3.6. “Мозг” для LLM: агенты (ReAct, LangGraph и SWE-задачи)

Зачем агенты?

Большие языковые модели способны генерировать текст и принимать решения, но сами по себе они не умеют *действовать* — взаимодействовать с файлами, API или внешними сервисами. **Агент** объединяет язык (LLM) и инструменты в единую систему: модель получает задания, формирует план, выбирает подходящий инструмент, передаёт ему входные данные, анализирует результаты и повторяет цикл, пока не достигнет цели. Такой механизм часто реализуют с помощью схемы **ReAct** (“Reasoning $\mapsto$ Action $\mapsto$ Observation $\mapsto$ Answer”), где цепочка рассуждений и действий чередуется[1]. На практике это позволяет свести галлюцинации и увеличить точность, поскольку модель может обращаться к внешним данным и обновлять свой план на основе обратной связи[1].

Компоненты агента

- **LLM** — основа, которая генерирует мысли (*reasoning*) и решает, какой инструмент применить.
- **Инструменты** — функции, API или команды, которые агент может вызывать. Это могут быть поисковые запросы, вызовы базы данных, выполнение кода и т.д.
- **Промпт** — инструкции, описывающие задачу и правила работы.

В документации LangGraph отмечается, что агент состоит именно из этих трёх компонентов: **LLM**, набор **инструментов** и **промпт**[2]. Модель работает в цикле: выбирает инструмент, получает ответ (*observation*) и использует его при выборе следующего действия, пока не выполнит задачу[2].

ReAct: схема «мысль $\rightarrow$ действие $\rightarrow$ обратная связь»

Исследование **ReAct** (Yao и др.) показало, что совмещение цепочки рассуждений и действий даёт значительный выигрыш. В алгоритме ReAct модель генерирует объяснение, затем решает, какой инструмент вызвать; результат действия добавляется в контекст, и модель продолжает рассуждать. Авторы показали, что такая процедура снижает галлюцинации и ошибки; на задачах HotpotQA и Fever ReAct, взаимодействуя с простым API Wikipedia, лучше справляется с проверкой фактов, чем цепочка рассуждений без действий[1]. На интерактивных задачах (ALFWorld, WebShop) метод превзошёл обученные агенты *imitation* и *reinforcement learning*, увеличив абсолютный успех на 34 % и 10 % соответственно[1]. ReAct можно реализовать вручную, описав инструментальную петлю, либо воспользоваться библиотеками, которые берут на себя оркестрацию.

LangGraph и граф-оркестрация

LangGraph — низкоуровневый фреймворк для построения, управления и развёртывания долговременно работающих и *сохранённых* агентов. Документация отмечает, что LangGraph используется компаниями, разрабатывающими будущие агентные системы (Klarna, Replit, Elastic и др.), и предоставляет инфраструктуру для построения *stateful*-агентов, устойчивых к сбоям, с поддержкой стриминга, *human-in-the-loop* и других возможностей[3]. В отличие от высокоуровневых "агентов" LangChain, LangGraph ориентирован на оркестрацию и не абстрагирует промпты или архитектуры; это «framework + runtime» для циклов, где важны выполнение, память и контроль[3].

Ключевые преимущества LangGraph

- **Долговременное выполнение (Durable execution)** — позволяет агенту выдерживать сбои и работать в течение длительных периодов, продолжая с места остановки[3].
- **Human-in-the-loop** — в любой момент можно приостановить выполнение, проанализировать состояние и скорректировать поведение, что особенно важно для критичных задач[3].
- **Обширная память** — поддержка краткосрочной и долгосрочной памяти позволяет сохранять контекст между сессиями и строить сложные рассуждения[3].
- **Отладка и наблюдаемость** — интеграция с LangSmith обеспечивает визуализацию траекторий агента, состояния и метрик[3].
- **Готовность к продакшену** — инфраструктура LangGraph рассчитана на масштабирование долгоживущих рабочих процессов, интеграцию с LangChain и LangSmith, а также обеспечивает удобное развёртывание[33].

Экосистема

- **LangSmith** — платформа для оценки и наблюдаемости; помогает анализировать траектории агентов, отлаживать вызовы LLM и повышать качество[3].
- **LangChain** — предоставляет высокоуровневые абстракции для LLM-приложений и включает готовых агентов, построенных поверх LangGraph; используется, если нужно быстро решить стандартные задачи, не углубляясь в оркестрацию[3].

Agent development с LangGraph

Файл `langgraph/agents/overview.md` описывает, как строятся готовые агенты:

- Агент определяет **LLM**, набор **tools** и **prompt**[2].
- LangGraph предоставляет **память** (short-term и long-term), **human-in-the-loop** контроль, **streaming** (передача токенов и результатов в реальном времени), а также инструменты для развёртывания и отладки (LangGraph Platform, LangGraph Studio)[2].
- Пре-пост-`hooks` позволяют организовать обработку до и после вызова модели: например, сжать историю сообщений, внедрить guardrails или организовать запросы к человеку[2].

Эти компоненты помогают быстро сконструировать **ReAct-агента** и визуализировать его граф: LangGraph генерирует граф состояний, отображающий связи между инструментами и действиями, что облегчает анализ и расширение[2].

SWE-agent: LLM-инженер для GitHub

SWE-agent — исследовательский проект, созданный группой из Princeton и Stanford, который демонстрирует, как мощный LLM может быть объединён с набором инструментов для решения реальных инженерных задач. SWE-agent позволяет модели (GPT-4 or Claude 3 Sonnet) *самостоятельно использовать инструменты* для исправления ошибок в реальных GitHub-репозиториях, поиска уязвимостей или выполнения любых кастомных задач[4]. Авторы подчёркивают, что агент оставляет максимальную свободу LLM: он работает "free-flowing", а все параметры и инструменты конфигурируются через один YAML-файл[4]. Проект позиционируется как *state-of-the-art* на бенчмарке SWE-bench, имеет полноценную документацию и ориентирован на исследовательское сообщество[4].

SWE-bench-Live

Для оценки таких агентов создана серия бенчмарков `SWE-bench` и `SWE-bench-Live`. **SWE-bench-Live** — это постоянно обновляемый набор задач, предназначенный для оценки способности AI-систем решать реальные инженерные тикеты. Репозиторий Microsoft объясняет, что датасет обновляется ежемесячно, чтобы обеспечить свежие и "неконтаминированные" задания; он включает более полутора тысяч инстансов из сотен реальных репозиториев[5]. Для создания окружений используется инструмент `RepoLaunch`, который с помощью агента LLM автоматически собирает тестовую среду для любого GitHub-проекта, обеспечивая корректное выполнение тестов[5]. Оценочный код основан на оригинальном SWE-bench, и результаты публикуются в открытом leaderboard.

Выводы и практические рекомендации

- **Агенты = LLM + инструменты + промпт.** Модель должна не просто генерировать текст, а взаимодействовать с внешним миром через набор инструментов. Используйте ReAct-паттерн, чтобы чередовать рассуждения и действия, уменьшать галлюцинации и отслеживать планы.
- **Оркестрация важна.** Для устойчивых, сложных сценариев нужны фреймворки вроде LangGraph: они предоставляют устойчивое выполнение, память, контроль человека, стриминг и инструменты для отладки и развёртывания[3].
- **Используйте готовые агенты и бенчмарки.** SWE-agent и его упрощённые версии показывают, как применять LLM-агентов для исправления кода и поиска ошибок. Бенчмарки SWE-bench и SWE-bench-Live дают объективную метрику и набор задач, постоянно обновляемых для исследования.
- **Многоэтапные инженерные задачи** требуют много циклов ReAct и доступа к большому количеству контекста и кода. Агент должен быть способен планировать, обращаться к внешним источникам и корректировать свой план, а также выдерживать долгие сеансы работы.

Агенты позволяют превратить LLM из «генератора текста» в полноценного **цифрового инженера**, который способен анализировать, принимать решения и выполнять комплексные действия в реальной среде.

Источники

[1] Источник: arxiv.org
[2] Источник: raw.githubusercontent.com
[3] Источник: docs.langchain.com
[4] Источник: raw.githubusercontent.com
[5] Источник: raw.githubusercontent.com