0. Цель лабораторной работы

В этой лабораторной работе вы разработаете и реализуете **небольшой, но надёжный workflow с LLM, используя LangGraph**.

Ваш workflow должен:

- Следовать паттерну **ReAct** (LLM рассуждает, решает, какой инструмент вызвать, вызывает его, анализирует результат и выбирает следующий шаг).

- Использовать **инструменты** (API-вызовы, локальные функции и т.д.).

- Использовать **структурированный вывод** (модели Pydantic, чтобы LLM возвращал машиночитаемые данные).

- Поддерживать логику **повторных попыток (retry),** когда шаг завершается ошибкой.

- Поддерживать **параллельное** выполнение шагов, где это уместно.

- Использовать **разделение ролей**: разные шаги пайплайна вызывают LLM с разными системными промптами (planner / writer / reviewer и т.п.).

Придумайте: «граф из 2–4 чётко определённых узлов», где каждый узел — это один вызов LLM с определённой ролью.

К концу работы вы должны уметь:

- Визуализировать граф вашего пайплайна (узлы, связи)[У LangGraph есть методы для визуализации].

- Объяснить, какие узлы можно выполнять параллельно.

- Продемонстрировать сквозной запуск на реальном примере запроса.

1. Технические требования (обязательные)

Фреймворк: langchain + langgraph

Контракты данных:

Используйте модели Pydantic для определения:

- входов инструментов

- выходов инструментов
- ответов LLM (например, LiteratureSummary, ProductReviewSummary, HypothesisStatement и т.д.)

LLM должен быть проинструктирован отвечать строго в этой схеме.

Повторные попытки:

- Как минимум один узел графа (например, вызов внешнего API вроде arXiv) должен иметь логику повторной попытки при ошибке.

Ролевое разделение:

- Каждая логическая подзадача — это отдельный вызов LLM с отдельным системным промптом.

Например: PlannerNode и WriterNode — два разных вызова LLM с разными инструкциями. Можно вызывать их последовательно.

Параллелизм:

- Если два вызова инструментов не зависят друг от друга (например, «получить отзывы из источника A» и «получить геоданные из источника B»), их необходимо реализовать как параллельные ветки в LangGraph, которые затем объединяются.

ReAct-цикл:

Как минимум один узел LLM должен:

- Читать текущее состояние,
- Решать, какой инструмент вызвать дальше,
- Вызывать его, затем продолжать рассуждение с учётом результата.

2. Вероятная структура вашего рабочего процесса

Ваш граф обычно имеет следующую структуру:

- Узел | Роль | Что делает

- Planner Node | LLM (аналитик/планировщик) | Получает сырой запрос пользователя и создаёт структурированный план (ключевые шаги, необходимые инструменты, фильтры и т.д.). Возвращает Pydantic модель.
- Tool Execution Nodes | Инструменты | Выполняют реальные действия (API, функции). Некоторые могут работать параллельно.
- Writer Node | LLM (писатель) | Синтезирует результаты в финальный, понятный пользователю ответ.
- (Опционально) Reviewer Node | LLM (редактор) | Улучшает черновик и указывает замечания в форме модели.

3. Примеры проектов

A. Ассистент литературного обзора

Запрос пользователя: «Дай краткий обзор последних работ по теме <topic>».

Planner Node → извлекает ключевые слова, фильтр по году, нужно ли анализировать авторов.

Инструменты (параллельно):

- search_arxiv(...) → список недавних статей
- author_stats(...) → показатели авторов (если нужно)

Writer Node → формирует краткий обзор: тренды, важные работы, открытые вопросы.

Retry: повтор попытки запроса arXiv при ошибке.

B. Поиск лучших сервисов поблизости

Запрос: «Найди лучший вариант рядом для <услуга/товар>».

Planner Node формирует LocalSearchPlan:

- target_item: что ищем
- min_rating

- max_distance_m
- verify_offering: нужно ли подтверждение предоставления услуги

Инструменты:

- search_candidates(...) → список мест
- get_offering_details(...)
- get_reviews(...)

Для каждого кандидата параллельно:

- OfferingCheckNode
- ReviewCheckNode

Writer Node → выбирает лучшие 2–3 варианта и объясняет почему.

Retry: при ошибке запросы повторяются.

C. Генератор гипотез

Запрос: «Тема: <topic>. Помоги сформировать исследовательскую гипотезу и эксперимент.»

Planner Node → определяет, нужны ли тренды / противоречия / эксперимент.

Узлы:

- TrendExtractorNode (LLM)
- ContradictionFinderNode (LLM)
- TrizHypothesisNode (LLM) → формирует гипотезу
- FeasibilityNode (LLM) → проект эксперимента

Writer Node → формирует финальный пакет.

Retry: повтор для длинной генерации гипотез.

4. Что нужно сдать

- Диаграмму графа
- Все модели Pydantic
- Код LangGraph:
    - узлы
    - связи
    - параллельные ветки
    - retry
- Один демонстрационный запуск (запрос → итоговый ответ)

5. Оценивание

Будет оцениваться:

- Использование LangGraph (а не просто один llm.invoke())
- Чёткий поток Planner → Tools → Writer
- ReAct-логика
- Везде Pydantic
- Реализованный retry
- Понятный и полезный итоговый вывод