МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Курс “Парадигмы и конструкции языков программирования”

Отчет по ДЗ
“Deep Research Agent”

Выполнил:
Студент группы ИУ5-32Б
Ковригина Д. М.
Преподаватель:
Гапанюк Ю.Е.

Москва 2025

Deep Research Agent — это многоагентная система для автоматизированного исследования и анализа информации из различных источников с использованием языковых моделей. Система построена на фреймворке LangGraph и использует GigaChat в качестве основной языковой модели.

Архитектура системы

Основные компоненты:

1. main.py - Главный интерфейс пользователя

Назначение: Точка входа для запуска системы, предоставляет интерактивный интерфейс для взаимодействия с агентом.

Ключевые функции:

- Интерактивная сессия исследования с поддержкой уточнений
- Быстрое тестирование без уточнений
- Проверка подключений к API (GigaChat, Tavily)
- Отображение мысленного процесса агента в реальном времени
- Меню выбора режимов работы

Особенности:

- Поддержка нескольких итераций уточнений (до 5)
- Детальное логирование мыслей агента
- Гибкая конфигурация через файл `.env`

2. deep_researcher.py - Ядро агента (LangGraph граф)

Назначение: Содержит основную логику агента, построенную как граф состояний LangGraph.

Архитектура графа:

Пользователь → Clarify → Research Brief → Supervisor → Researchers → Final Report

Ключевые узлы графа:

1. **clarify_with_user** - Анализ запроса и уточнение деталей
2. **write_research_brief** - Формирование детального исследовательского брифа
3. **research_supervisor** (подграф) - Управление исследовательским процессом

    - supervisor - Планирование и делегирование задач
    - supervisor_tools - Выполнение инструментов руководителя
4. **Исследовательские подграфы** - Параллельные исследования
    - researcher - Индивидуальные исследователи
    - researcher_tools - Использование инструментов поиска
    - compress_research - Сжатие результатов
5. **final_report_generation** - Генерация финального отчета

Особенности:

- Многоуровневая архитектура с супервизором и исследователями
- Параллельное выполнение исследовательских задач
- Механизм обработки ограничений токенов
- Система логирования мыслей агента

3. state.py - Определения состояний и структур данных

Назначение: Содержит Pydantic модели для типизированных состояний графа.

Ключевые компоненты:

Структурированные выходы (Structured Outputs):

- ConductResearch - Делегирование исследовательских задач
- ResearchComplete - Сигнал завершения исследования
- Summary - Структура для хранения сводок
- ClarifyWithUser - Модель для уточнений у пользователя
- ResearchQuestiom - Исследовательский вопрос и бриф

Определения состояний:

- AgentInputState - Входное состояние (только сообщения)
- AgentState - Основное состояние агента
- SupervisorState - Состояние супервизора исследований
- ResearchState - Состояние индивидуального исследователя
- ReasearcherOutputState - Выходное состояние исследователя

4. utils.py - Вспомогательные функции и инструменты

Назначение: Содержит утилиты, инструменты поиска и логирование.

Ключевые модули:

Инструменты поиска:

- tavily_search() - Асинхронный поиск через Tavily API
- tavily_serach_async() - Параллельное выполнение поисковых запросов
- summarize_webpage() - Суммаризация контента веб-страниц
- think_tool() - Инструмент стратегического размышления

Логирование:

- ThoughtLogger - Логгер для мыслей агента
    - Логирование размышлений, делегирования, поиска
    - Временные метки и контекст
    - Консольный вывод в реальном времени

Утилиты моделей:

- get_api_key_for_model() - Получение API ключей
- is_token_limit_exceeded() - Обработка ограничений токенов
- get_model_token_limit() - Получение лимитов моделей

Утилиты обработки:

- remove_up_to_last_ai_message() - Усечение истории сообщений
- get_today() - Форматирование даты

5. configuration.py - Управление конфигурацией

Назначение: Централизованное управление настройками системы.

Ключевые настройки:

Перечисления:

- SearchAPI - Доступные API поиска (Tavily, GigaChat, None)

Конфигурационные классы:

- Configuration - Основной класс конфигурации

6. prompts.py (не показан, но упомянут) - Промпты системы

Назначение: Содержит шаблоны промптов для различных этапов исследования.

Рабочий процесс системы

Этап 1: Уточнение запроса

1. Пользователь задает вопрос
2. Агент анализирует, нужны ли уточнения
3. Если нужно - задает уточняющие вопросы
4. Если нет - начинает исследование

Этап 2: Планирование исследования

1. Преобразование запроса в детальный бриф
2. Инициализация супервизора исследований
3. Создание стратегии исследования

Этап 3: Выполнение исследований

1. Супервизор разбивает задачу на подзадачи
2. Делегирует задачи исследователям
3. Исследователи параллельно:
    - Выполняют поисковые запросы
    - Анализируют результаты
    - Используют rhink_tool() для размышлений
    - Собирают информацию

Этап 4: Обработка результатов

1. Сжатие и структурирование данных
2. Агрегация результатов от всех исследователей
3. Проверка полноты информации

Этап 5: Генерация отчета

1. Создание финального отчета
2. Обработка ограничений токенов
3. Форматирование вывода

Ключевые особенности

1. Многоагентная архитектура

- Супервизор для координации
- Множество исследователей для параллельной работы
- Специализированные агенты для разных задач

2. Обработка ограничений

- Механизмы для работы с ограничениями токенов
- Автоматическое усечение контента
- Повторные попытки при ошибках

3. Логирование и отладка

- Детальное логирование мыслей агента
- Временные метки для анализа
- Консольный вывод в реальном времени

4. Гибкая конфигурация

- Поддержка разных API поиска
- Настройка моделей для разных задач
- Контроль параллелизма и итераций

5. Интерактивность

- Поддержка уточняющих вопросов
- Возможность прерывания
- Пошаговое выполнение

Интеграции

Поддерживаемые API:

1. **GigaChat** - Основная языковая модель
    - Модель: GigaChat-2-Max
    - Контекстное окно: 32768 токенов
    - Специальная интеграция через langchain_gigachat
2. **Tavily** - Поисковый API
    - Оптимизированный для исследований
    - Поддержка raw content
    - Тематическая фильтрация