

Программный комплекс - фреймворк быстрого прототипирования приложений на основе БЯМ (далее – ProtoLLM) RU.СНАБ.00855-02 81 01 предназначен для поддержки технологических операций и типовых пайплайнов работы с БЯМ, включая дообучение моделей, работу с дополненной выборкой, ансамблирование и промптинг в целях разработки систем поддержки принятия решений, обеспечивающих процессы цифровой трансформации и интеллектуализации промышленных производств.

# ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ

## 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.

## Классы решаемых задач

* Прототипирование систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов .
* Прототипирование систем с БЯМ на основе RAG (поисковой дополненной генерации).
* Повышение эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий.
* Генерация синтетических отраслевых данных для тестирования ИИ-ассистентов и СППР.

## Функциональные ограничения на применение

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.

## 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.

# ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ

## Алгоритм упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов

Алгоритм предназначен упрощения и ускорения прототипирования систем на основе БЯМ за счет повышения эффективности настройки модулей, реализующих работу с плагинами, обеспечивающими взаимодействие с внешними сервисами, сторонними приложениями, инструментами и моделями машинного обучения.

Блок-схема алгоритма показана на рисунке 2.1.1.

Изображение выглядит как текст, диаграмма, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 2.1.1 - Блок-схема алгоритма  упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов

Алгоритм 2.1.1 включает в себя следующие этапы:

1. Запрос пользователя:  на этом этапе пользователь вводит свой запрос, который детально описывает решаемую задачу, указывает доступные внешние инструменты, которые могут быть использованы в качестве плагинов, предоставляет информацию о предпочтительных технологиях и ограничениях.
2. Библиотека шаблонов для подключения плагинов содержит готовые шаблоны для взаимодействия с типовыми сервисами с REST-интерфейсом, реализациями моделей машинного обучения (включая AutoML [1]), внешними приложениями. В т.ч. библиотека поддерживает основные интерфейсы модулей библиотеки langchain [2] (взаимодействие БЯМ с инструментами через стандартные средства библиотеки), включает шаблоны для работы с библиотеками sklearn и pytorch (ML-модели), содержит шаблоны для интеграции с библиотекой pydantic (валидация данных), предоставляет шаблоны для работы с библиотекой geojson (взаимодействие с гео-сервисами через REST API).
3. Выбор типа плагина: на этом этапе на основе запроса пользователя определяется необходимый тип плагина: ML-модель, внешний сервис, приложение, RAG-система.
4. Выбор типа БЯМ: при выборе учитываются факторы производительности, специфики задачи и доступных ресурсов.
5. Выбор инфраструктуры для запуска БЯМ: определяется оптимальная вычислительная инфраструктура для выбранной БЯМ, учитываются требования к производительности и доступные ресурсы.
6. Подключение выбранного типа плагина: в зависимости от выбора на шаге 3, система инициирует процесс подключения соответствующего плагина:  ML-модель, внешний сервис, приложение, RAG-система.
7. Генерация кода для взаимодействия БЯМ и плагина: система автоматически генерирует код, обеспечивающий взаимодействие между БЯМ и выбранным плагином. Возможны два подхода к генерации кода:
   * a) Использование статичных шаблонов из библиотеки,
   * b) Применение БЯМ для решения задачи кодогенерации (требует дополнительной настройки для работы с локальными БЯМ и внешними сервисами).
8. Модификация системного запроса: производится адаптация системного запроса для корректной работы с новым плагином.
9. Авто-валидация: система проводит автоматическую проверку корректности интеграции, выполняются тесты на совместимость БЯМ и плагина, проверяется соответствие результатов ожидаемым параметрам.

Представленный алгоритм обеспечивает комплексный подход к интеграции БЯМ с различными типами плагинов, значительно упрощая процесс прототипирования и повышая эффективность разработки систем на основе больших языковых моделей.

## Алгоритм упрощения и ускорения прототипирования систем с БЯМ на основе RAG

Алгоритм предназначен упрощения и ускорения прототипирования систем на основе БЯМ за счет повышения эффективности настройки модулей, использующих техники RAG (Retrieval Augmented Generation) [3]. Повышение эффективности достигается за счет использования библиотеки шаблонов, а также интеллектуальных сценариев конфигурирования RAG под различных инструменты и задачи.

Блок-схема алгоритма показана на рисунке 2.2.1.

Изображение выглядит как текст, диаграмма, Шрифт, линия

Автоматически созданное описание

Рисунок 2.2.1 - Блок-схема алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG

Алгоритм 2.2.1 включает в себя следующие этапы:

1. Запрос пользователя: на этом этапе пользователь вводит свой запрос, который описывает данные и сценарий их использования. Запрос может включать информацию о дескрипторах данных, желаемом результате и любых дополнительных специфических требованиях.
2. Библиотека RAG-шаблонов: обращение к предварительно созданной библиотеке шаблонов RAG (Retrieval Augmented Generation). Эта библиотека содержит различные структурные шаблоны, оптимизированные под разные сценарии использования RAG, включая:
   * модульные сценарии RAG,
   * шаблоны для взаимодействия с векторными базами данных,
   * управление сложными запросами к нескольким базам данных,
3. Выбор конфигурации базы данных может включать выбор среди доступных векторных баз данных (например, ChromaDB или Elasticsearch), выбор алгоритма лексического поиска (например, BM25 или ANN).
4. Выбор модели создания сжатых представлений или выбор ембеддинга (например, Elasticsearch или multilingual-e5-large).
5. Выбор логики маршрутизации запросов к БД: этот этап включает настройку логики, определяющей, как запросы будут направляться к базе данных в случае, когда системе нужно выбрать из нескольких имеющихся БД. Это может быть реализовано через статический сценарий или с использованием более сложного агентного алгоритма (алгоритм 1.3).
6. Настройка аугментации запроса: выбирается стратегия аугментации –однократная или итеративная, в зависимости от сложности запроса и требуемой точности.
7. Авто-валидация: система проводит автоматическую валидацию настроенной конфигурации, используя открытые библиотеки (например, pydantic) для проверки корректности данных, структуры запросов и ответов системы.

Важно отметить, что алгоритм обеспечивает совместимость с основными интерфейсами модулей библиотеки langchain, что позволяет легко интегрировать его в существующие проекты, использующие эту популярную библиотеку для работы с языковыми моделями.

Этот расширенный алгоритм значительно упрощает и ускоряет процесс прототипирования систем на основе больших языковых моделей (БЯМ), автоматизируя многие сложные этапы настройки RAG и позволяя разработчикам быстро адаптировать системы под различные задачи и инструменты.

## Алгоритм повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий

Алгоритм направлен на повышение эффективности использования БЯМ путем комбинирования свойств различных агентов [4], которые могут быть различными моделями или одной моделью с разными ролями.

На вход алгоритма (см. рисунок 2.3.1) поступает запрос пользователя, который сопоставляется с набором шаблонов-агентов и соотносится с имеющимися вычислительными ресурсами под задачу. На этапе обучения компонент алгоритма будут определены универсальные подмножества агентов (например, крупные агенты на основе моделей с >70B параметрами для достижения наилучшего качества, средние агенты и малые агенты на основе легковесных моделей <7B параметров) умеющие эффективно взаимодействовать при существующих ресурсах [5, 6].

В соответствии с входным запросом обученный оркестратор выбирает из доступных подмножеств (ограниченных ресурсами) агентов, которые совместно планируют запрос. Каждый агент имеет доступ к подмножеству внешних инструментов и плаганов взаимодействие с которыми позволяет ему достичь наилучших результатов (в соответствии с этапом обучения и тестирования агентов).

В соответствии с составленным планом происходит выполнение подзапросов и агрегация результатов для вывода конечного ответа пользователю. Пользователь, в свою очередь, может оценить предоставленный ответ, что будет использоваться как источник информации об эффективности решения и, потенциально, для дообучения и улучшения агентов и их ансамблей.

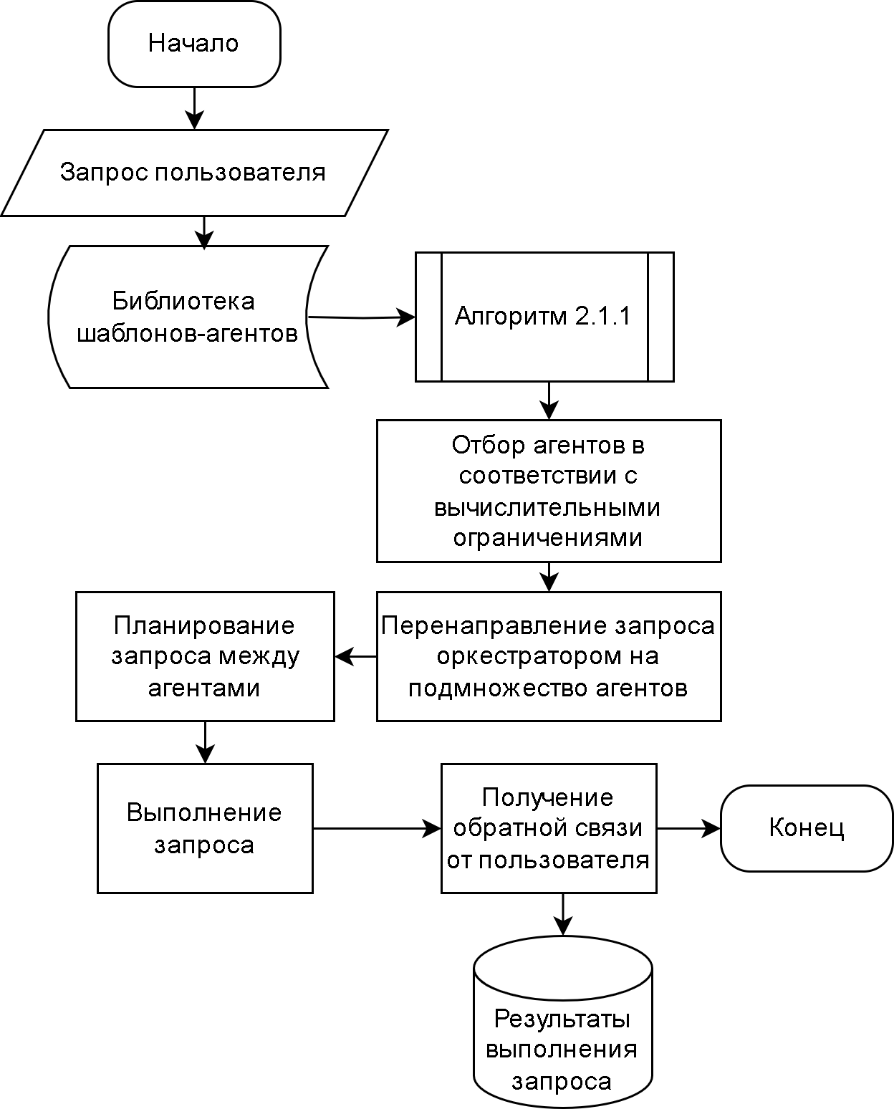


Рисунок X - Блок-схема алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий

## Алгоритм создания сложных синтетических примеров для дообучения БЯМ

Предложенный алгоритм (см. рисунок 2.4.1) концентрируется на создании сложных примеров с помощью БЯМ большего размера, а также с помощью специализированных готовых пайплайнов для решения типовых задач, но в рамках предметно-ориентированных данных [7].

На вход алгоритм получает набор данных, который будет использоваться для создания синтетических примеров, тип задачи, количество примеров для генерации, начальную инструкцию от пользователя (общее описание задачи) при наличии, а также размеченные примеры в формате пар <запрос пользователя>-<результат> также при их наличии. Также необходимо указать доступные для синтеза вычислительные ресурсы, которые определят какие модели возможно использовать.

Входные данные могут поступить либо на готовый пайплайн, либо, если готового специализированного пайплайна нет, в общий пайплайн по синтезу начальных инструкций. Специальный пайплайн для генерации сложных примеров для вопросно-ответного и диалогового ассистента по множеству документов (RAG) предусматривает генерацию сложных вопросов через поиск неявных связей в тексте путем агрегации и суммаризации информации. В свою очередь, для задачи суммаризации текстов также предусмотрен набор системных промптов и моделей обеспечивающих следование инструкциям. Также возможно расширение алгоритма новыми пайплайнами.

После генерации множества синтетических примеров при необходимости происходит их улучшение с помощью привлечения эволюционного алгоритма, способного повысить разнообразие примеров.

Итоговые синтетические примеры подвергаются эвристической пост-обработке на соответствие грамматике и синтаксису языка (в нашем случае русский и английский), а также отсутствию полных и почти-полных дубликатов. После этого полученные синтетические примеры можно выгрузить в label studio для человеческой оценки их качества.

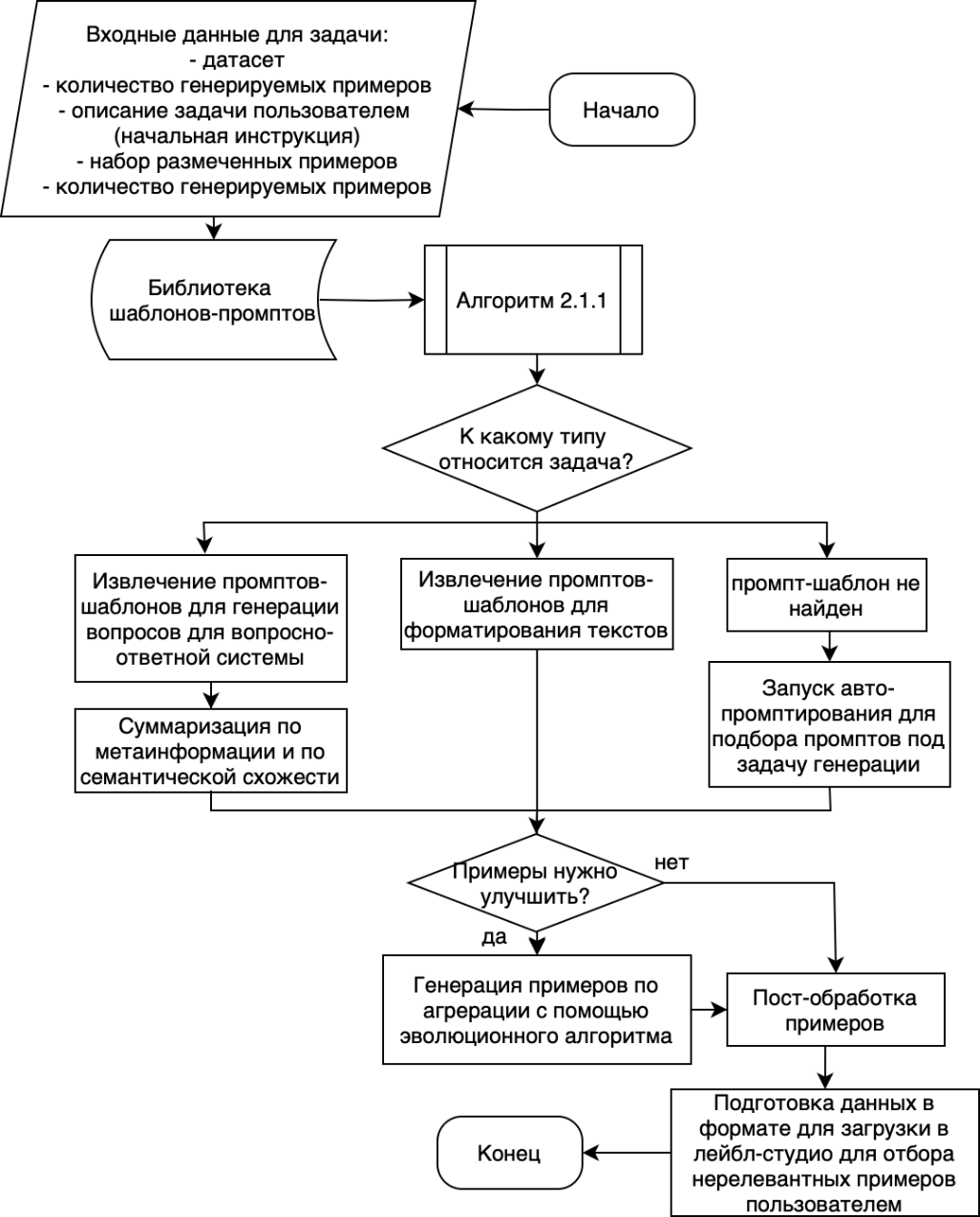


Рисунок 2.4.1 - Блок схема алгоритма создания сложных синтетических примеров для дообучения БЯМ

## Структура фреймворка

Структура ProtoLLM показана на рисунке 3.5.1. Она включает несколько компонентов – разбитых на две группы (protollm – основные компоненты фреймворка, используемые при разработке производных систем и protomllm\_tool – вспомогательные компоненты, которые запускаются отдельно на расчетном сервере). На рисунках 3.5.2 и 3.5.3 приведены диаграмма классов protollm и protomllm\_tool.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 3.5.1 – Структура программного комплекса ProtoLLM

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок 3.5.2 – Диаграмма классов ядра protollm

Изображение выглядит как текст, снимок экрана, чек, Параллельный

Автоматически созданное описание

Рисунок 3.5.3 – Диаграмма классов protollm\_tools

Фреймворк состоит из следующих компонентов:

1) компонент упрощения и ускорения прототипирования систем с БЯМ на основе RAG;

2) компонент упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов;

3) компонент   повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий;

4) компонент  создания сложных синтетических примеров для дообучения БЯМ.

5) вспомогательные компоненты для разработки серверной части приложений на основе БЯМ.

Компонент для автоматизированного подключения внешних сервисов, приложений, и моделей в виде плагинов основан на алгоритме 2.1 обеспечивает взаимодействие БЯМ с внешними инструментами, а также выбор типа БЯМ и инфраструктуры для ее выполнения. Компонент для работы с RAG, основанный на алгоритме 2.2, обеспечивает упрощение и повышение эффективности работы с внешними источниками данных.  Компонент повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий основан на алгоритме 2.3 и обеспечивает интеллектуальное взаимодействие моделей между собой. Финальным этапом использования фреймворка является создание приложения на основе БЯМ для решения конкретной предметной задачи.

1. Компонент для работы с плагинами

Этот компонент реализуется как гибкая система расширений, позволяющая легко интегрировать новые функциональности в фреймворк. Реализуется стандартизированный интерфейс для подключения плагинов, обеспечивающий единообразие и простоту создания новых расширений. API для взаимодействия плагинов с основным фреймворком предоставляет контролируемый доступ к ключевым функциям. Система версионирования плагинов позволяет поддерживать обратную совместимость и управлять обновлениями.

1. Компонент для работы с RAG

Данный компонент фокусируется на быстром разворачивании и настройке RAG-систем на основе библиотеки шаблонов. Ключевой особенностью является возможность настройки системы индексирования и векторизации документов, которая преобразует текстовые данные в числовые векторы для быстрого семантического поиска. Реализуются современные алгоритмы семантического поиска для нахождения наиболее релевантных фрагментов информации. Важную роль играют механизмы ранжирования и фильтрации результатов, обеспечивающие выбор наиболее подходящей информации. Компонент интегрируется с различными типами хранилищ данных, включая векторные и традиционные БД, обеспечивая гибкость в выборе источников информации. Для оптимизации производительности реализуется система кэширования часто запрашиваемой информации. Адаптивные стратегии извлечения информации позволяют динамически корректировать процесс поиска в зависимости от контекста и специфики задачи.

1. Компонент повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий

Этот компонент обеспечивает создание и управление сложными системами, состоящими из множества взаимодействующих агентов на основе БЯМ и иных приложений. Компонент реализует функциональность для управления мультиагентными системами, позволяя автоматизировать внедрение агентов в систему, настройку агентов, а также взаимодействие агентов в составе системы. Компонент организует выполнение задач через создание настраиваемых пайплайнов и предоставляет возможности к централизованному управлению доступными в системе ресурсами (модели, в том числе, БЯМ, внешние сервисы, БД). Оптимизация работы с агентами достигается как за счет ансамблирования агентов средствами ансамбль агента и маршрутизации вызовов агентов посредством агента роутера, так и благодаря эффективному использованию общего пула заданных ресурсов. Пример диаграммы последовательности для запросов к отдельному агенту представлен на Рисунке 3.5.4, пример маршрутизации запросов пользователя агентом роутером представлен на Рисунке 3.5.5. Компонент также предоставляет возможность для конфигурирования и запуска API-сервиса, через который доступны REST и Websocket эндпойнты для просмотра и вызова агентов. Функциональность SDK компоненты может быть использована отдельно в целях интеграции агентов, агента-роутера и ансамблевого агента в существующие или разрабатываемые системы мультиагентных решений.

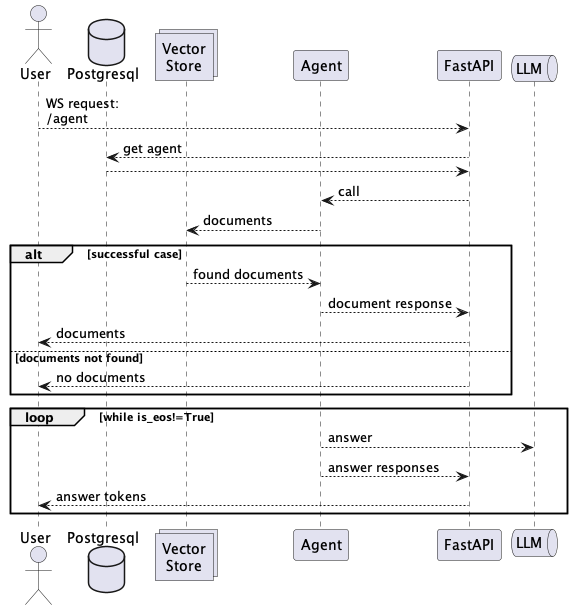


Рисунок 3.5.4 – UML Диаграмма последовательности обработки запросов пользователя отдельным агентом

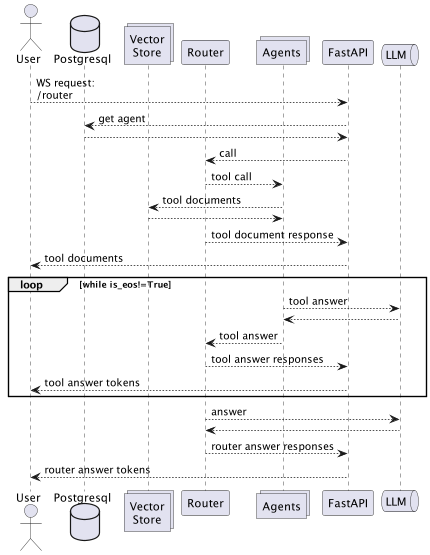


Рисунок 3.5.5 – UML Диаграмма последовательности обработки запросов пользователя агентом-роутером

1. Компонент для работы с синтетическими данными

Этот компонент играет ключевую роль в улучшении качества и разнообразия данных для обучения и тестирования БЯМ. Реализуются генераторы различных типов синтетических данных, способные создавать тексты, имитирующие реальные данные. Важной особенностью являются алгоритмы аугментации данных, которые позволяют расширять существующие наборы данных путем создания их модифицированных версий. Компонент включает механизмы валидации и оценки качества синтетических данных, обеспечивающие их соответствие требованиям и реалистичность. Интеграция с системами машинного обучения позволяет автоматически генерировать данные, основываясь на текущих потребностях модели и выявленных паттернах.

1. Сервисные компоненты для работы с моделями (включая развернутые локальные модели) и реализации внешнего API для приложения.

Сервисные компоненты обеспечивает унифицированный доступ к различным языковым моделям, как внешним, так и локальным. Ключевой особенностью является единый интерфейс для работы с разными моделями, что упрощает интеграцию и замену моделей в приложениях. Для оптимизации работы с внешними API реализуется система кэширования и оптимизации запросов, снижающая нагрузку на внешние сервисы и улучшающая отзывчивость системы. Механизмы балансировки нагрузки позволяют эффективно распределять запросы между различными провайдерами API, обеспечивая высокую доступность и производительность. Эти компоненты развертываются отдельно (на сервере, где запущены модели).

# ИСПОЛЬЗУЕМЫЕ ТЕХНИЧЕСКИЕ СРЕДСТВА

Фреймворк Proto.LLM разработан на языке Python 3.10 и используется с применением среды разработки PyCharm или аналогичной. В таблице 4.1 приведён перечень внешних библиотек и инструментов разработки, отладки и сопровождения, необходимых для его использования.

Таблица 4.1 – Языки и технологии разработки

|  |  |
| --- | --- |
| Внешние библиотеки | Вспомогательные инструменты для разработки, отладки, сопровождения |
| langchain-core версия не ниже 0.3.0  langchain-community версия не ниже 0.3.3  langchain-chroma версия не ниже 0.1.4  chromadb версия не ниже 0.5.3  fastapi версия не ниже 0.111.1  accelerate версия не ниже 0.29.3  numpy версия не ниже 1.26.4  pandas версия не ниже 1.3.0  celery версия не ниже 5.4.0  sqlalchemy версия не ниже 2.0.36  asyncpg версия не ниже 0.30.0  redis версия не ниже 5.2.1  pydantic версия не ниже 2.10.3  pydantic-settings версия не ниже 2.6.1  uvicorn версия не ниже 0.32.1  orjson версия не ниже 3.10.12  websockets версия не ниже 14.1  pyyaml версия не ниже 6.0.2  langchain-openai версия не ниже 0.2.12  elasticsearch версия не ниже 8.17.0  transformers версия не ниже 4.47.1  torch версия не ниже 2.2.2  langgraph версия не ниже 0.2.60  langfuse версия не ниже 2.57.0 | pytest версия не ниже 8.2.  testfixtures версия не ниже 6.18  pytest-docker версия не ниже 3.1.1  pytest-dotenv версия не ниже 0.5.2  psycopg2-binary версия не ниже 2.9.10 |

Для обеспечения работы содержательных функций фреймворка ProtoLLM, необходимо использовать вычислительный сервер под управлением ОС Linux со следующими характеристиками:

1. Процессор (CPU):

* Минимальная конфигурация: 8-ядерный процессор (например, Intel Core i7 или AMD Ryzen 7).
* Рекомендуемая конфигурация: 16+ ядер, серверные процессоры типа Intel Xeon или AMD EPYC для работы с большими объемами данных и множеством параллельных процессов.

1. Оперативная память (RAM):

* Минимальная конфигурация: 32 ГБ.
* Рекомендуемая конфигурация: 64-128 ГБ или более, в зависимости от размера моделей и объема обрабатываемых данных.

1. Графический процессор (GPU):

* Минимальная конфигурация: NVIDIA GPU с 8 ГБ VRAM (например, GTX 1070 или выше).
* Рекомендуемая конфигурация: NVIDIA Tesla V100, A100 или аналогичные профессиональные GPU с 16-32 ГБ VRAM или более.

1. Система хранения:

* Минимальная конфигурация: HDD 500 ГБ.
* Рекомендуемая конфигурация: SSD 1-2 ТБ для быстрого доступа к данным и моделям.
* Сеть: высокоскоростное подключение к интернету (1 Гбит/с или выше) для загрузки моделей и работы с облачными сервисами.

# ВЫЗОВ И ЗАГРУЗКА

## 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. 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.

## 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.

## 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 5.1.3.3 – 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 5.1.3.1 конфигурация пула ресурсов представлена в конфигурационном файле Листинга 5.1.3.3. 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 5.1.3.2.

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.

## 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.

# ВХОДНЫЕ ДАННЫЕ

## Состав и структура входных данных

### 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 6.1.1 указаны входные данные, необходимые для корректной работы модуля.

Таблица 6.1.1 – 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 | - |

### 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 6.1.2 указаны типы, структура, а также способы ввода и ограничения входных данных.

Таблица 6.1.2 – 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 | - |

### 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 6.1.3 указаны типы, структура, а также способы ввода и ограничения входных данных.

Таблица 6.1.3 – 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 |

## 6.1.4 *Состав и структура входных данных для модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ*

Таблица 6.1.4 – 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 | - |

## Подготовка входных данных

### 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 6.2.1 – 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 6.2.2 – 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) |

### 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 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 6.2.3.

Листинг 6.2.3 – 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 6.2.4). 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 6.2.4 – 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 6.2.5). 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 6.2.5 – 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.

### 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 компоненты повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий не требуется.

### Подготовка входных данных для модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ

Подготовка документов аналогична подготовке данных, описанных в 6.2.2, если пользователь не воспользовался готовым датасетом или не произвел обработку своими средствами. При использовании функций protollm необходимо отметить, что в листинге 6.2.4 необходимо внести небольшие изменения – убрать секции чанкования и токенизации и сохранить полученные распаршенные 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».

# ВЫХОДНЫЕ ДАННЫЕ

## Состав и структура выходных данных

### 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 7.1.1 - 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 | - |

### Состав и структура выходных данных модуля на основе алгоритма упрощения и ускорения прототипирования систем с БЯМ на основе RAG

В таблице 7.1.2 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 7.1.2 – 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 |

### Состав и структура выходных данных модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий

В Таблице 7.1.3 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 эндпойнта Websocket приходят промежуточные результаты, а также конечный результат генерации.

Таблица 7.1.3 – 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 | - |

### Состав и структура выходных данных модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ

В таблице 7.1.4 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 7.1.4 – 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 | - |

## Интерпретация выходных данных

### 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 7.2.1. Финальный ответ и промежуточные шаги представляют собой ответ на естественном языке.

Листинг 7.2.1 – Использование модуля сбора данных результатов эксперимента

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

### 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 7.2.2. 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 7.2.2 – 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) |

### Интерпретация выходных данных модуля на основе алгоритма повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий

Структура выходного словаря представлена в листингах 7.2.3 – 7.2.4. 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 7.2.3 – 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 7.2.4 – 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.

### Интерпретация выходных данных модуля на основе алгоритма создания сложных синтетических примеров для дообучения БЯМ

После запуска цепочки генерации синтетических примеров, как показано в листинге 7.2.5, можно забрать результат в виде json-объекта или сохранить на устройство. Возвращаемые поля показаны в листинге 7.2.6, где content указывает на конкретные части документов, которые использовалась при генерации примера.

Листинг 7.2.5 – 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 7.2.6 – Формат результата работы синтетической генерации

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

# Список использованной литературы

1. Tornede A. et al. Automl in the age of large language models: Current challenges, future opportunities and risks //arXiv preprint arXiv:2306.08107. – 2023.

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

3. 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.

4. Fang C. et al. Llm-ensemble: Optimal large language model ensemble method for e-commerce product attribute value extraction //Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. – 2024. – С. 2910-2914.

5. Li J. et al. More agents is all you need //arXiv preprint arXiv:2402.05120. – 2024.

6. Shen W. et al. Small llms are weak tool learners: A multi-llm agent //arXiv preprint arXiv:2401.07324. – 2024.

7. Gan Z., Liu Y. Towards a Theoretical Understanding of Synthetic Data in LLM Post-Training: A Reverse-Bottleneck Perspective //arXiv preprint arXiv:2410.01720. – 2024.

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Лист регистрации изменений | | | | | | | | | |
| Изм. | Номера листов (страниц) | | | | Всего листов (страниц) в докум. | № документа | Входящий № сопроводи-тельного докум. и дата | Подп. | Дата |
| изме-ненных | заме-ненных | новых | Аннули-рован-ных |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |