

# Лабораторная работа 2: Проектирование и реализация мультиагентной системы с использованием LangChain и LangGraph

## 1. Цель лабораторной работы

В этой лабораторной работе вам предстоит спроектировать и реализовать небольшую **мультиагентную систему (МАС)** поверх LLM с использованием **LangChain** и **LangGraph**.

Ваша система должна:

- содержать несколько агентов с различными ролями;
- реализовывать как минимум один паттерн МАС из лекций (router, planner–executor, supervisor, последовательный workflow и т.п.);
- демонстрировать базовую логику **передачи управления (handoff)** между агентами;
- использовать **tool calling** (вызов инструментов) и хотя бы минимальное **управление памятью** (memory management).

Система должна помогать решать **реалистичные задачи**, которые важны лично вам (учёба, программирование, планирование и т.п.), а не быть просто игрушечным примером.

---

## 2. Формат сдачи

Предпочтительный формат — **репозиторий** (GitHub / GitLab или архив с папкой), в котором:

- понятная структура (например, `src/`, `notebooks/`, `README.md`, `requirements.txt` или `pyproject.toml`);
- есть одна основная точка входа, показывающая, как запустить систему (скрипт или ноутбук);
- есть короткий `README` с описанием идеи, инструкцией по установке зависимостей и запуску демонстрации.

Если репозиторий по каким-то причинам неудобен, допускается **хорошо структурированный Jupyter-ноутбук**, который одновременно содержит код и

пояснения.

---

### 3. Технические требования и использование модели

Вы должны использовать ту же модель Qwen и тот же сервер vLLM, что и в Лабораторной работе 1:

- **Модель:** Qwen, запущенная через ваш локальный или удалённый сервер vLLM с OpenAI-совместимым API.
- **Доступ:**  
Используйте те же значения `OPENAI_API_BASE`, `OPENAI_API_KEY` (или `"EMPTY"`, если это подходит вашей конфигурации) и имени модели, что и в ЛР 1.
- **Библиотеки:**
  - `langchain-openai` и другие компоненты LangChain,
  - `Langgraph` для построения графа,
  - любые дополнительные небольшие утилиты по необходимости.

Не переходите в этой работе на другой хостинг или другую модель: цель — **переиспользовать и углубить** вашу настройку из ЛР 1, добавив к ней мультиагентную оркестрацию, tool calling и управление памятью.

---

### 4. Сценарий: «Мультиагентный помощник по учёбе и продуктивности»

Базовая идея (её можно модифицировать и расширять):

Вы создаёте **мультиагентного помощника**, который помогает с задачами, связанными с учёбой, программированием и/или повседневной продуктивностью. Помощник принимает запрос пользователя, передаёт его подходящим агентам, при необходимости использует **инструменты (tool calls)** (например, простые Python-функции, калькулятор, небольшой «локальный справочник») и поддерживает некоторое **состояние памяти** между шагами или взаимодействиями (например, запоминает предыдущие вопросы в сессии или краткий профиль пользователя).

Необходимо реализовать **как минимум трёх различных агентов** (с разными ролями и логикой, как в коде, так и в графе), например:

- **Агент-роутер / анализатор**, который классифицирует запрос и решает, какие агенты должны быть задействованы.

- Один или несколько **предметно-ориентированных агентов** (например, агент теории, агент проектирования архитектуры, агент-помощник по коду, агент-планировщик).
- Опционально — агент, отвечающий за **работу с памятью** (чтение/обновление истории) или за **вызов инструментов** (tool-агент).

Агенты могут использовать одну и ту же модель Qwen, но должны иметь **разные промпты, ответственность и поведение**.

---

## 5. Требования и задания

### Задание 1 – Проектирование архитектуры

В репозитории (или в ноутбуке) создайте **короткий архитектурный документ** (Markdown-файл или раздел в markdown), в котором вы опишете:

- Какие агенты в системе существуют и за что каждый из них отвечает.
- Какие паттерны MAC из лекции вы реализуете (router + специализированные агенты, planner-executor, supervisor и т.п.) и как это отражено в вашем дизайне.
- Небольшую диаграмму (Mermaid, ASCII или картинка), показывающую поток управления и данных: от входного запроса пользователя до перераспределения по агентам и возврата ответа; где именно происходит передача управления (handoff).

Отдельно опишите, **где и как используется tool calling** (какой агент вызывает какие инструменты и зачем) и **как организовано управление памятью** (что хранится, где хранится, как это влияет на последующие шаги).

---

### Задание 2 – Реализация в LangGraph + LangChain (с использованием Qwen через vLLM)

Реализуйте систему как граф **LangGraph**, использующий вашу модель Qwen из ЛР 1:

- Определите общий **state** (например, с помощью `TypedDict`), в котором минимум есть:
  - входной запрос пользователя (`question` / `query`),
  - промежуточные поля (классификация, план, частичные ответы),
  - финальный ответ,

- поля, связанные с памятью (история сессии, краткие заметки о пользователе и т.п.).
- Реализуйте несколько **узлов графа**, соответствующих вашим агентам. Каждый узел:
  - читает релевантные части состояния;
  - вызывает **модель Qwen через LangChain** (так же как и в ЛР 1);
  - при необходимости вызывает **инструмент** (Python-функцию, мини-БД, простой API и т.п.);
  - обновляет состояние результатами своей работы.
- Реализуйте **передачу управления (handoff)** через рёбра графа или условную маршрутизацию так, чтобы путь по графу зависел от типа запроса или промежуточного состояния.
- Реализуйте хотя бы минимальную форму **управления памятью**, например:
  - хранение в state списка предыдущих вопросов/ответов в рамках одной сессии;
  - мини-хранилище истории в оперативной памяти или простом файле;
  - небольшой RAG-подобный компонент, где один из агентов извлекает заметки из списка/файла.

Сделайте простой скрипт или ячейку в ноутбуке, которая показывает, как пользователь может вызвать ваш граф с запросом, получить финальный ответ и посмотреть, какие агенты были задействованы.

---

### Задание 3 – Эксперименты и неформальная оценка

Запустите вашу систему на **нескольких запросах** (минимум пяти), среди которых:

- один концептуальный/теоретический вопрос про MAC или LLM-агентов;
- один вопрос по проектированию/архитектуре;
- один вопрос по реализации/программированию;
- ещё несколько запросов, связанных с **повседневными задачами**, для которых предназначен ваш помощник.

Для каждого запроса зафиксируйте:

- сам текст запроса;

- какие агенты/узлы были активированы и в каком порядке;
- где были задействованы инструменты и как использовалась память (например: «router → planner → code\_helper; память обновлена новым планом; вызван инструмент для расчёта дат»);
- короткий комментарий о том, оказался ли ответ полезным, и что в нём хотелось бы улучшить.

В конце сделайте небольшой раздел с «оценкой», в котором опишите несколько неформальных критериев (например, корректность маршрутизации, субъективная полезность, использовалась ли память, были ли вызовы инструментов осмысленными) и проанализируйте, насколько хорошо система этим критериям соответствует.

---

## Задание 4 – Рефлексия

Напишите краткое размышление о проделанной работе:

- Что в вашем дизайне MAC, логике handoff, использовании инструментов и управлении памятью сработало хорошо?
  - В каких местах система вела себя неожиданно или ошибалась (например, неверный выбор агента, плохое использование памяти, бессмысленные tool-calls)?
  - Как бы вы продолжили развивать систему, если бы было больше времени (например, добавили бы ревьюера, более глубокую память, новые инструменты, более сложные паттерны взаимодействия между агентами)?
- 

## 6. Краткий обзор возможных для реализации идей

Выбирая конкретный сценарий, подумайте о ситуациях, где одного «общего» чат-бота уже недостаточно, а небольшой «команды» специализированных помощников было бы гораздо удобнее. Это может быть учебный помощник, который помнит, какие темы по этому курсу вы уже обсуждали, предлагает, что повторить дальше, и через инструменты генерирует небольшие квизы или чек-листы. Или помощник по программированию, который учитывает контекст текущего проекта, объясняет ошибки с учётом уже обсуждённого кода и через инструменты умеет просматривать или преобразовывать фрагменты. Другой вариант — планировщик, который превращает размытые цели в конкретный пошаговый план, использует небольшие функции для работы с датами и временем и сохраняет краткую историю выполненных задач, чтобы со временем подстраивать свои рекомендации.

Постарайтесь выбрать такой сценарий, которым вы действительно могли бы пользоваться в течение семестра. Цель этой работы — не создать максимально большую и сложную систему, а получить небольшой, но целостный мультиагентный пример, который демонстрирует паттерны MAC, осмысленный tool calling и работу с памятью и при этом реально помогает в какой-то части вашей повседневной деятельности.