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)

ПРОГРАММА И МЕТОДИКи ПРИЕМОЧНЫХ ИСПЫТАНИЙ

ЛИСТ УТВЕРЖДЕНИЯ

Инв.№подл. GGJLKподп.

Подп. и дата

Инв.№ дубл.

Подп. и дата

Взам.инв.№

Инв.№подл. GGJLKподп.

Подп. и дата

Инв.№ дубл.

Подп. и дата

Взам.инв.№

RU.СНАБ.00855-01 51 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 51 01-ЛУ

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

ПРОГРАММА И МЕТОДИКи ПРИЕМОЧНЫХ ИСПЫТАНИЙ

RU.СНАБ.00855-01 51 01

Инв.№подл. GGJLKподп.

Подп. и дата

Инв.№ дубл.

Подп. и дата

Взам.инв.№

Инв.№подл. GGJLKподп.

Подп. и дата

Инв.№ дубл.

Подп. и дата

Взам.инв.№

Листов 42

2024

АННОТАЦИЯ

Документ содержит программу и методику приемочных испытаний программного комплекса – фреймворк Proto.LLM быстрого прототипирования приложений на основе больших языковых моделей (БЯМ). Фреймворк Proto.LLM входит в состав открытого ПО, разрабатываемого в рамках плана Исследовательского центра в сфере искусственного интеллекта «Сильный ИИ в промышленности (ИЦ ИИ) в рамках соглашения с АНО «Аналитический центр при Правительстве Российской Федерации» (ИГК 000000D730324P540002), № 70–2022–00141, с целью повышения эффективности прототипирования приложений на основе больших языковых моделей (БЯМ).

Содержание

1. ОБЪЕКТ ИСПЫТАНИЙ 4

2. ЦЕЛЬ ИСПЫТАНИЙ 4

3. ТРЕБОВАНИЯ К ПРОГРАММЕ 4

4. ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ 4

5. СРЕДСТВА И ПОРЯДОК ИСПЫТАНИЙ 4

6. МЕТОДЫ ИСПЫТАНИЙ 4

6.1. 4

6.2. 4

# ОБЪЕКТ ИСПЫТАНИЙ

Фреймворк быстрого прототипирования приложений на основе больших языковых моделей (БЯМ).

Обозначение – ProtoLLM.

# ЦЕЛЬ ИСПЫТАНИЙ

Целью испытаний является проверка соответствия фреймворка Proto.LLM требованиям xfcnyjuj технического задания (приложение Д) в составе Программы исследовательского центра в сфере искусственного интеллекта "Сильный искусственный интеллект в промышленности" (далее ИЦ СИИП).

# ТРЕБОВАНИЯ К ПРОГРАММЕ

Испытания проводятся на открытом фреймворке Proto.LLM, код которого доступен по ссылке <https://github.com/aimclub/ProtoLLM>.

# ТРЕБОВАНИЯ К ПРОГРАММНОЙ ДОКУМЕНТАЦИИ

Настоящая Программа и методика испытаний разработана в соответствии со следующими документами:

* ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению;
* ГОСТ 34.603-92 Виды испытаний автоматизированных систем.

Программная и эксплуатационная документация фреймворка Proto.LLM должна соответствовать комплектности предоставления документов ИЦ СИИП, быть оформленной в соответствии с требованиями ЕСПД и представленной на бумажных и/или электронных носителях.

# СРЕДСТВА И ПОРЯДОК ИСПЫТАНИЙ

В испытаниях принимают участие специалисты ИЦ СИИП (Университет ИТМО), Ассоциации “Искусственный интеллект в промышленности”.

Испытания проводятся на технических средствах ИТМО по адресу: 199034, С-Петербург, Биржевая линия д. 14.

Для проведения испытаний необходима вычислительная система 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.

Для проведения испытаний необходимо наличие на компьютере интерпретатора python версии 3.10.

Для проведения испытаний необходимо установить библиотеку Proto.LLM на ПК для проведения испытаний. Например, через команду *pip install protollm.*

Для проведения испытаний необходимо провести подготовительные работы по разворачиванию большой языковой модели и вспомогательных инструментов из protollm\_tools:

1. Для разворачивания БЯМ нужно запустить protollm-api и, как минимум, один инстанс protollm-worker, если используется локальная llm и контекстный сервис запросов к БЯМ ‘protollm\_sdk.jobs.job\_context.JobContext.llm\_api’.
2. В случае использования внешней БЯМ этот шаг можно пропустить и использовать ‘protollm\_sdk.jobs.job\_context.JobContext.outer\_llm\_api’. Для разворачивания protollm-api нужно перейти в директорию protollm\_tools/llm-api и запустить контейнеры reddis, RabbitMQ и контейнер с API, воспользовавшись protollm\_tools/llm-api/docker-compose.yml файлом и утилой docker-compose к docker. После этого нужно перейти в директорию protollm\_tools/llm-worker/deployment и также создать и запустить контейнер, используя файл protollm\_tools/llm-worker/deployment/docker-compose.yml. Перед запуском контейнеров нужно скорректировать параметры в docker-compose.yml файле, указывающие на путь до весов модели. Контейнер llm-worker запускается на вычислительном устройстве с одной или несколькими nvidia, так, чтобы суммарной видеопамяти хватало на разворачивание выбранной модели. Для использования внешней БЯМ необходимо указать ключ доступа к vsegpt, хранящийся в файле API\_KEY.txt на ПК для проведения испытаний.
3. После обеспечения работы БЯМ для исполнения запросов RAG необходимо поднять контекстные сервисы SDK. Для этого нужно пройти в директорию deployment/sdk-base и запустить контейнеры собрав и запустив контейнеры используя файл deployment/sdk-base/docker-compose.yml. В случае если для llm-api и контекстных сервисов sdk планируется использовать общие Redis и RabbitMQ, можно собрать все кроме них, но указать их в настройках celery и flower.
4. Последним шагом настройки инфраструктуры для осуществления испытаний, является разворачивание контейнера с векторной базой данных ChromaDB. Для этого требуется воспользоваться protollm\_tools/deployment/embedding\_sevice/docker-compose.yml.

# МЕТОДЫ ИСПЫТАНИЙ

Проверяется возможность применения версии фреймворка ProtoLLM для решения различных задач в рамках прототипирования приложения на основе больших языковых моделей.

Программа считается выдержавшей испытание при условии, что результат выполнения методик данного пункта совпадает с ожиданиями, описанными по ходу текста.

## Комплексная проверка возможности интеграции БЯМ без необходимости внесения изменений в архитектуру приложения и интеграции с инструментами для оценки и сравнения эффективности различных конфигураций системы на основе БЯМ

Лиза -> Ярослав

Пп. ТЗ:

Д3.2.2 Система должна поддерживать интеграцию БЯМ без необходимости внесения изменений в архитектуру приложения.

Д3.2.9 Фреймворк должен поддерживать интеграцию с инструментами для оценки и сравнения эффективности различных конфигураций системы на основе БЯМ.

Цель испытания:

Испытание направлено на оценку функциональных возможностей фреймворка ProtoLLM для интеграции БЯМ без необходимости внесения изменений в архитектуру приложения и интеграции с инструментами для оценки и сравнения эффективности различных конфигураций системы.

Одним из ключевых преимуществ является поддержка различных больших языковых моделей (БЯМ), включая как локальные, так и облачные решения на сторонних сервисах. Для обеспечения высокой эффективности работы фреймворка предусмотрена возможность оценки качества ответов, что позволяет выбирать наиболее подходящую БЯМ.

Способ испытаний.

Процесс оценки качества работы фреймворка включает следующие этапы:

1. Импортировать реализованные в системе классы, обеспечивающие интеграцию с БЯМ.

import os

import uuid

from protollm\_sdk.jobs.llm\_api import LLMAPI

from protollm\_sdk.models.job\_context\_models import PromptModel

from protollm\_sdk.jobs.outer\_llm\_api import OuterLLMAPI

from dotenv import load\_dotenv

from deepeval.models.base\_model import DeepEvalBaseLLM

from deepeval.test\_case import LLMTestCaseParams, LLMTestCase

from deepeval.metrics import GEval

from openai import OpenAI

from openai.\_types import NOT\_GIVEN

from dataclasses\_json import dataclass\_json

from dataclasses import dataclass

from typing import Dict

def local\_llm(question: str,

meta: dict,

host: str,

port: str | int):

llmapi = LLMAPI(llm\_api\_host=host, llm\_api\_port=port)

llm\_request = PromptModel(job\_id=str(uuid.uuid4()), meta=meta, content=question)

res = llmapi.inference(llm\_request)

return res.content

def outer\_llm(question: str, meta: dict, key: str, model: str):

llmapi = OuterLLMAPI(key, model)

llm\_request = PromptModel(job\_id=str(uuid.uuid4()), meta=meta, content=question)

res = llmapi.inference(llm\_request)

return res.content

if \_\_name\_\_ == "\_\_main\_\_":

load\_dotenv("config.env")

q = "Что необходимо обеспечить в рамках формирования улично-дорожной сети (УДС) Санкт-Петербурга?"

context = "В рамках указанной задачи необходимо обеспечить формирование опорного каркаса улично-дорожной сети (далее - УДС) Санкт-Петербурга за счет развития современной транспортной инфраструктуры, повышения связности объектов транспортной инфраструктуры, увеличения пропускной способности, развития магистралей непрерывного движения и магистралей с улучшенными условиями движения, вылетных магистралей, строительства многоуровневых развязок и транспортно-пересадочных узлов."

question = f"Question: {q}; Context: {context}"

cloud\_llm = “meta-llama/llama-3.1-70b-instruct”

local\_llm = "10.32.15.21"

local\_llm\_port = 6672

meta = {"temperature": 0.05, "tokens\_limit": 4096, "stop\_words": None}

key = os.environ.get("VSE\_GPT\_KEY")

local\_llm\_ans = local\_llm(question, meta, local\_llm, local\_llm\_port)

outer\_llm\_ans = outer\_llm(question, meta, key, cloud\_llm)

metric\_local = correctness\_metric.measure(

LLMTestCase(

input=question,

actual\_output=local\_llm\_ans,

expected\_output="В рамках формирования улично-дорожной сети (УДС) Санкт-Петербурга необходимо обеспечить формирование опорного каркаса УДС за счет развития современной транспортной инфраструктуры, повышения связности объектов транспортной инфраструктуры, увеличения пропускной способности, развития магистралей непрерывного движения и магистралей с улучшенными условиями движения, строительства многоуровневых развязок и транспортно-пересадочных узлов.",

retrieval\_context=None,

)

)

metric\_outer = correctness\_metric.measure(

LLMTestCase(

input=question,

actual\_output=outer\_llm\_ans,

expected\_output="В рамках формирования улично-дорожной сети (УДС) Санкт-Петербурга необходимо обеспечить формирование опорного каркаса УДС за счет развития современной транспортной инфраструктуры, повышения связности объектов транспортной инфраструктуры, увеличения пропускной способности, развития магистралей непрерывного движения и магистралей с улучшенными условиями движения, строительства многоуровневых развязок и транспортно-пересадочных узлов.",

retrieval\_context=None,

)

)

print(f"Question: {q}")

if metric\_local < metric\_outer:

print(f"Ответ VseGPT LLM: \n {outer\_llm\_ans}")

print(f"Metric outer: {metric\_outer}")

else:

print(f"Ответ локальной LLM: \n {local\_llm\_ans}")

print(f"Metric local: {metric\_local}")

1. Определить метрики оценки и настроить их параметры.
2. Выполнить расчет метрик на ответах, сгенерированных выбранными БЯМ.

**Критерии успеха:**

* + - 1. Получение ответа от используемой БЯМ.
      2. Получение значения метрики качества ответа.

## Комплексная проверка возможности создания прототипов RAG систем с автоматизацией процессов индексации, обработки и загрузки данных в векторные БД, а также возможностью настройки процесса retrieval в системах на основе RAG

Настя

Пп. ТЗ:

Д3.2.1 Фреймворк должен обеспечивать возможность создания прототипов агентных и RAG систем с использованием БЯМ за счет высокоуровневых интерфейсов и библиотек шаблонов для работы с БЯМ и RAG.

Д3.2.4 Фреймворк должен предоставлять инструменты для автоматизации процессов индексации, обработки и загрузки данных в векторные БД.

Д3.2.8 Фреймворк должен обеспечивать возможность настройки процесса retrieval в системах на основе RAG.

Проверка направлена на оценку возможности использования фреймворка для создания прототипов RAG систем с использованием БЯМ с применением автоматизированных процессов индексации, обработки и загрузки данных в векторные БД, а также настройки процесса извлечения информации (retrieval) из баз данных.

Способ испытаний.

1. Для осуществления испытаний используется локальная БЯМ и контейнер с векторной базой данных ChromaDB, развернутые в рамках подготовительного этапа (см. раздел «Средства и порядок испытаний»).

2. Для обеспечения автоматизированной индексации, обработки и загрузки данных в БД, требуется прописать инструкцию с параметрами для этого процесса. Она описывается в файле с названием docs\_processing\_config.yaml. 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”. Пример оформления инструкции описан в листинге 6.2.1.

Листинг 6.2.1 – Пример файла с описанием настройки процессов индексации, обработки и загрузки данных

|  |
| --- |
| 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' |

3. После подготовки файла с описанием настроек обработки документов необходимо подготовить файл с переменными окружения с названием chroma.env. Этот файл также необходимо поместить в директорию /config\_files, рядом с файлом docs\_processing.yaml. В нем требуется прописать IP-адрес и порт базы данных ChromaDB, критерий, по которому будут сравниваться документы, названия коллекций, где будут лежать загруженные документы, а также название и URL-адрес модели токенизации. Пример описания такого файла изображен в листинге 6.2.2.

Листинг 6.2.2 – Пример файла с описанием переменных окружения

|  |
| --- |
| 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' |

4. После готовности файлов с настройками и переменными окружения необходимо запустить скрипт загрузки (см. листинг 6.2.3). Перед загрузкой требуется подтянуть все настройки и переменные окружений из файлов. Для этой задачи необходимо создать объект класса ChromaSettings. В качестве входных параметров принимается путь до файла "chroma.env". Обработка документов и их загрузка происходят при помощи функции load\_documents\_to\_chroma\_db. В качестве входных параметров принимается объект класса ChromaSettings.

Листинг 6.2.3 – Пример кода обработки документов и загрузки их в базу данных

|  |
| --- |
| from os.path import dirname from pathlib import Path  from stairs\_rag.settings.chroma\_settings import ChromaSettings from stairs\_rag.stores.chroma.chroma\_loader import load\_documents\_to\_chroma\_db  db\_settings = ChromaSettings(\_env\_file=Path(dirname(\_\_file\_\_), 'configs', 'chroma.env'))  load\_documents\_to\_chroma\_db(settings=db\_settings) |

5. После того, как загружены все документы, для работы с RAG системой необходимо 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.

6. 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.

В листинге 6.2.4 приводится скрипт запуска системы RAG.

Листинг 6.2.4 – Пример кода запуска RAG-системы для ответа на пользовательский запрос.

|  |
| --- |
| 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?', use\_advanced\_rag=False)  # Шаг 4. Получение ответа модели из базы данных.  rd = get\_reddis\_wrapper()  result = load\_result(rd, job\_id, job\_name) |

**Критерий успеха.**

Результатом прохождения испытания является получение ответа от RAG системы на основе документов, загруженных в ChromaDB.

## Комплексная проверка возможности создания прототипов агентных систем с унифицированным способом для подключения внешних сервисов и приложений в виде плагинов

Алина:

Д3.2.1 Фреймворк должен обеспечивать возможность создания прототипов агентных и RAG систем с использованием БЯМ за счет высокоуровневых интерфейсов и библиотек шаблонов для работы с БЯМ и RAG.

Д3.2.3 Фреймворк должен предоставлять унифицированный способ для подключения внешних сервисов и приложений в виде плагинов.

Д3.2.6 Фреймворк должен поддерживать реализацию агентов для повышения эффективности работы БЯМ.

Проверка направлена на оценку возможности прототипирования агентных систем, которые должны позволять подключать и управлять получением информации из внешних сервисов и приложений.

**Способ испытаний:**

Для проверки необходимо использовать химический пайплайн single-agent системы. В нем используется класс Llama31ChatModel для обращений к удаленной модели через библиотеку requests. Для реализации агентной логики используется библиотека LangChain, а именно классы AgnetExecutor, ChatPromptTemplate, MessagesPlaceholder, SystemMessagePromptTemplate, HumanMessagePromptTemplate и модуль create\_structured\_chat\_agent:

1. Необходимо открыть директорию:

<https://github.com/aimclub/ProtoLLM/tree/main/examples/real_world/chemical_pipeline>

1. Скачать файл

[https://github.com/aimclub/ProtoLLM/blob/main/examples/real\_world /chemical\_pipeline/llama31\_chemical\_example.py](https://github.com/aimclub/ProtoLLM/blob/main/examples/real_world%20/chemical_pipeline/llama31_chemical_example.py),

в 274 строке указать ключ доступа к vsegpt. В строке 301 указать актуальное расположение файла с тестовыми запросами (включает 5 запросов)

<https://github.com/aimclub/ProtoLLM/blob/main/examples/real_world/chemical_pipeline/queries_responses_chemical.xlsx> или с более крупным датасетом <https://github.com/aimclub/ProtoLLM/blob/main/examples/real_world/chemical_pipeline/queries_responses_chemical_large.xlsx> .

1. Запустить файл из заранее подготовленного виртуального окружения, командой:

*$ python llama31\_chemical\_example.py*

После выполнения данной команды и запуска скрипта будет производиться чтение

заранее подготовленных запросов из excel файла. В нем же, в колонке *decomposers\_tasks* будет отображена вся цепочка действий, выполненная агентом. Последнее действие - итоговый ответ системы.

**Критерии успеха:**

1. В логах присутствует информация о предложенных LLM инструментах для вызова, например:

*Action:```{ "action": "gen\_mols\_alzheimer", "action\_input": { "num": 5 } }```*

1. Отсутствует уведомление об ошибках подключения к БЯМ (в случае *403 Client Error* рекомендуется проверить валидность ключа).

## Проверка возможности создания, управления и использования ансамблей языковых моделей

Маша

Д3.2.12 Фреймворк должен обеспечивать инструменты для создания, управления и использования ансамблей языковых моделей.

**Цель испытаний:**

Проверка направлена на оценку способности обеспечения инструментов создания, управления и использования ансамблей языковых агентов средствами фреймворка Proto.LLM.

**Способ испытаний:**

Для проверки необходимо подготовить данные в виде документов из двух схожих доменных областей, а также подготовить общий пул моделей, используемых агентами. Также должны быть подготовлены сами агенты системы, которые реализуют интерфейсы базового класса StreamingAgent.

1) В рамках подготовки векторной базы необходимо создание 3 коллекций: коллекция первого домена; коллекция второго домена;

коллекция, объединяющая документы из двух доменов.

Каждый документ необходимо загрузить в базу данных соответствующего ему домена и в базу с объединенными доменами. Для этого необходимо выполнить чанкирование документа и использовать модель эмбеддер, который будет указан в общем пуле моделей. Количество чанков в объединенной коллекции должно равняться сумме количества чанков доменных коллекций.

2) В рамках проверки в качестве моделей БЯМ, используемых для генерации, будет применяться модель Qwen2.5-72b. В рамках подготовки пула необходимо:

- Получить файлы, содержащие веса моделей, конфигурационные файлы, метаданные. Веса модели могут быть получены с применением утилиты git lfs с https://huggingface.co/Qwen/Qwen2.5-7B-Instruct.

- Подготовить конфигурационный файл балансировщика нагрузки.

- Подготовить манифест docker-compose, в качестве сервисов манифеста должны быть указаны: балансировщик нагрузки, два сервиса сервинга моделей. Аргументы сервисов должны соответствовать модели Qwen2.5-72b, должны быть выставлены значения ключей API, тома, а также номера используемых CUDA карт.

- Docker-compose манифест должен быть запущен на кластере с помощью команды ```docker compose up -d --build```.

3) Для корректного проведения проверки мультиагентной системы необходимо создать подготовить агентов, для этого:

- Создается класс RAGAgent на базе пайплайна Langchain, который соответствует предоставленным интерфейсам StreamingAgent класса SDK компоненты;

- Подготавливаются три агента типа RAG. Для этого создается конфигурационный файл YAML, в который вносятся аргументы агента, в том числе путь импортируемого плагина в виде класса RAGAgent. Пример конфигурационного файла представлен в Листинге 6.4.1.1. Инициализация агентов производится автоматически на старте API сервиса компоненты. Код класса агента должен существовать в PYTHONPATH запускаемого приложения для корректного импортирования по указанному в конфигурационном файле пути;

- Агент-роутер, обеспечивающий маршрутизацию запросов между существующими в системе агентами для оптимизации запросов и ресурсов, является частью SDK. Отдельное добавление агента-роутера для выполнения проверки не требуется;

- Ансамблевый агент отвечает за ансамблирование результатов работы агентов в рамках мультиагентной системы, является частью SDK и также не требует добавления для выполнения проверки;

- Также в конфигурационный файл (protollm\_tools/llm-agents-api/examples/admin-config.yml) необходимо внести модели БЯМ, модели эмбеддингов, векторные базы данных, используемых агентами.

4) Для корректной работы с агентами через API сервиса необходимо добавление файла с инициализацией объекта Entrypoint из конфигурационного файла (protollm\_tools/llm-agents-api/examples/main.py). Запуск производится командой ```poetry run python main.py```.

5) Подготавливается таблица с вопросами. Набор вопросов составляется с учетом специфики домена данных. С применением Postman осуществляются запросы по всем вопросам на эндпойнт `/agent` с направлением к агентам rag\_domain1, rag\_domain2, rag\_domain3 посредством указания их идентификаторов. На Рисунке 6.4.1.1 представлен пример Postman запроса к агентам (без маршрутизации и ансамблирования).

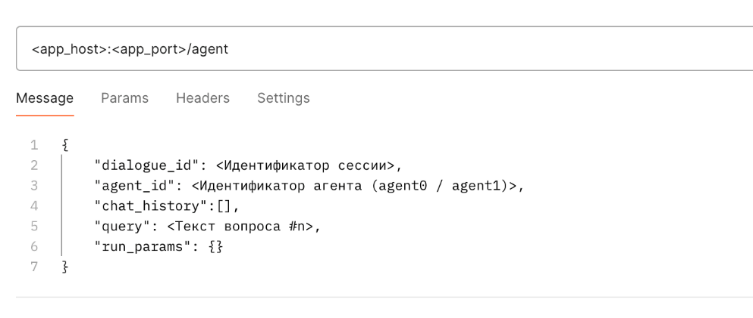


Рисунок 6.4.1.1 - Пример Postman запроса к серверу API компоненты модуля повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий

* При получении ответов, содержащих документы, релевантные запросу, документы заносятся в таблицу результатов. Результат генерации (поле `result`), полученный финальным сообщением с идентификатором `is\_eos=True`, добавляется в таблицу результатов.

- Аналогично осуществляются запросы по всем вопросам на эндпойнты `/router` и `/ensemble` с направлением к агентам роутинга и ансамблирования, соответственно. На Рисунке 6.4.1.2 представлен пример Postman запроса к агентам (без маршрутизации и ансамблирования).

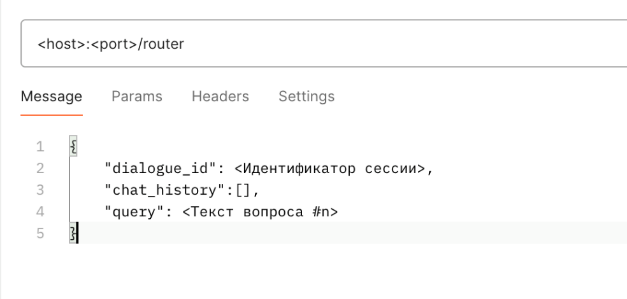


Рисунок 6.4.1.2 - Пример Postman запроса к агенту-роутеру API сервиса компоненты модуля повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий.

Порядок возвращаемых сообщений может отличаться. При получении сообщения с идентификатором `is\_eos=True`, результат генерации (поле `result`) добавляется в таблицу результатов. Промежуточный результат, содержащий извлеченные документы также добавлятся в таблицу результатов.

6) Значения в таблице результатов должны быть оценены в соответствии со следующими критериями:

* Ответ не содержит избыточного количества повторений.
* Ответ имеет логически завершенную структуру (нет бессмысленного продолжения текста).
* Ответ представлен на русском языке.
* Отсутствуют грамматические и стилистические ошибки, искажающие смысл.
* Полнота ответа (В ответе модели содержится ответ на поставленный вопрос)

Ключевым критерием оценки качества работы агентов является число полных ответов агента.

Критерии успеха:

1. В выводе консоли балансировщика нагрузки и контейнеров сервинга моделей отсутствуют ошибки, он содержит записи об успешном старте процессов-воркеров.
2. В выводе консоли запущенного приложения есть информация о старте uvicorn сервера на указанном в конфигурационном файле порту (8000). Также после успешного запуска можно увидеть OpenAPI документацию по адресу [http://0.0.0.0:8080/docs](http://0.0.0.0:8000/docs).
3. По результатам запроса на вебсокет запущенного приложения (/agent) агенты должны вернуть набор промежуточных сообщений, которые включают документы-источники, а также стриминговый ответ - результат генерации. В Листинге 6.4.2.5 представлены пример ожидаемого сгенерированного ответа системы, который должен быть в потоке сообщений последним.
4. Таблица результатов, составленная по результатам генерации, содержит следующие колонки: вопрос; ответ агента №1; документы, использованные агентом №1; ответ агента №2; документы, использованные агентом №2; ответ агента №3; документы, использованные агентом №3; ответ роутер-агента; документы, использованные роутер-агентом; ответ ансамблевого агента; документы, использованные ансамблевым агентом. Значения колонки документов не пусты во всех случаях, все ответы получены от соответствующих агентов.

Листинг 6.4.2.5 – Шаблон результата, содержащего полный сгенерированный ответ агента, полученного с сервиса API компоненты модуля 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": <частичный результат генерации ответа на запрос>

}

## Проверка возможности генерации сложных синтетических примеров, пригодных для дообучения БЯМ

Маша:

Способ испытаний: Описание примера из папки Examples и Руководства программиста с описанием того как оценить наличие функций ниже

Д3.2.6 Фреймворк должен предоставлять инструменты и шаблоны для генерации сложных синтетических примеров, пригодных для дообучения БЯМ.

## Комплексная проверка комплектности программных модулей и программной документации

Аня:

Д3.1.1  Фреймворк состоит из следующих программных 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 синтетических примеров для дообучения БЯМ.

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

Д4.1 На Фреймворк должна быть разработана следующая программная документация:

-               спецификация;

-               текст программы;

-               описание программы;

 Д4.2 На Фреймворк должно быть разработано руководство программиста.

Цель испытания: проверка наличия всех компонентов фреймворка Proto.LLM и программно-эксплуатационной документации, а также ее соответствие требованиям ЕСПД.

Способ испытания:

1. Открыть репозиторий Proto.LLM по ссылке <https://github.com/aimclub/ProtoLLM/tree/main>.

2. Проверить содержимое каталогов на наличие компонентов, предусмотренных техническим заданием.

3. Проверить содержимое каталога <https://github.com/aimclub/ProtoLLM/tree/main/docs> на наличие документов, предусмотренных техническим заданием.

Критерии успеха:

1) Компонент упрощения и ускорения прототипирования систем с БЯМ на основе RAG считается присутствующим, если репозиторий <https://github.com/aimclub/ProtoLLM/tree/main> содержит каталог <https://github.com/aimclub/ProtoLLM/tree/main/protollm/rags> с подкаталогами ../configs, …/pipeline, …/rag\_core, …/settings, …/stores, в которых содержатся файлы с кодом и конфигурациями.

2) Компонент упрощения и ускорения прототипирования систем с БЯМ с возможностью автоматизированного подключения внешних сервисов, приложений, моделей в виде плагинов считается присутствующим, если репозиторий <https://github.com/aimclub/ProtoLLM/tree/main> содержит каталог <https://github.com/aimclub/ProtoLLM/tree/main/protollm/agents> с подкаталогом [llama31\_agents](https://github.com/aimclub/ProtoLLM/tree/main/protollm/agents/llama31_agents).

3) Компонент повышения эффективности БЯМ за счет использования ансамблей моделей и мультиагентных технологий считается присутствующим, если репозиторий <https://github.com/aimclub/ProtoLLM/tree/main> содержит каталог https://github.com/aimclub/ProtoLLM/tree/main/protollm/ensembles\_ma.

4) Компонент создания синтетических примеров для дообучения БЯМ считается присутствующим, если репозиторий <https://github.com/aimclub/ProtoLLM/tree/main> содержит каталог <https://github.com/aimclub/ProtoLLM/tree/main/protollm/synthetics>.

5) Вспомогательные компоненты для разработки серверной части приложений на основе БЯМ считается присутствующим, если репозиторий https://github.com/aimclub/ProtoLLM/tree/main содержит каталог <https://github.com/aimclub/ProtoLLM/tree/main/protollm_tools> с подкаталогами [llm-api](https://github.com/aimclub/ProtoLLM/tree/main/protollm_tools/llm-api), [llm-worker](https://github.com/aimclub/ProtoLLM/tree/main/protollm_tools/llm-worker), [sdk](https://github.com/aimclub/ProtoLLM/tree/main/protollm_tools/sdk).

6) Проверка на наличие документов считается пройденной, если каталог <https://github.com/aimclub/ProtoLLM/tree/main/docs> содержит файлы спецификации, текста программы, описания программы, руководства программиста.

## Комплексная проверка наличия шаблонов для типовых систем с RAG и агентами

Коля:

Д3.2.10 Фреймворк должен предоставлять готовые шаблоны для типовых систем с RAG и агентами, которые можно использовать как отправную точку для разработки.

Д5.1 Возможность разворачивания прототипов приложений на основе не менее 10 шаблонов, включая шаблоны для RAG, шаблоны для агентов, шаблоны для управления подключением плагинов.

Д5.2. Возможность создания сложных синтетических данных на основе не менее чем 5 шаблонов.

Проверка направлена на оценку наличия типовых шаблонов для типовых систем с RAG и агентами для создания производных систем на основе фреймворка ProtoLLM.

**Способ проверки:**

1. Перейти в репозиторий <https://github.com/aimclub/ProtoLLM.>

2. Перейти к каталог templates

3. Перейти под каталог prompt\_templates, убедиться в наличии py-файлов в которых расположены шаблоны промптов для типовых сценариев использования (БЯМ) - промпты для RAG, промпты для вопросно-ответных систем и систем-ассистентов, промты для вычисления метрик при проведение численных экспериментов.

4. Перейти под каталог code\_templates, убедиться в наличии py-файлов с шаблонами для сборки типовых систем (агентной системы, системы с вызовом функций, системы с использованием RAG).

5. Открыть файл, убедиться в наличии шаблонов для генерации синтетических данных - protollm/templates/prompt\_templates/synthetic\_data\_prompts.py

**Критерий успеха:**

Проверка считается успешной, если в каталогах присутствуют не менее 10 примеров с промптами и кодовыми шаблонами и не менее 5 шаблонов для синтетических данных.

## Проверка наличия обработки сбоев и исключений

Коля:

Д3.3.2 Должна быть предусмотрена обработка сбоев и исключений при работе алгоритмов, сопровождаемая однозначно интерпретируемыми сообщениями оператору

В данном пункте осуществляется реализации обработки сбоев и исключений при работе алгоритмов фреймворка ProtoLLM.

**Способ проверки:**

1. Для запуска проверки необходимо выполнить код, приведенный в листинге 6.8.1.

Листинг 6.8.1 – Скрипт запуска эксперимента по вводу ошибочных данных в модуль работы с агентами фреймворка ProtoLLM

|  |
| --- |
| from protollm.agents.llama31\_agents.llama31\_agent import Llama31ChatModel  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  # Define tools using the @tool decorator  @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"],  )  # Create the ChatPromptTemplate  prompt = ChatPromptTemplate.from\_messages(  [  system\_message,  MessagesPlaceholder(variable\_name="chat\_history", optional=True),  human\_message,  ]  )  # Initialize the custom LLM  llm = Llama31ChatModel(  api\_key="WRONG\_API\_KEY",  base\_url=" WRONG\_BASE\_URL ",  model="WRONG\_MODEL\_NAME",  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"],  )  # Example usage of the agent  if \_\_name\_\_ == "\_\_main\_\_":  # Question for the agent  user\_question = "What is the sum and product of 15 and 27?"  # Use invoke instead of run  response = agent\_executor.invoke({"input": user\_question})  # Access the output  final\_answer = response["output"]  # Print the final answer  print(f"Agent's Response: \n {final\_answer}") |

**Критерий успеха:**

Проверка считается успешной, если в консоли выводится ошибка, начинающая с «API request failed» и далее детализирующая причины ошибки.

## Проверка наличия сведений о порядке обращения к разработчикам при выявлении сбоев и ошибок при работе библиотеки

Коля

Д3.3.3 На сервисе веб-хостинга кода Фреймворка должны быть приведены сведения о порядке обращения к разработчикам при выявлении сбоев и ошибок при работе библиотеки.

В данном пункте осуществляется проверка наличия сведений о порядке обращения к разработчика в репозитории фреймворка ProtoLLM.

**Способ проверки:**

1. Перейти в репозиторий <https://github.com/aimclub/ProtoLLM>

2. Открыть файл README.md

3. Перейти к секции Contacts, проверить наличие контактов

**Критерий успеха:**

Проверка считается успешной, если секции указаны контакты разработчиков.

## Проверка наличия возможности предоставления API для программного управления всеми основными функциями системы

Юра

Способ испытаний: Описание как вызываются эндпойнты (<https://github.com/aimclub/ProtoLLM/blob/main/protollm_tools/llm-api/protollm_api/backend/endpoints.py>)

Д3.2.9 Фреймворк должен предоставлять API для программного управления всеми основными функциями системы.

Фреймворк предоставляет FastAPI интерфейс для взаимодействия со всеми компонентами системы.

Листинг 6.10.1 отражает логику построения эндпоинтов на основе FastAPI.

Листинг 6.10.1 – Интерфейс взаимодействия с эндпоинтами фреймворка

|  |
| --- |
| import logging  from fastapi import APIRouter  from protollm\_api.backend.broker import send\_task, get\_result  from protollm\_api.config import Config  from protollm\_sdk.models.job\_context\_models import (  PromptModel, ResponseModel, ChatCompletionModel,  PromptTransactionModel, ChatCompletionTransactionModel,  PromptTypes  )  from protollm\_sdk.object\_interface.redis\_wrapper import RedisWrapper  logging.basicConfig(level=logging.INFO)  logger = logging.getLogger(\_\_name\_\_)  def get\_router(config: Config) -> APIRouter:  router = APIRouter(  prefix="",  tags=["root"],  responses={404: {"description": "Not found"}},  )  redis\_db = RedisWrapper(config.redis\_host, config.redis\_port)  @router.post('/generate', response\_model=ResponseModel)  async def generate(prompt\_data: PromptModel, queue\_name: str = config.queue\_name):  transaction\_model = ChatCompletionTransactionModel(  prompt=ChatCompletionModel.from\_prompt\_model(prompt\_data),  prompt\_type=PromptTypes.CHAT\_COMPLETION.value  )  await send\_task(config, queue\_name, transaction\_model)  logger.info(f"Task {prompt\_data.job\_id} was sent to LLM.")  return await get\_result(config, prompt\_data.job\_id, redis\_db)  @router.post('/chat\_completion', response\_model=ResponseModel)  async def chat\_completion(prompt\_data: ChatCompletionModel, queue\_name: str = config.queue\_name):  transaction\_model = ChatCompletionTransactionModel(  prompt=prompt\_data,  prompt\_type=PromptTypes.CHAT\_COMPLETION.value  )  await send\_task(config, queue\_name, transaction\_model)  logger.info(f"Task {prompt\_data.job\_id} was sent to LLM.")  return await get\_result(config, prompt\_data.job\_id, redis\_db)  return router |

Заспуск API может быть осуществлен с помощью кода, представленного в листинге 6.10.2.

Листинг 6.10.2 – Интерфейс взаимодействия с эндпоинтами фреймворка

|  |
| --- |
| from fastapi import FastAPI  from protollm\_api.config import Config  from protollm\_api.backend.endpoints import get\_router  app = FastAPI()  config = Config.read\_from\_env()  app.include\_router(get\_router(config)) |

При успешном запуске в терминале появится соответствующее сообщение.

## Проверка наличия поддержки механизмов для обеспечения воспроизводимости экспериментов с различными конфигурациями БЯМ и плагинов

Юра

Д3.2.11 Фреймворк должен поддерживать механизмы для обеспечения воспроизводимости экспериментов с различными конфигурациями БЯМ и плагинов.

С помощью фреймворка ProtoLLM создается сценарий взаимодействия БЯМ с плагинами системы.

Листинг 6.11.1 – Пример создания сценария агента с помощью ProtoLLM

|  |
| --- |
| from langchain.agents import (  create\_structured\_chat\_agent,  AgentExecutor,  )  from langchain.prompts import ChatPromptTemplate, MessagesPlaceholder  from langchain\_core.prompts import (  SystemMessagePromptTemplate,  HumanMessagePromptTemplate,  )  from stairs\_sdk.jobs.job import Job  from stairs\_sdk.jobs.job\_context import JobContext  from stairs\_llm\_agent.models import StairsLLMAgentResult  from stairs\_llm\_agent.config import CUSTOM\_USER\_MESSAGE, CUSTOM\_SYSTEM\_MESSAGE  from stairs\_llm\_agent.parse\_result import parse\_intermediate\_steps  from stairs\_llm\_agent.tools import (  query\_database\_rag,  get\_time,  get\_resource,  restore\_works\_edges,  start\_schedule,  extract\_scheduling\_params  )  from protollm.agents.llama31\_agents.llama31\_agent import Llama31ChatModel  class StairsLLMAgentJob(Job):  """  Job class representing the main execution of the Stairs LLM Agent.  Attributes:  tools (list): List of tools available to the agent.  prompt (ChatPromptTemplate): Chat prompt template used to structure the conversation.  llm (Llama31ChatModel): Language model instance used by the agent for generating responses.  """  def \_\_init\_\_(self):  """Initializes the StairsLLMAgentJob with tools, prompt, and language model."""  super().\_\_init\_\_()  self.tools = [  query\_database\_rag,  get\_time, get\_resource,  restore\_works\_edges,  start\_schedule,  extract\_scheduling\_params  ]  self.prompt = ChatPromptTemplate.from\_messages(  [  SystemMessagePromptTemplate.from\_template(  CUSTOM\_SYSTEM\_MESSAGE,  input\_variables=["tools", "tool\_names"],  ),  MessagesPlaceholder(variable\_name="chat\_history", optional=True),  HumanMessagePromptTemplate.from\_template(  CUSTOM\_USER\_MESSAGE,  input\_variables=["input", "agent\_scratchpad"],  ),  ]  )  self.llm = Llama31ChatModel()  def run(self, job\_id: str, ctx: JobContext, \*\*kwargs):  """Executes the agent job, generating and saving the result.  Args:  job\_id (str): Unique identifier for the job execution.  ctx (JobContext): Context object for managing the job's lifecycle and data storage.  \*\*kwargs: Additional parameters which includes:  request (str): The user query string.  is\_project (bool): Flag indicating if the request is a project.  is\_scheduling (bool): Flag indicating if the request is for scheduling.  Returns:  None. Saves the job result directly into the provided job context.  """  request: str = kwargs.get("request", "")  is\_project: bool or None = kwargs.get("is\_project", None)  request += "" if is\_project is None else f" is\_project={is\_project}"  is\_scheduling: bool or None = kwargs.get("is\_scheduling", None)  request += "" if is\_scheduling is None else f" is\_scheduling={is\_scheduling}"  agent\_request = {"input": request} # , "context": {"is\_project": is\_project, "is\_scheduling": is\_scheduling}}  agent = create\_structured\_chat\_agent(  llm=self.llm,  tools=self.tools,  prompt=self.prompt,  stop\_sequence=True,  )  agent\_executor = AgentExecutor.from\_agent\_and\_tools(  agent=agent,  tools=self.tools,  verbose=True,  return\_intermediate\_steps=True,  output\_keys=["output"],  )  result = agent\_executor.invoke(agent\_request)  parsed\_result = parse\_intermediate\_steps(result)  dumped\_result = parsed\_result.model\_dump()  ctx.result\_storage.save\_dict(job\_id, dumped\_result)  print("written to redis: ", parsed\_result) |

При запуске эксперимента, результаты работы БЯМ с инструментами записываются в базу данных.

Листинг 6.11.2 – Пример записи результата работы БЯМ в базу данных

|  |
| --- |
| import uuid  from stairs\_sdk.jobs.utility import construct\_job\_context  from stairs\_sdk.utils.reddis import get\_reddis\_wrapper, load\_result  from stairs\_llm\_agent.jobs import StairsLLMAgentJob  def prepare\_test\_data(request: str, is\_project: str or None, is\_scheduling: str or None, tools: list[str]) -> dict:  return {  "request": request,  "is\_project": is\_project,  "is\_scheduling": is\_scheduling,  "tools": tools  }  SCHEDULE\_QUERY = "Запусти планирования для проекта"  SCHEDULE\_TEST = prepare\_test\_data(SCHEDULE\_QUERY, True, False, ["start\_schedule"])  def main():  job\_id = str(uuid.uuid4())  ctx = construct\_job\_context("agent")  job = StairsLLMAgentJob()  request, is\_project, is\_scheduling, \_ = SCHEDULE\_TEST  job.run(job\_id=job\_id, ctx=ctx, request=request, is\_project=True, is\_scheduling=False)  rd = get\_reddis\_wrapper()  result = str(load\_result(rd, job\_id, "agent"))  if \_\_name\_\_ == "\_\_main\_\_":  main() |

При успешном запуске в терминале будет пошагово отображаться история действий агента.

## Комплексная проверка быстродействия и точности конфигураций системы на основе БЯМ

Аня+Коля+Лиза Луценко (я добавлю описание из статьи, Коля или Лиза добавят подходящие кусочки кода из репо к статье):

Д5.3 Среднее время отклика прототипа приложения на основе БЯМ: <10 с для запросов к системе на основе БЯМ, < 20 с  для запросов к системе на основе БЯМ с RAG и агентной реализацией.

Д5.4 Увеличение средней точности ответов прототипов RAG систем на 10% по сравнению с решением, основывающемся только на БЯМ (без использования RAG).

Д5.6. Улучшение точности ответов не менее чем на 30% при одновременном использовании агентов на основе БЯМ с плагинами и RAG  по сравнению с одиночной моделью.

Д5.7. Улучшение точности ответов не менее чем на 15% при использовании агентов на основе БЯМ с плагинами по сравнению с одиночной моделью.

Цель проверки: оценка быстродействия и точности системы на основе БЯМ, протипированной с помощью Proto.LLM.

Способ проверки:

1. Открыть репозиторий <https://github.com/ITMO-NSS-team/llm-agents-for-smartcities-paper>.

2. Запустить код из ноутбука <https://github.com/ITMO-NSS-team/llm-agents-for-smartcities-paper/blob/main/pipelines/tests/tests_for_experiments.py>.

3. Открыть файлы с расчетными метриками в каталоге …/pipelines/tests/test\_results.

4.

## Проверка наличия тестов

Коля

3.3.1 Все программные изделия должны быть снабжены модульными и интеграционными тестами, позволяющими проверить их работоспособность.

В данном пункте осуществляется проверка наличия модульных и интеграционных тестов, позволяющими проверить их работоспособность в репозитории фреймворка ProtoLLM.

**Способ проверки:**

1. Перейти в репозиторий <https://github.com/aimclub/ProtoLLM>.

2. Последовательно перейти в каталоги tests, protollm\_tools/llm-api/tests, protollm\_tools/sdk/tests.

3. Перейти на страницу <https://github.com/aimclub/ProtoLLM/actions/workflows/unit-build.yml>, убедиться в корректном статусе выполнения модульных тестов (зеленая отметка).

**Критерий успеха:**

Проверка считается успешной, если в каталогах представлены модульные и интеграционные тесты и отчет об их прохождении.

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 |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |