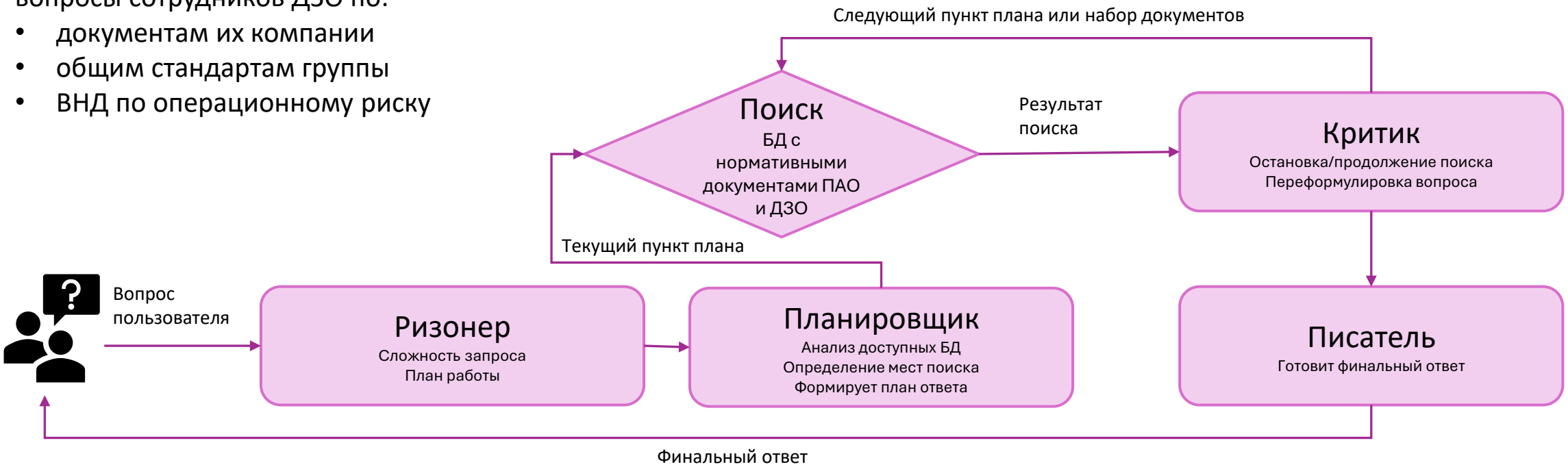
7. Агентные системы и современные подходы к построению - продолжение

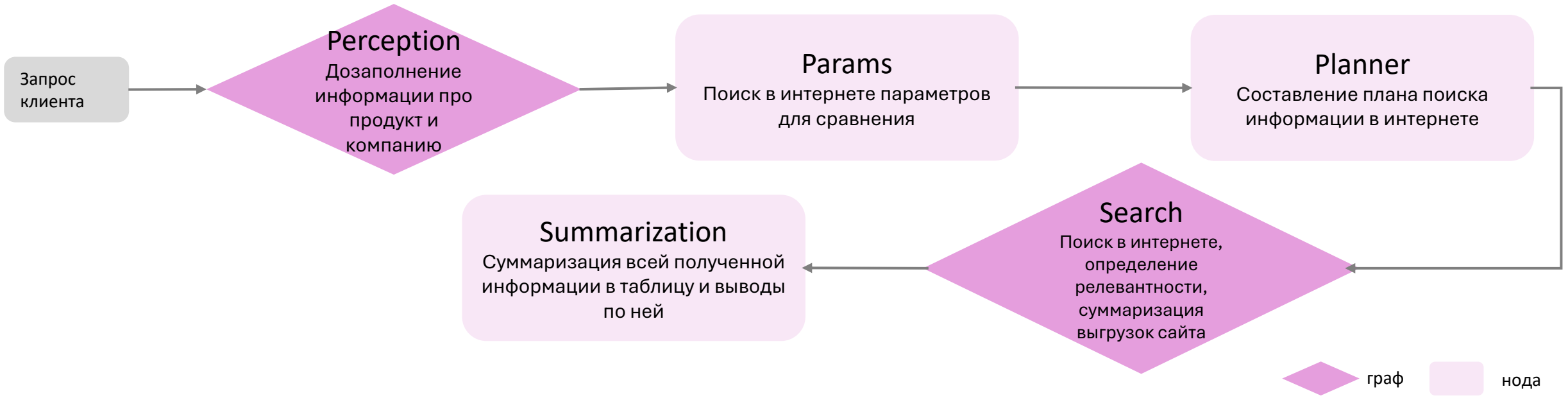
Агент-методолог (Риски)

Агент будет использоваться для ответов на вопросы сотрудников ДЗО по:

- документам их компании
- общим стандартам группы
- ВНД по операционному риску



Концептуальная схема GigaLink



Формат тестовой корзины:

Запрос	Тема	Ресурс	inputs	Составной запрос	**Gold ссылки (модерированные)**	Ответ агента	Логи – state каждого агента
сбербанк кредит	кредит	тренды	{'user_query': 'сбербанк кредит', 'topic': None, 'num_links': 3,'only_rag': False, 'quick_rank': False, 'max_n_iterations':1}	0			

AI-тренажер клиентского менеджера

Цель

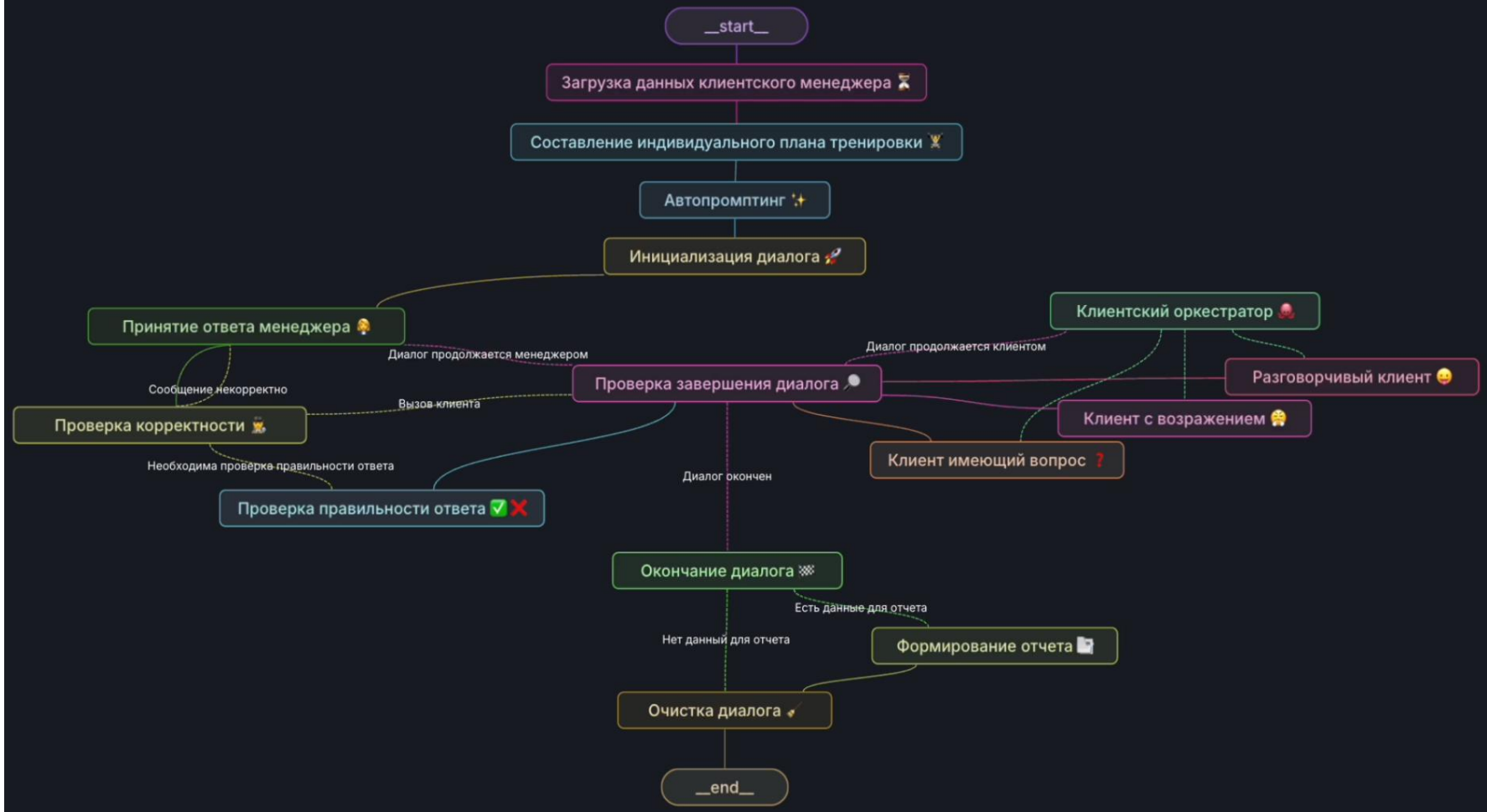
- Довести навык продаж продукта КМ до установленного уровня
- Подготовить КМ к встрече с конкретным клиентом

Роль агента

- Тренер, обучающий продажам
- Автоматизирует руководителей КМ и коучей

Функциональность

- Находит западающие навыки КМ в ИСУ
- Разрабатывает план тренировки
- Симулирует ситуацию с цифровым клиентом
- Оценивает корректность речи и знание условий продукта
- Формирует отчет об успешности обучения, отправляет КМ и его руководителю



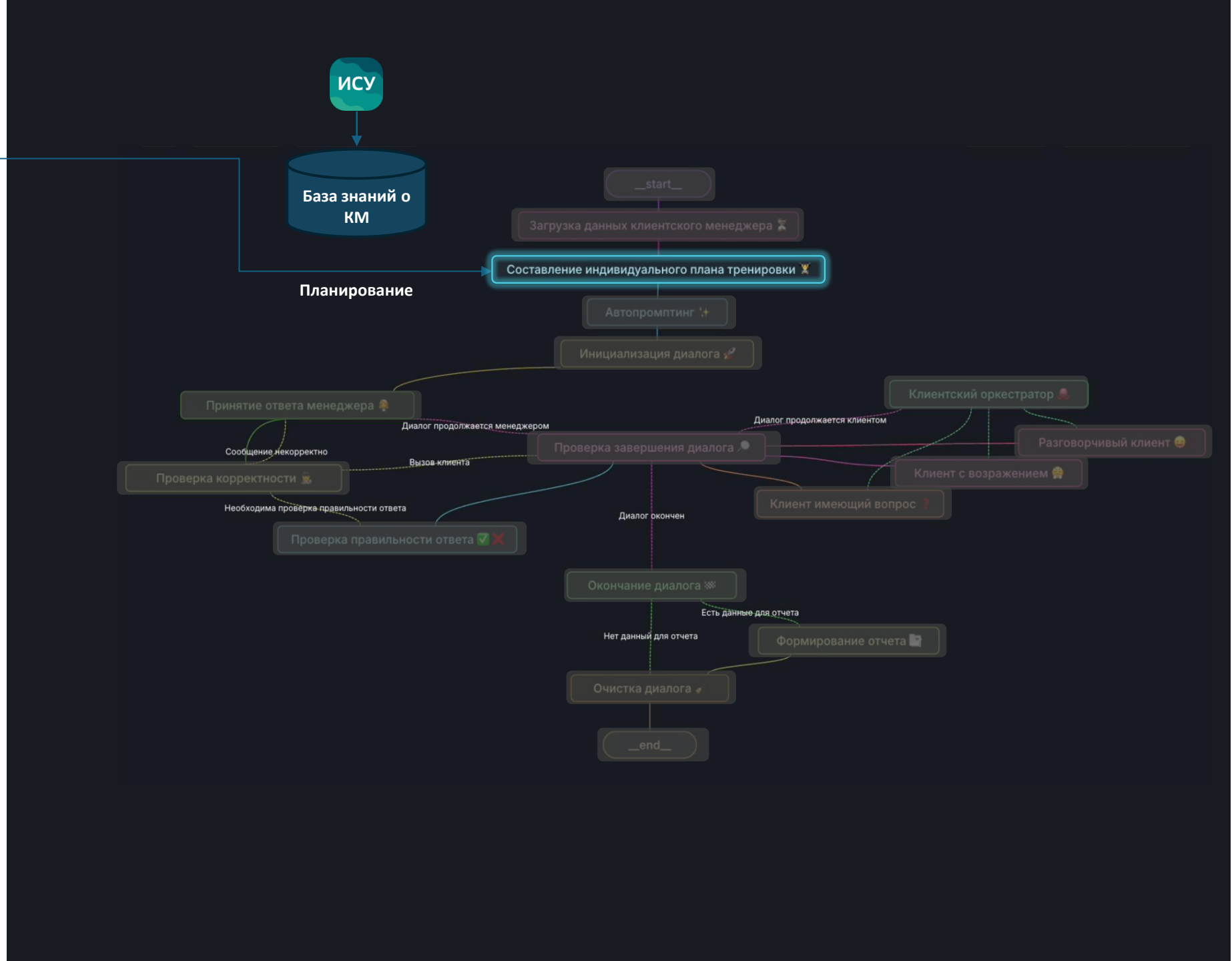
Агент составляет план обучения сотрудника на основе западающих навыков

Агент формирует и запускает релевантную симуляцию клиента

Агент выбирает агента-делегата

Агент получает ответ КМ и проверяет его на корректность

Агент завершает диалог и отправляет отчет о навыках КМ в ИСУ



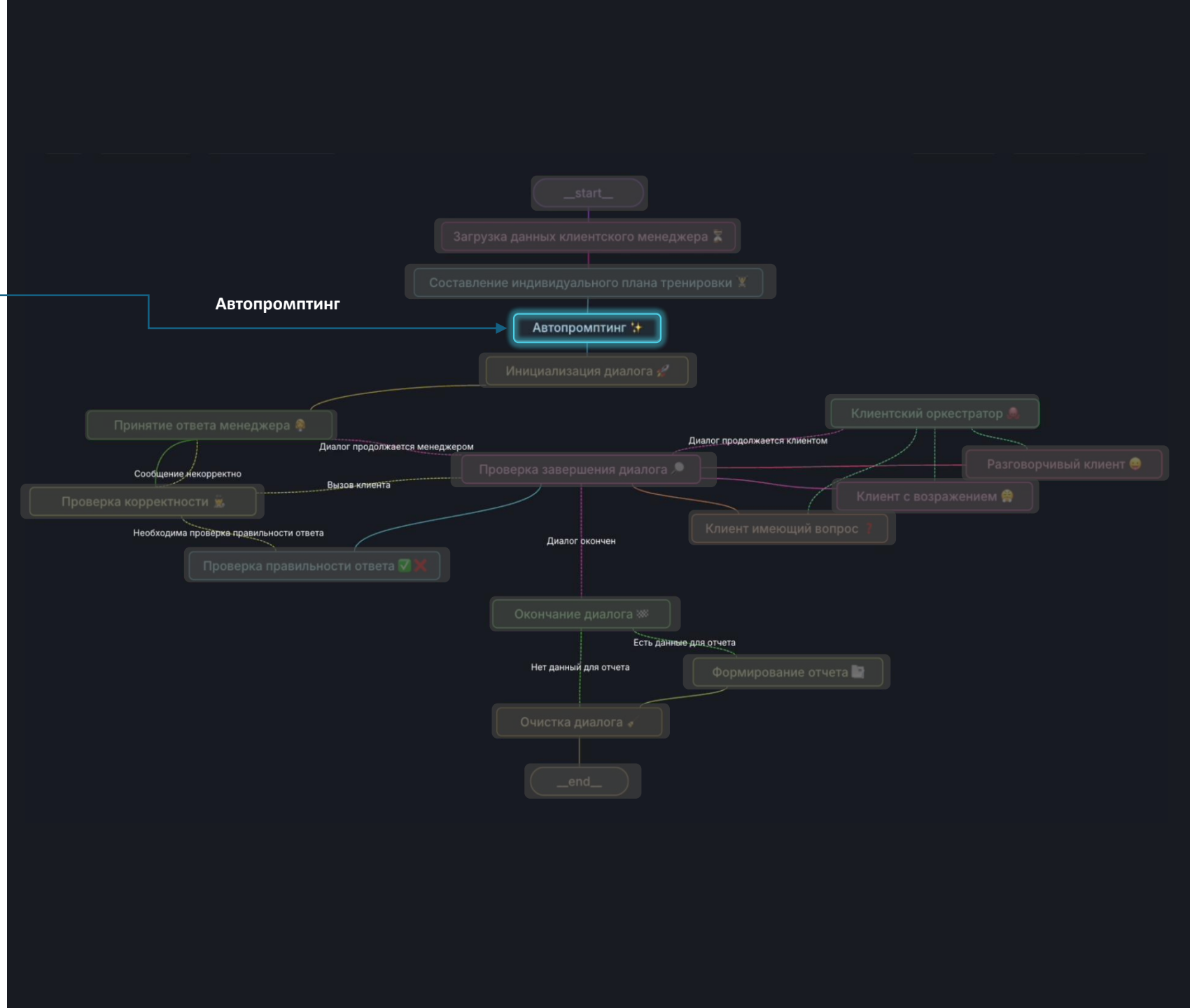
Агент составляет план обучения сотрудника на основе западающих навыков

Агент формирует и запускает релевантную симуляцию клиента

Агент выбирает агента-делегата

Агент получает ответ КМ и проверяет его на корректность

Агент завершает диалог и отправляет отчет о навыках КМ в ИСУ



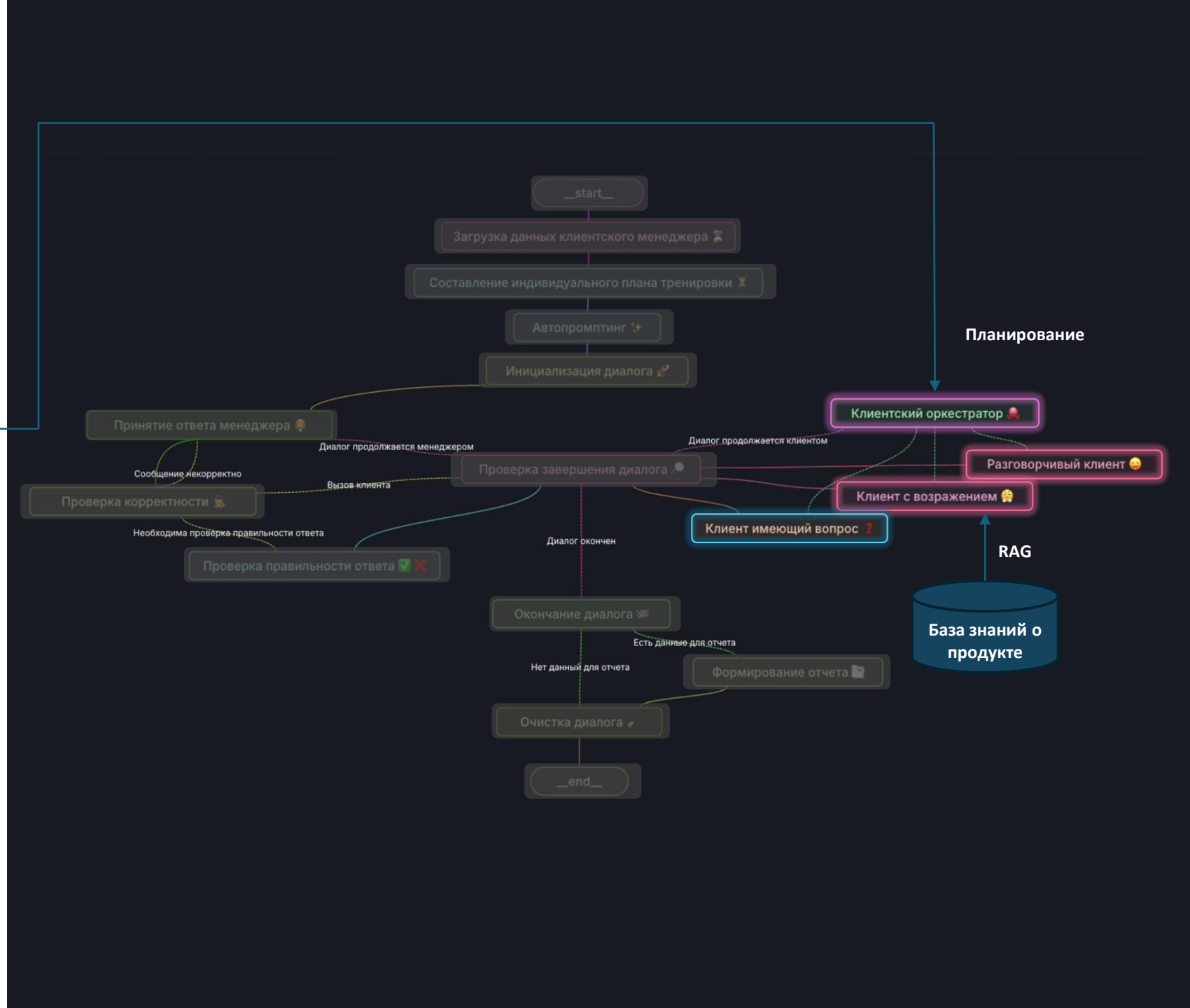
Агент составляет план обучения сотрудника на основе западающих навыков

Агент формирует и запускает релевантную симуляцию клиента

Агент выбирает агента-делегата

Агент получает ответ КМ и проверяет его на корректность

Агент завершает диалог и отправляет отчет о навыках КМ в ИСУ



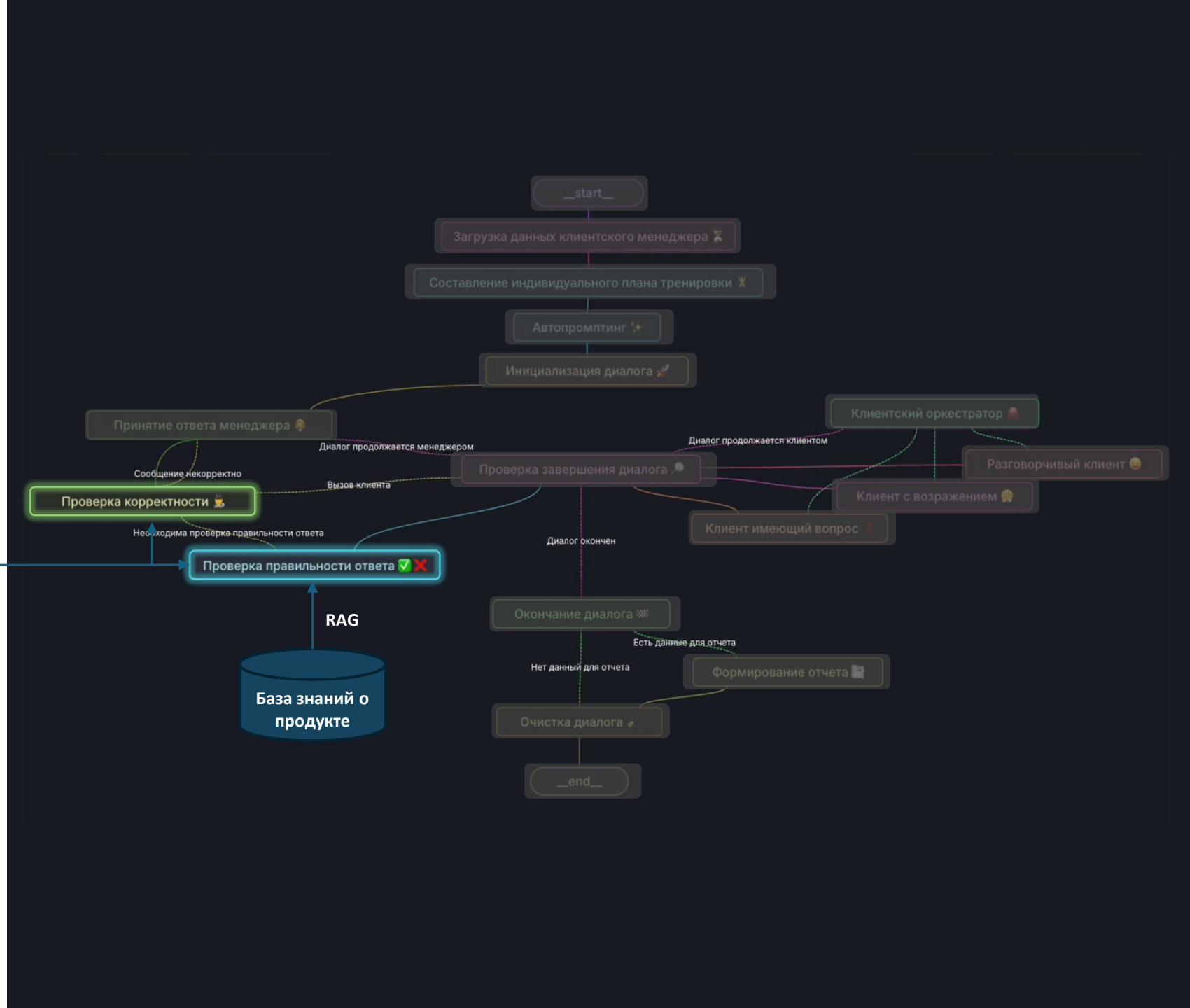
Агент составляет план обучения сотрудника на основе западающих навыков

Агент формирует и запускает релевантную симуляцию клиента

Агент выбирает агента-делегата

Агент получает ответ КМ и проверяет его на корректность

Агент завершает диалог и отправляет отчет о навыках КМ в ИСУ



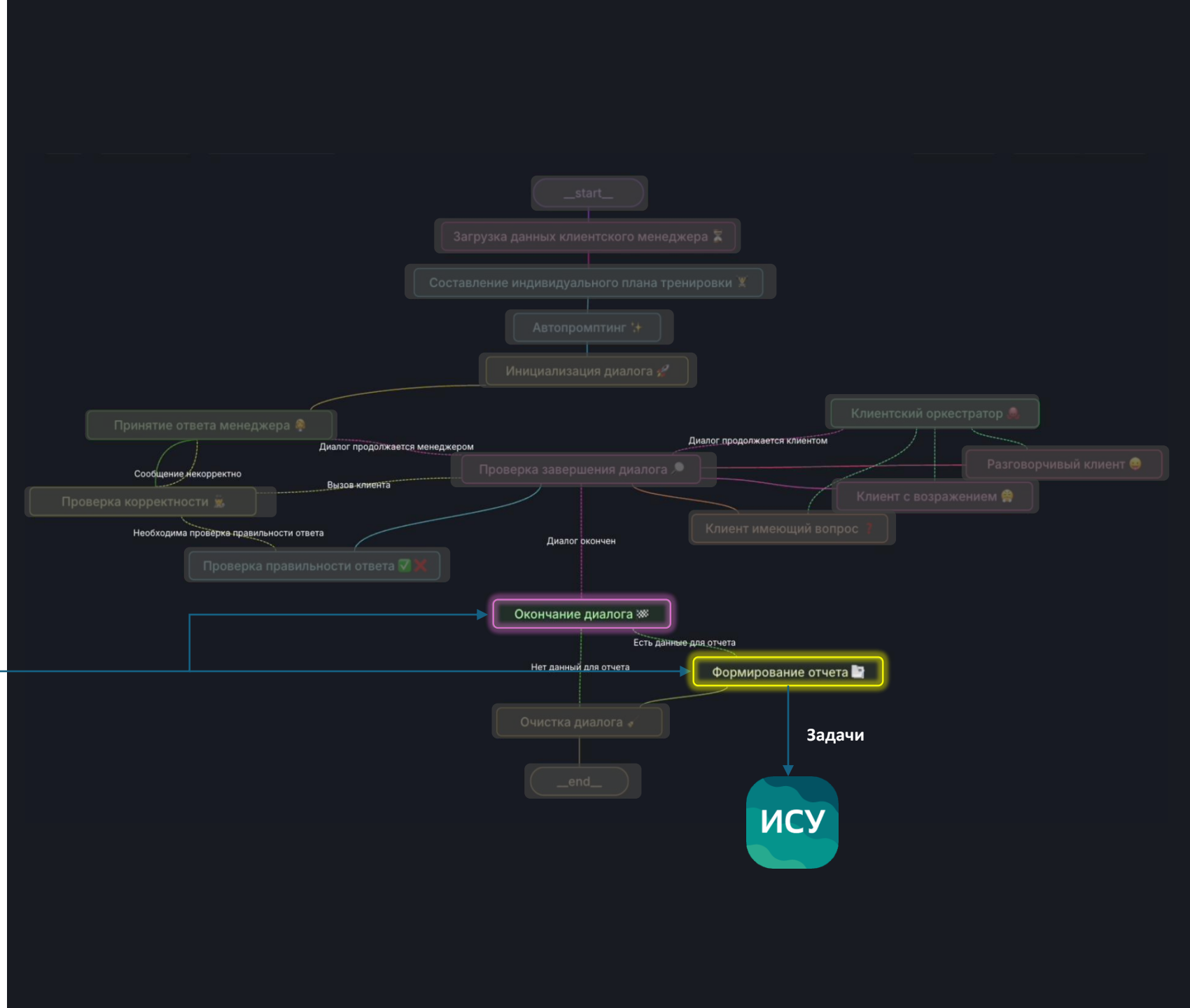
Агент составляет план обучения сотрудника на основе западающих навыков

Агент формирует и запускает релевантную симуляцию клиента

Агент выбирает агента-делегата

Агент получает ответ КМ и проверяет его на корректность

Агент завершает диалог и отправляет отчет о навыках КМ в ИСУ

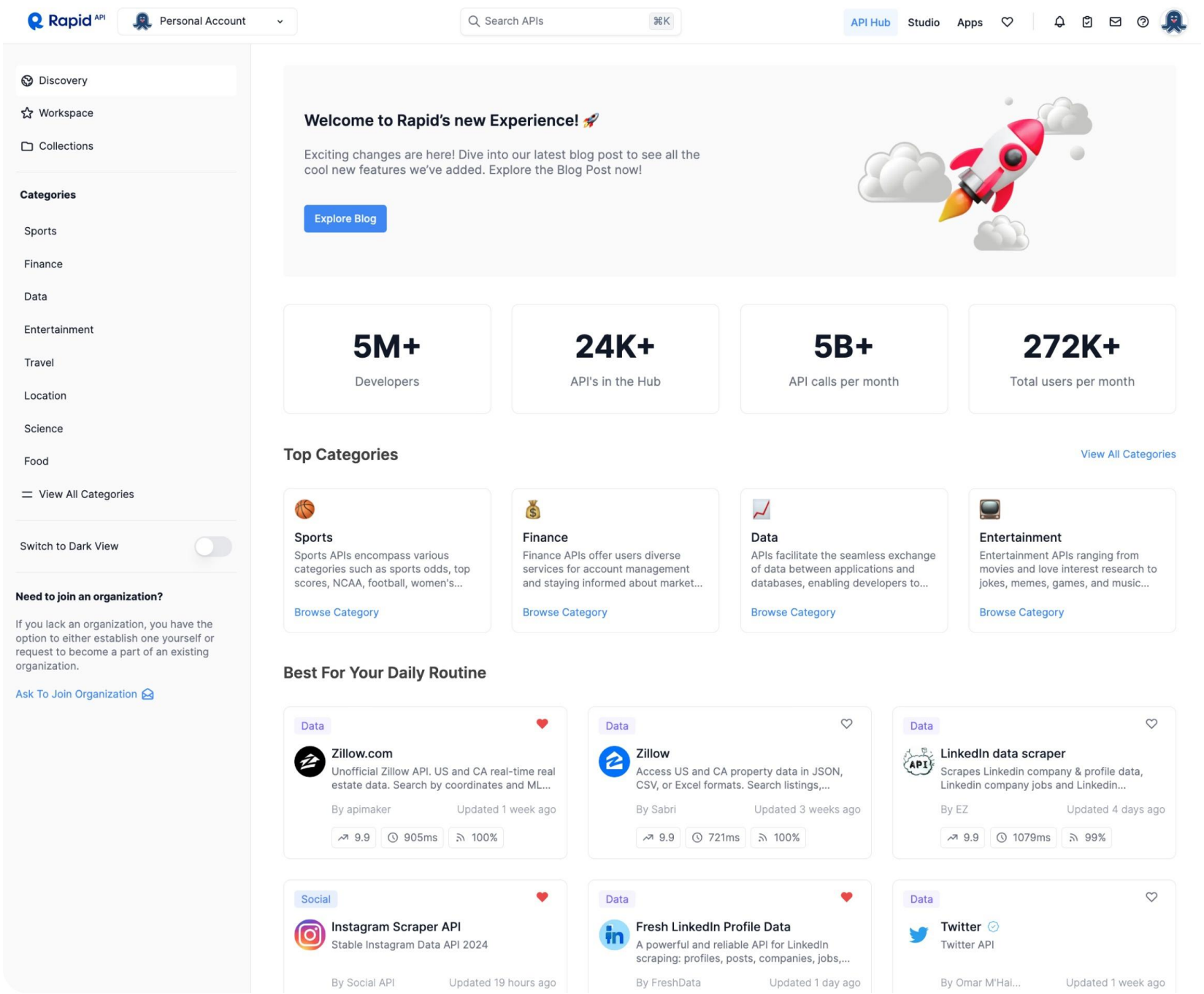


Векторная БД	Поддержка HNSW	Поддержка графовых связей (встроенная)	Хранение метаданных	Гибридный поиск (вектор + текст/фильтры)	Сильные стороны	Слабые стороны
Milvus	Да (индекс HNSW)	Нет (только через внешние связи)	Да	Да (через фильтры)	GPU-ускорение, множество индексов (HNSW/IVF/DiskANN/GPU), доказанный масштаб до миллиардов векторов, production-кейсы	Сложная настройка (etcd+MinIO+Pulsar), требователен к ресурсам, overkill для малых данных
Qdrant	Да (только HNSW для плотных векторов)	Нет (но есть фильтрация по метаданным)	Да	Да (pre-filtering)	Лучшая фильтрация (pre-filtering), Rust надёжность, один исполняемый файл, оптимизация памяти	Деградация на больших объёмах без шардинга
Weaviate	Да (индекс HNSW по умолчанию)	Да (граф знаний, GraphQL API)	Да	Да (BM25 + векторы)	GraphQL API, граф знаний, гибридный поиск (BM25+векторы)	Медленнее для чистого векторного поиска, сложные графовые запросы дорогие
ChromaDB	Нет (использует IVF)	Нет	Да	Ограниченно	Pip install, нулевая конфигурация, LangChain/LlamaIndex first-class	Single-node, лимит ~1M векторов без тюнинга, нет RBAC/HA, высокое потребление памяти
pgvector	Да (с версии 0.8.0)	Нет (но можно использовать PostgreSQL для связей)	Да (через обычные столбцы)	Да (через полнотекстовый поиск PostgreSQL)	SQL интеграция, PostgreSQL инфраструктура, ACID для метаданных, любое масштабирование через Citus	Медленная индексация
Redis Stack	Да (модуль RedisSearch поддерживает HNSW)	Нет	Да	Да (через RedisSearch)	Задержка <10ms, in-memory скорость, протестирован на 1B, поддерживает on-disk ANN	Дорого (in-memory)
Elasticsearch	Да (через поле dense_vector с index_options: hnsw)	Нет (но можно моделировать связи через документы)	Да	Да (full-text + векторы + фильтры)	Mature ecosystem, гибридный (full-text+векторы+фильтры), богатый инструментарий	Не оптимизирован для только векторов, тяжёлый по ресурсам
LanceDB	Нет (использует IVF-PQ)	Нет	Да	Ограниченно	500M+ на одном узле, память 0.1-0.2 GB на 1M (disk), zero-copy версионирование, S3/GCS direct	Молодая экосистема, в основном IVF-PQ, нет кластера
ClickHouse	Да (поддерживает HNSW)	Нет	Да	Да (SQL аналитика + векторы)	SQL аналитика + векторы, гибридные запросы, колонночное хранилище	Не чисто векторная БД, сложная настройка индексов
Vespa	Да (поддерживает HNSW)	Нет (но есть возможности для сложных структур)	Да	Да (векторы+текст+структура)	Billion-scale proven, гибридный (векторы+текст+структура)	Сложная настройка, тяжелая инфраструктура, крутая кривая обучения

Вопросы

Библиотека real-world API

- https://www.rapidapi.com/



Свои функции (tool)

Цель: создать бота, который сообщает о текущем времени в выбранной локации

1

```
def get_current_time(location):
"""Get the current time for a given location"""
print(f"get_current_time called with location: {location}") location_lower = location.lower()
for key, timezone in TIMEZONE_DATA.items():
        if key in location_lower:
                print(f"Timezone found for {key}")
                current_time = datetime.now(ZoneInfo(timezone)).strftime("%I:%M %p")
                return json.dumps({
                        "location": location,
                        "current_time": current_time })
print(f"No timezone data found for {location_lower}")
return json.dumps({"location": location, "current_time": "unknown"})
```

Влияет на качество!!

Формат 0_0

https://github.com/RutamBhagat/LangChainHCCourse3/blob/main/course_3/OpenAI_Function_Calling.ipynb
https://github.com/ai-forever/gigachain/blob/master/cookbook/gigachat_functions_agent.ipynb - гига
https://learn.microsoft.com/en-us/azure/ai-services/openai/how-to/function-calling
https://developers.sber.ru/docs/ru/gigachat/api/function-calling

Нейминг

- **Используйте краткие, описательные имена** в формате snake_case (например: get_weather, calculate_tax, search_database)
- **Избегайте версионирования в имени** для основных функций. Если необходима версия, добавляйте её как суффикс (_v2) только при параллельной поддержке старых версий
- **Имя должно начинаться с глагола**, отражающего действие: fetch_, calculate_, update_, create_, delete_

Описание

- **Первое предложение — ключевое**: начинайте с чёткого глагола и основного назначения ("Получает текущую погоду для указанного города")
- **Укажите источник данных** во втором предложении, если это важно ("Использует данные OpenWeatherMap с обновлением каждые 10 минут")
- **Приведите краткий пример использования** в конце описания, если функция сложна

Свои функции (tool)

2

```
messages = [{"role": "user", "content": "What's the current time in San Francisco"}] # Single function call #
```

3

```
tools = [ { "type": "function",
"function": {
        "name": "get_current_time",
        "description": "Get the current time in a given location",
        "parameters": {
                "type": "object",
                "properties": { "location": { "type": "string", "description": "The city name, e.g. San Francisco", }, },
                "required": ["location"], }, } } ]
```

4

```
response = client.chat.completions.create(
model=name_model,
messages=messages,
tools=tools,
tool_choice="auto", )
```

Свои функции (tool)

5

```
if response_message.tool_calls:
        for tool_call in response_message.tool_calls:
            if tool_call.function.name == "get_current_time":
                function_args = json.loads(tool_call.function.arguments)
                print(f"Function arguments: {function_args}")
                time_response = get_current_time( location=function_args.get("location") )
                messages.append({ "tool_call_id": tool_call.id,
                                  "role": "tool",
                                  "name": "get_current_time",
                                  "content": time_response, })
else: print("No tool calls were made by the model.")
```

6

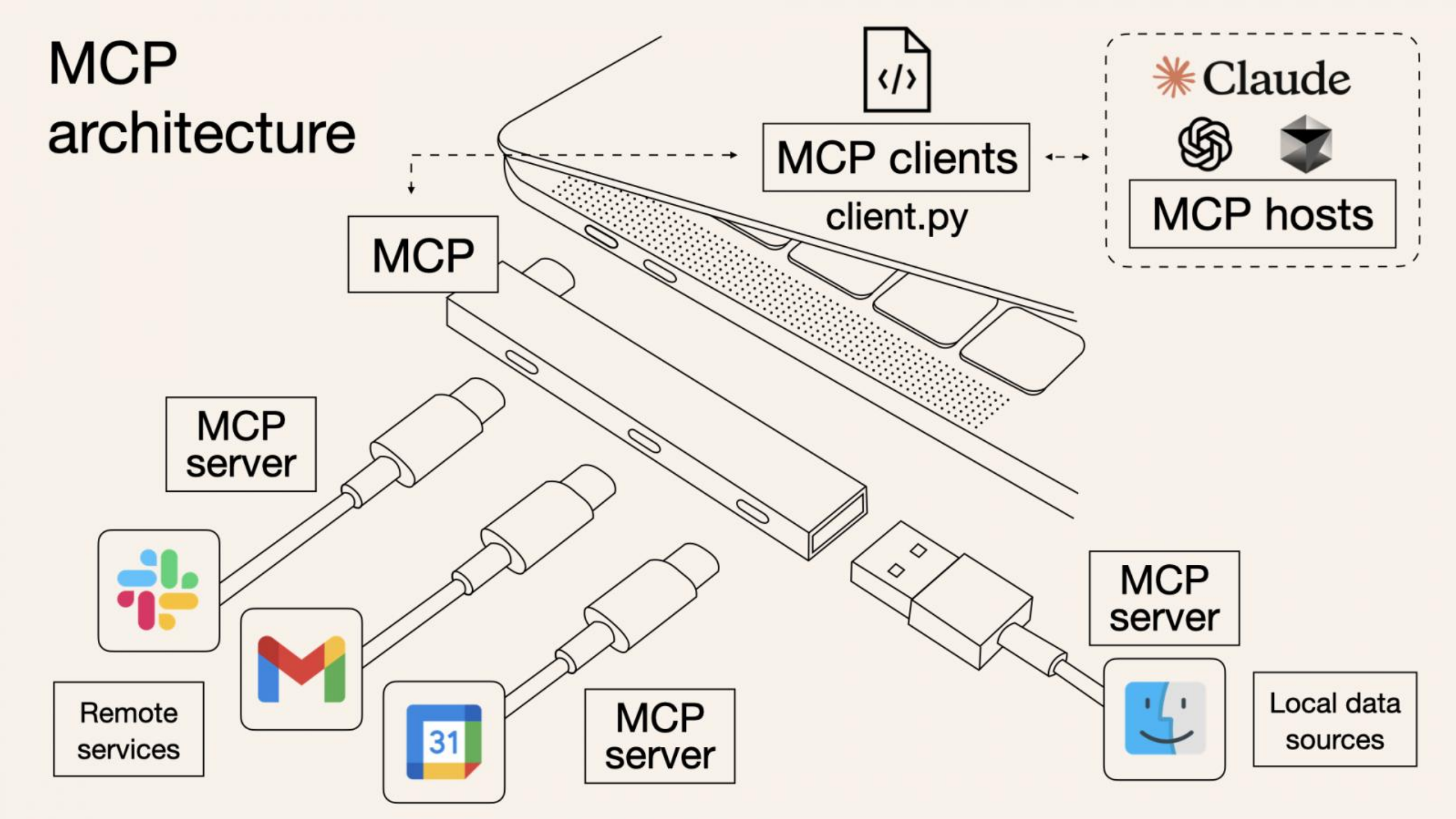
```
final_response = client.chat.completions.create( model=deployment_name, messages=messages, )
```

Эдвенсед тул юзинг

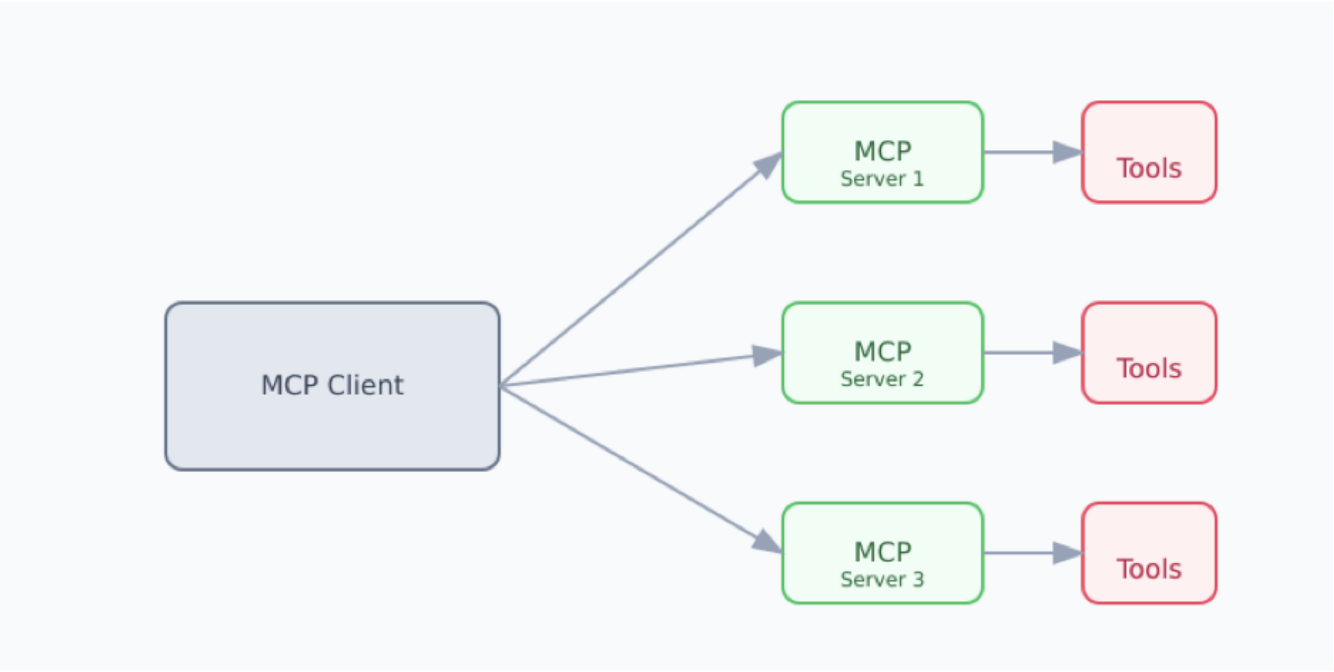
- Параллельный вызов функций
- Вызов нескольких функций + агрегация результата
- Дополнение описания функции деталями.
- Дополнение системного промпта
- Попросите модель задавать уточняющие вопросы, если данного контекста недостаточно (не все атрибуты функций можно заполнить)
- "Используйте только те функции, которые вам были предоставлены« (если LLM пытается обратиться к функциям, которых нет)

How to improve tool calling

- Не передавать все функции сразу (чтобы не перегружать контекст передаем только нужные функции, а не про запас)
- Группировать функции по смыслу (математические функции вместе)
- Загружать динамически при необходимости (учет прав пользователя, не забивать контекст)


MCP
architecture
MCP clients
client.py
Claude
MCP hosts
MCP
MCP
server
MCP
server
MCP
server
Remote
services
Local data
sources

MCP



Роль MCP-сервера — организовать диалог пользователя и tool через llm.

Преимущества:
- Быстрое добавление нового tool
- Унификация tool для LLM на стороне MCP

Проблемы:
- Безопасность

Создать свой сервер: https://blog.openreplay.com/ru/создать-mcp-сервер-пошагово-примеры-кода/

Имеющиеся сервера: https://github.com/punkpeye/awesome-mcp-servers https://mcpservers.org/

https://yandex.cloud/ru/docs/glossary/mcp?utm_referrer=https%3A%2F%2Fyandex.ru%2F

LINUX FOUNDATION

Structure output

```
response = client.chat.completions.create(
response_format={
    "type": "json_schema",
    "json_schema": {
        "name": "math_reasoning",
        "schema": {
            "type": "object",
            "properties": {
                "steps": {
                    "type": "array",
                    "items": {
                        "type": "object",
                        "properties": {
                            "explanation": {"type": "string"},
                            "output": {"type": "string"}
                        },
                        "required": ["explanation", "output"],
                        "additionalProperties": False
                    }
                },
                "final_answer": {"type": "string"}
            },
            "required": ["steps", "final_answer"],
            "additionalProperties": False
        },
        "strict": True
    }
}
```

https://cookbook.openai.com/examples/structured_outputs_intro

```
from pydantic import BaseModel

class MathReasoning(BaseModel):
    class Step(BaseModel):
        explanation: str
        output: str

    steps: list[Step]
    final_answer: str
--
response_format=MathReasoning,
```

```
result = json.loads(response)
steps = result['steps']
final_answer = result['final_answer']
for i in range(len(steps)):
    print(f"Step {i+1}: {steps[i]['explanation']}\n")
    display(Math(steps[i]['output']))
    print("\n")
```

Использовать description

Использовать строгий режим (**strict: true**) - модель точно соответствовала вашей схеме

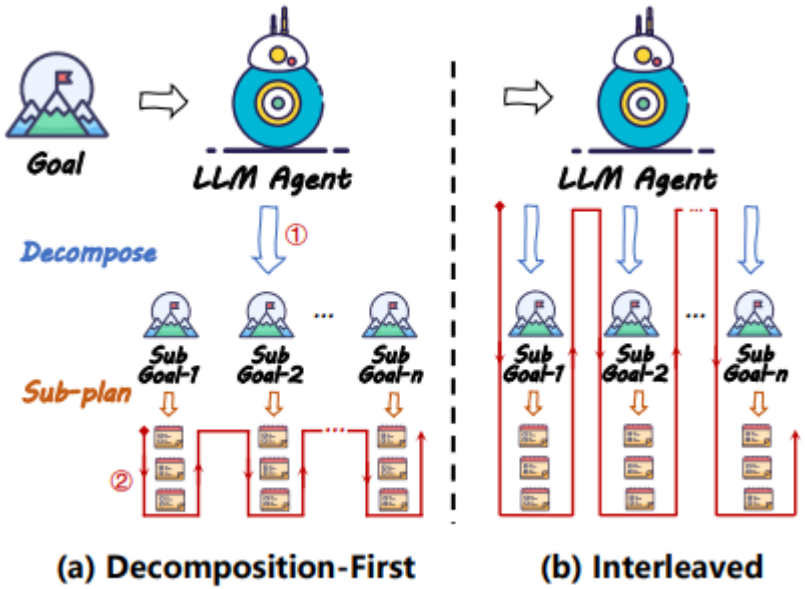
Надежность: fallback-механизмы

- **Альтернативные источники**: если основной API недоступен, используйте запасной. Bypass
- **Деградация функциональности**: предоставьте частичный ответ с пометкой об ограничениях (или welcome-сообщении)
- **Кэширование данных**: используйте последние известные данные с указанием времени актуальности

Вопросы

Планинг

Способы (в общем):



Метод	Идея	LLM's task
Декомпозиция задачи (a)	Разделяй и властвуй	Task decomposition Subtask planning
Многоплановый выбор (выбор инструкции)	Создайте несколько планов и выберите оптимальный	Plans generation Plans evaluation
Внешний планировщик с поддержкой (планировщик – tool)	Формализовать задачу и вызвать внешний планировщик	Task formalization
Рефлексия и Улучшение (b)	Размышление на основе опыта и уточнение плана	Plan generation Reflection Refinement
С использованием внешней памяти (support к другим методам, RAG-style)	Использование памяти для облегчения планирования	Plan generation Memory extraction

https://arxiv.org/pdf/2402.02716

Reasoning – Chain-of-thought на стероидах

O1 от openai для задач идет по плану:

- *Анализ проблемы* — модель переписывает задачу своими словами, выявляет ключевые условия.
- *Декомпозиция на подзадачи* — разбивает сложную задачу на более простые части.
- *Систематическое решение* — последовательно решает каждую часть, шаг за шагом.
- *Альтернативные решения* — при необходимости рассматривает разные подходы или проверяет несколько вариантов решения.
- *Самопроверка* — периодически проверяет, соответствует ли промежуточный результат условию, нет ли противоречий.
- *Самокоррекция* — если обнаружена ошибка или противоречие, модель возвращается и исправляет, прежде чем идти дальше.

Лучшая практика — указать на необходимость рассуждения или «Покажи ход решения», «Объясни, как ты пришел к ответу», «Решай поэтапно»

Другие приемы:

1. Self-Consistency - Вместо одного прогона «CoT» - N разных и голосованием определяют итог.

2. **Scratchpad** - модель записывает промежуточные вычисления (арифметику, список аргументов) в отдельный «Блокнот» (scratchpad).

3. Калибровка уверенности (**Uncertainty Calibration**) - temperature scaling, MC-dropout делают распределение вероятностей более честным.

4. Интерактивное уточнение (**Iterative Refinement**) - Модель генерирует черновой ответ, потом по цепочке «само-вопросов» анализирует слабые места и улучшает его.

5. Использование **prefix-tree (trie)** (хранить ассоциативный массив,), чтобы не дать модели «уйти в рассуждения» не по теме.

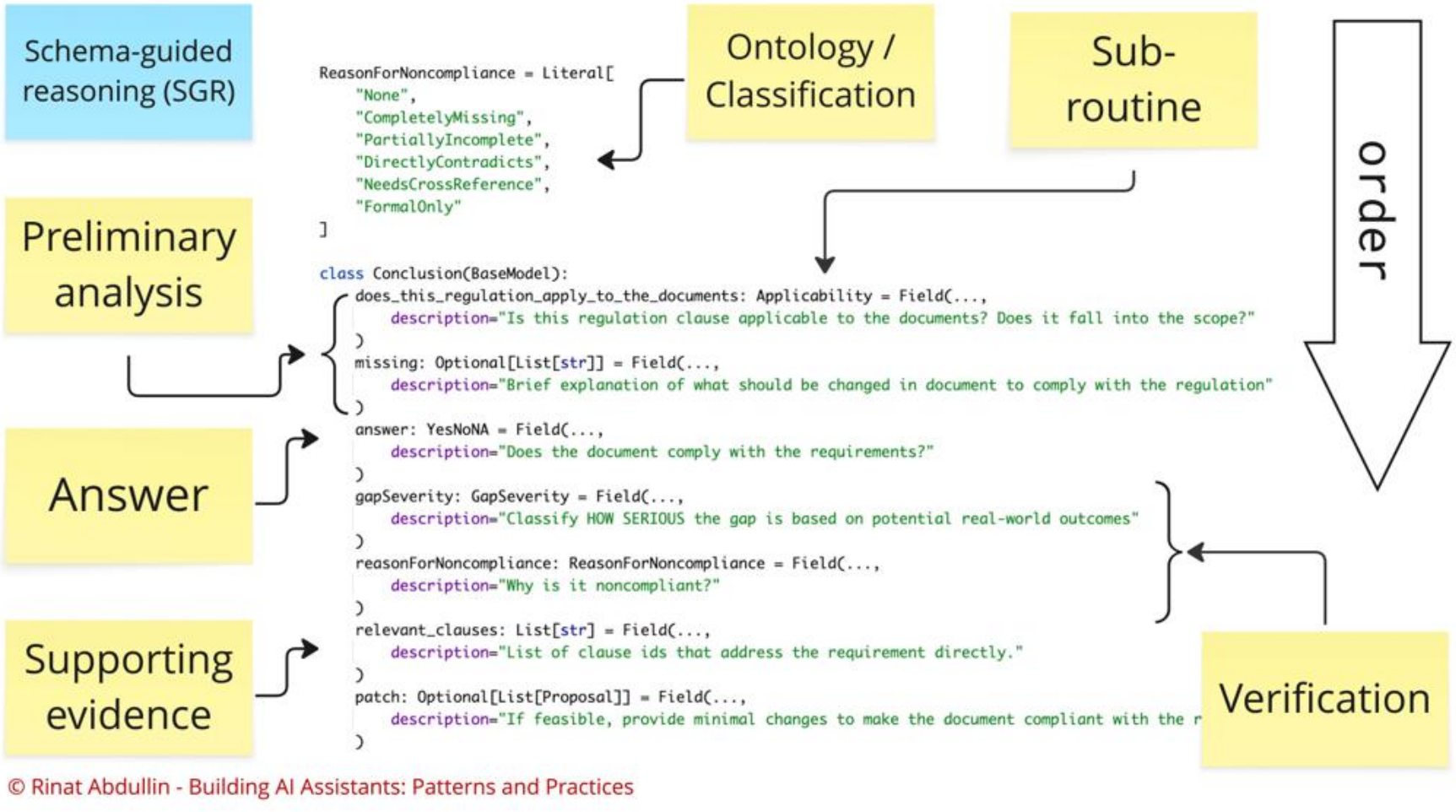
6. RAG из логических цепочек подтягивается по мере необходимости

Schema-Guided Reasoning

Структурированное описание того, как модель должна рассуждать

Паттерны:

- Каскад — последовательные шаги
- Маршрутизация — выбор подхода под задачу
- Цикл — уточнение через повтор



Мысленный контрольный список эксперта в предметной области -> структурированная схема рассуждений для LLM.

Вопросы

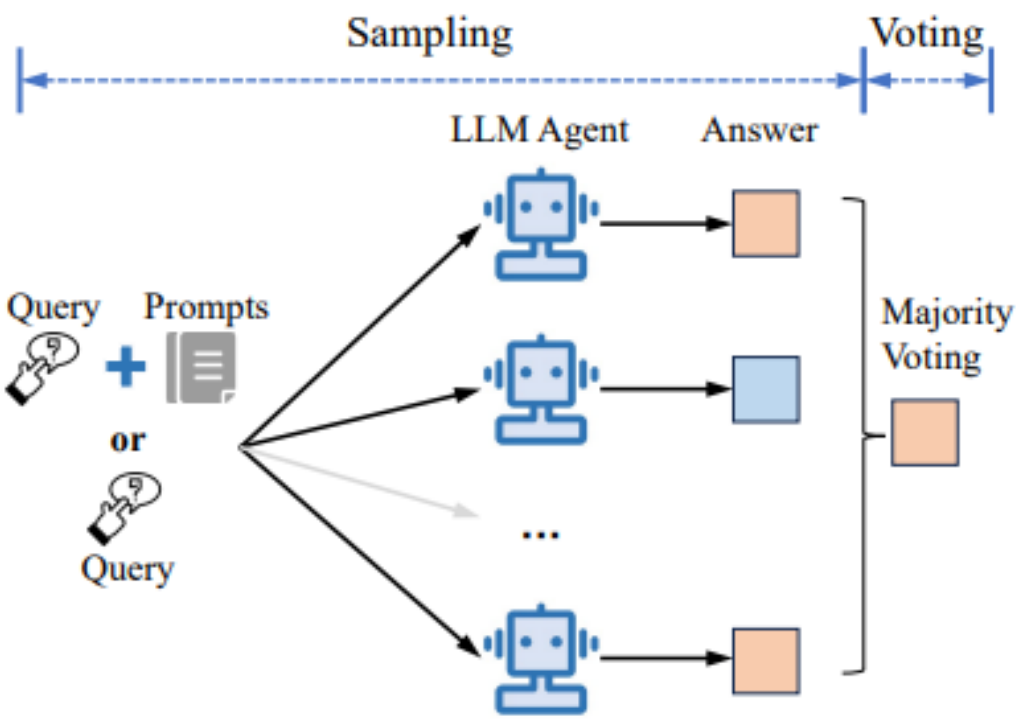
Мультиагентность

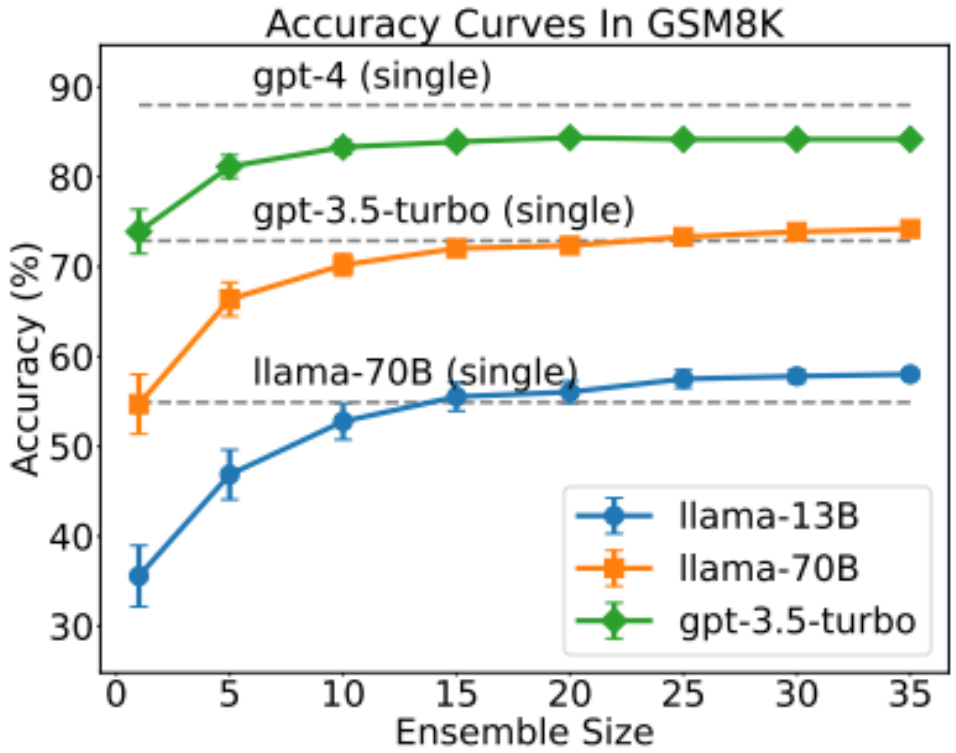
- **Модульность**: По аналогии с микросервисной архитектурой - проще разработать отдельных агентов для каждой из целей.
- **Экспертность** (специализация): Можно создавать агентов-экспертов, специализирующихся на отдельных доменах, что поможет повысить качество.
- **Контроль**: Коммуникацию агентов гораздо проще контролировать (в отличии от вызова функций).

Мультиагентность: повышение качества

https://arxiv.org/pdf/2402.05120

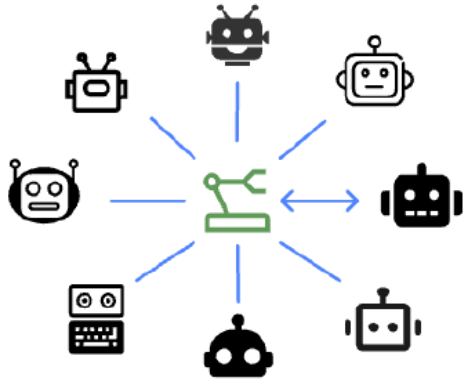
-> На 15 п.п. лучше, в 64 раза дороже и дольше



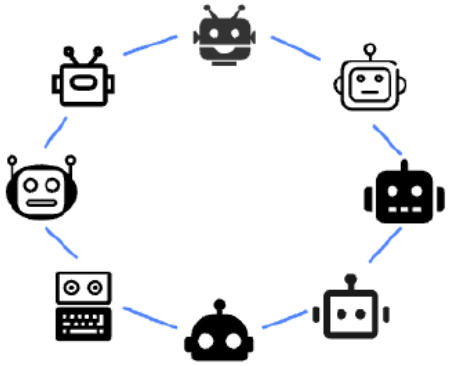


Варианты организации взаимодействия Агентов

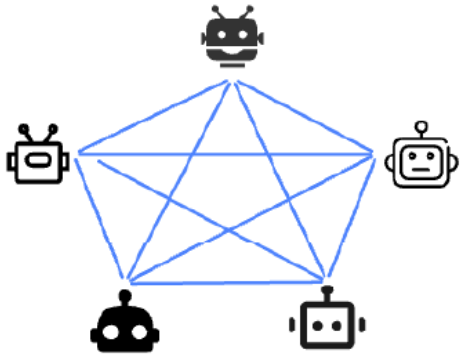
Проще – лучше. Усложнять только если возникает необходимсть!



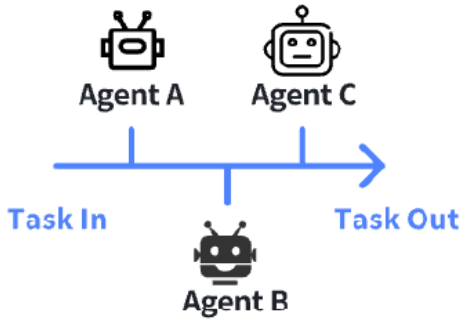
Star



Ring



Graph



Bus

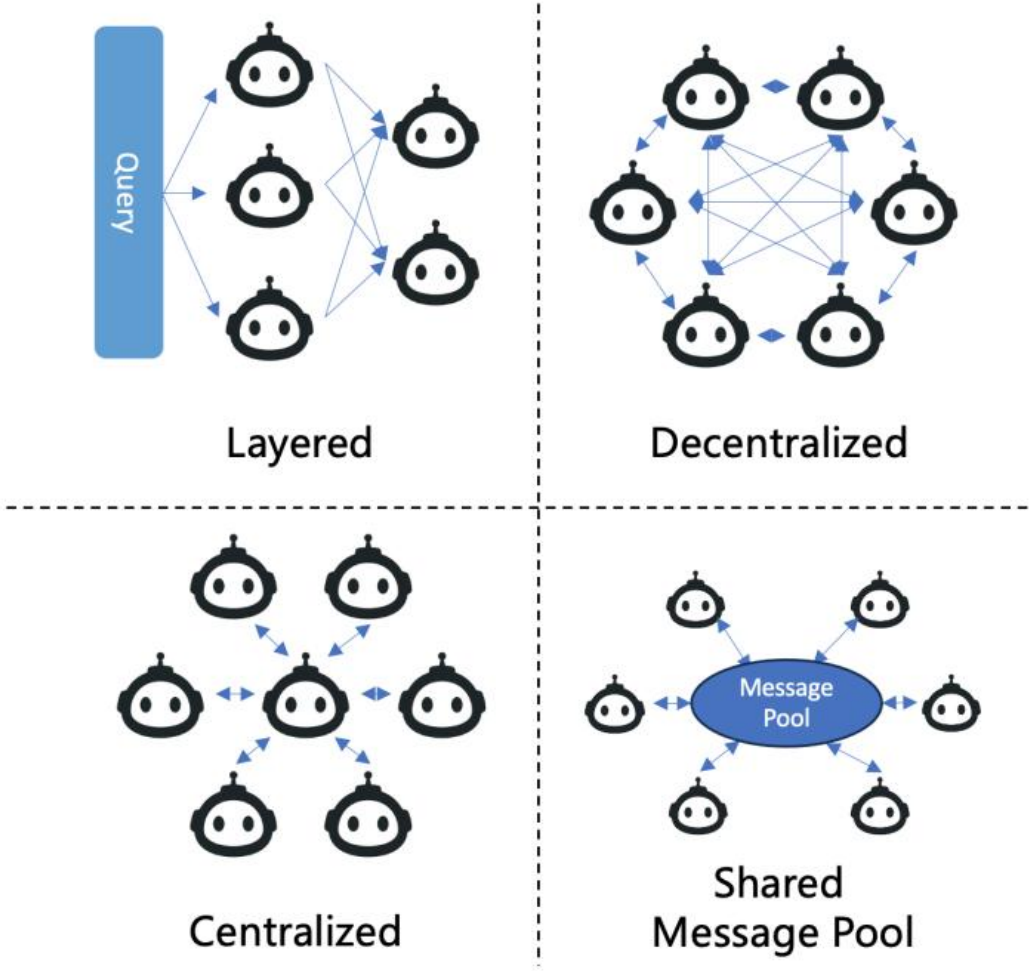
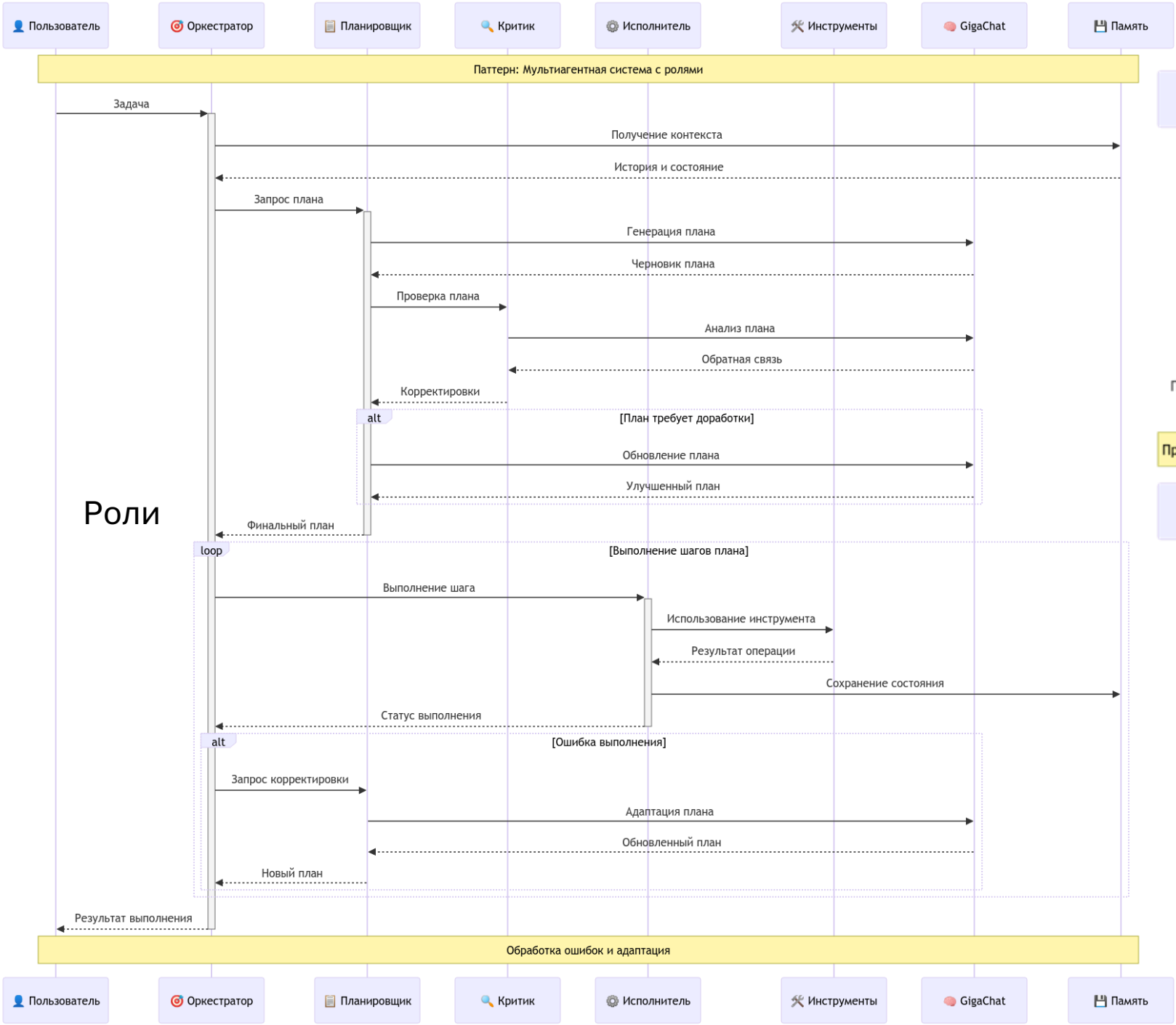


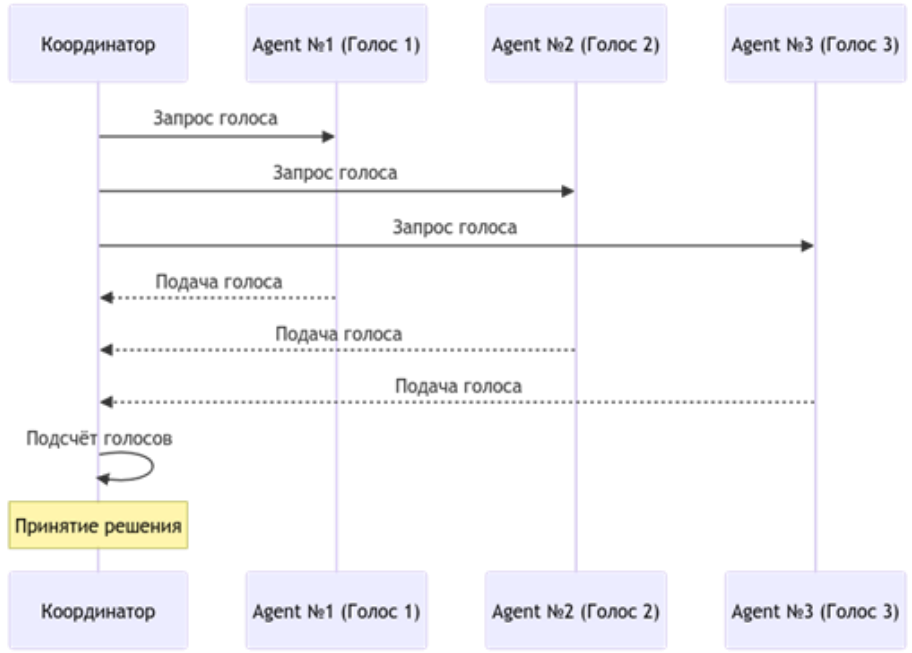
Figure 3: The Agent Communication Structure.

https://langchain-ai.github.io/langgraphjs/concepts/multi_agent/#multi-agent-architectures

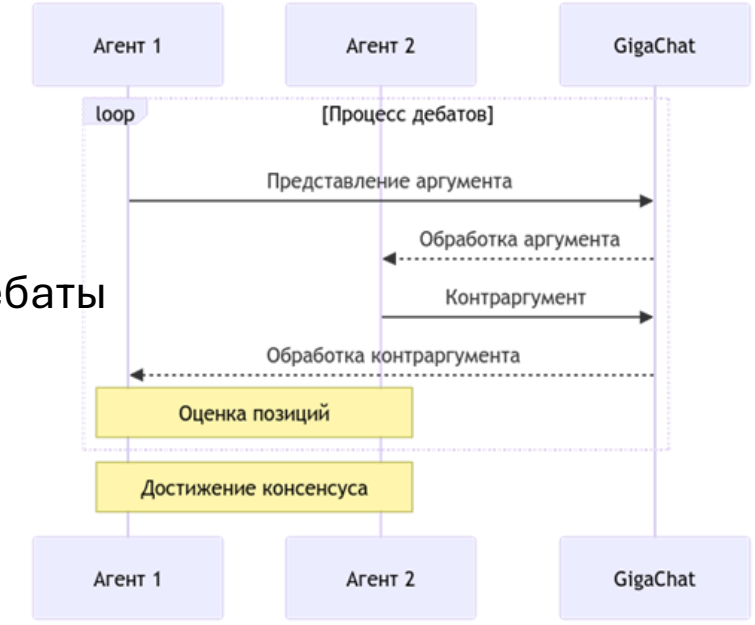


Роли

Голосование

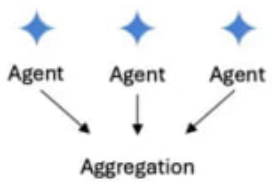
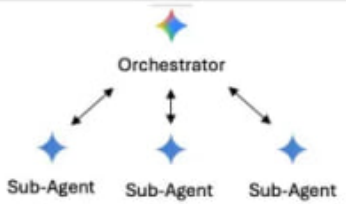
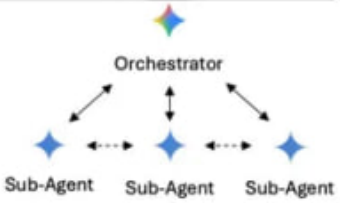


Дебаты



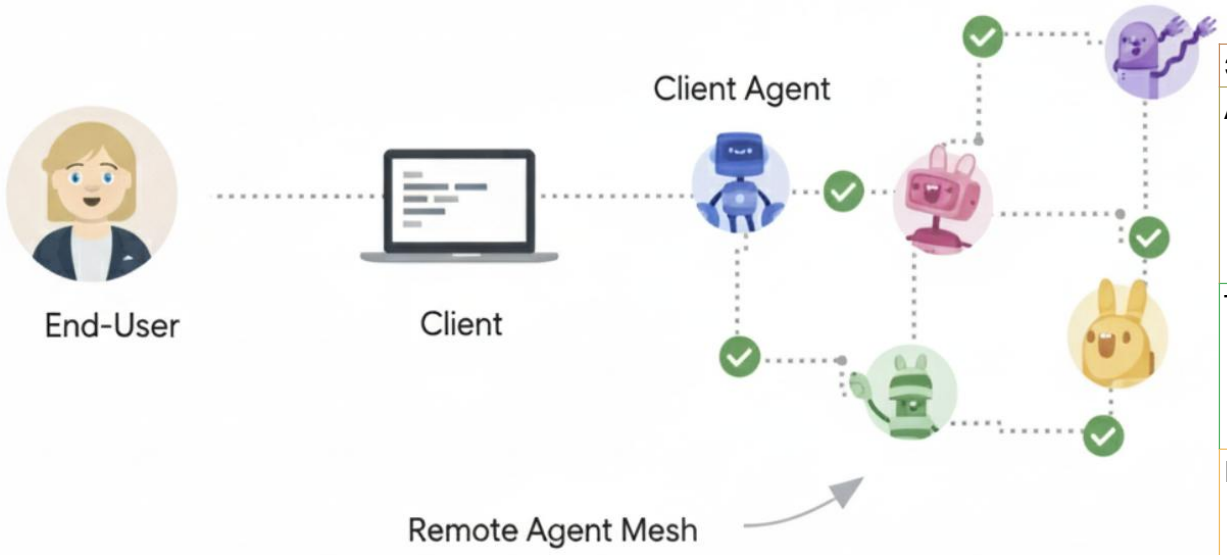
Towards a Science of Scaling Agent Systems

https://arxiv.org/pdf/2512.08296v1

Characteristic	SAS	MAS (Independent)	MAS (Decentralized)	MAS (Centralized)	MAS (Hybrid)
Interaction Type					
LLM Calls	$O(k)$	$O(nk) + O(1)$	$O(dnk) + O(1)$	$O(rnk) + O(r)$	$O(rnk) + O(r) + O(p)$
Sequential Depth	k	k	d	r	r
Comm. Overhead	0	1	$d \cdot n$	$r \cdot n$	$r \cdot n + p \cdot m$
Parallelization Factor	1	n	n	n	n
Memory Complexity	$O(k)$	$O(n \cdot k)$	$O(d \cdot n \cdot k)$	$O(r \cdot n \cdot k)$	$O(r \cdot n \cdot k + p \cdot n)$
Coordination	Sequential	Parallel + Synthesis	Sequential Debate	Hierarchical	Hierarchical + Peer
Consensus	-	Synthesis	Debate	Orchestrator	Orchestrator

A2A Protocol – между своими и чужими агентами

A2A обеспечивает окончательный общий язык для взаимодействия агентов



Remote Agent - A2A Server – »черный ящик»

https://a2a-protocol.org/

Элемент	Описание	Ключевая цель
Agent Card	Документ с метаданными в формате JSON, описывающий личность агента, его возможности, конечную точку, навыки и требования к аутентификации.	Позволяет клиентам находить агентов и понимать, как безопасно и эффективно с ними взаимодействовать.
Task	Единица работы с отслеживанием состояния, инициируемая агентом, с уникальным идентификатором и определенным жизненным циклом.	Облегчает отслеживание длительных операций и обеспечивает многоэтапное взаимодействие и совместную работу.
Message	Один этап взаимодействия между клиентом и агентом, включающий в себя контент и роль («пользователь» или «агент»).	Передает инструкции, контекст, вопросы, ответы или обновления статуса, которые не обязательно являются официальными документами.
Part	Основной контейнер содержимого (например, TextPart, FilePart, DataPart), используемый в сообщениях и артефактах.	Обеспечивает гибкость, необходимую агентам для обмена различными типами контента в сообщениях и артефактах.
Artifact	Материальный результат, полученный агентом в ходе выполнения задачи (например, документ, изображение или структурированные данные).	Предоставляет конкретные результаты работы агента, обеспечивая структурированность и возможность извлечения данных.

LINUX FOUNDATION

Google

Agora – организация взаимодействия для своих агентов

- https://arxiv.org/html/2410.11905v1 – Agora A2A communication protocol:

- **Структурированных данных** для частых коммуникаций, таких как регулярные запросы данных.

- **Естественного языка** для редких или сложных взаимодействий, где требуется гибкость.

- **Рутины, написанные LLM,** для промежуточных случаев, где требуется автоматизация, но не постоянное использование LLM.

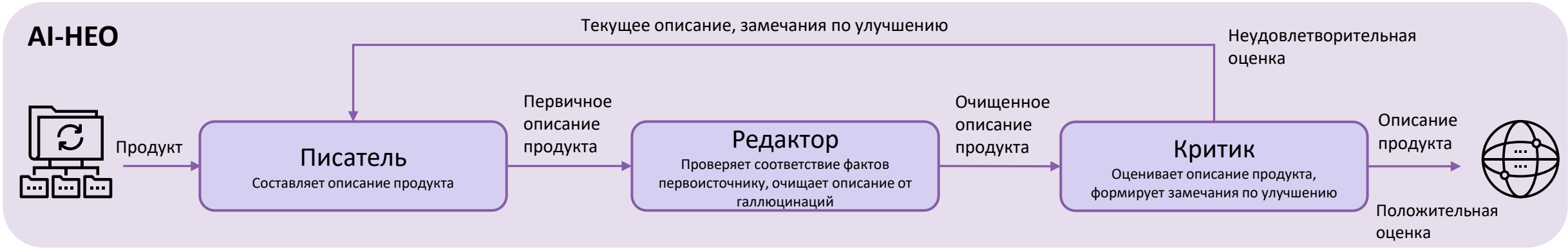
А чем еще управлять?

- Распределение кредитов (в зависимости от их вклада в выполнение задачи)
- Управление опытом (Experience Management Protocol - сбор, хранение и анализ опыта, полученного агентами)
- Общение через события (агент 1 добавил товар в корзину, агент 2 начал прогноз доставки)

HEO: AI описание дата продуктов

Агентная система описаний дата-продуктов и их составляющих (таблиц и колонок) для задач поиска

увеличение релевантности и сокращение времени поиска нужного пользователю дата-продукта



Агент по балансу карты на номере 900

Агентная система на платформе Skillflow

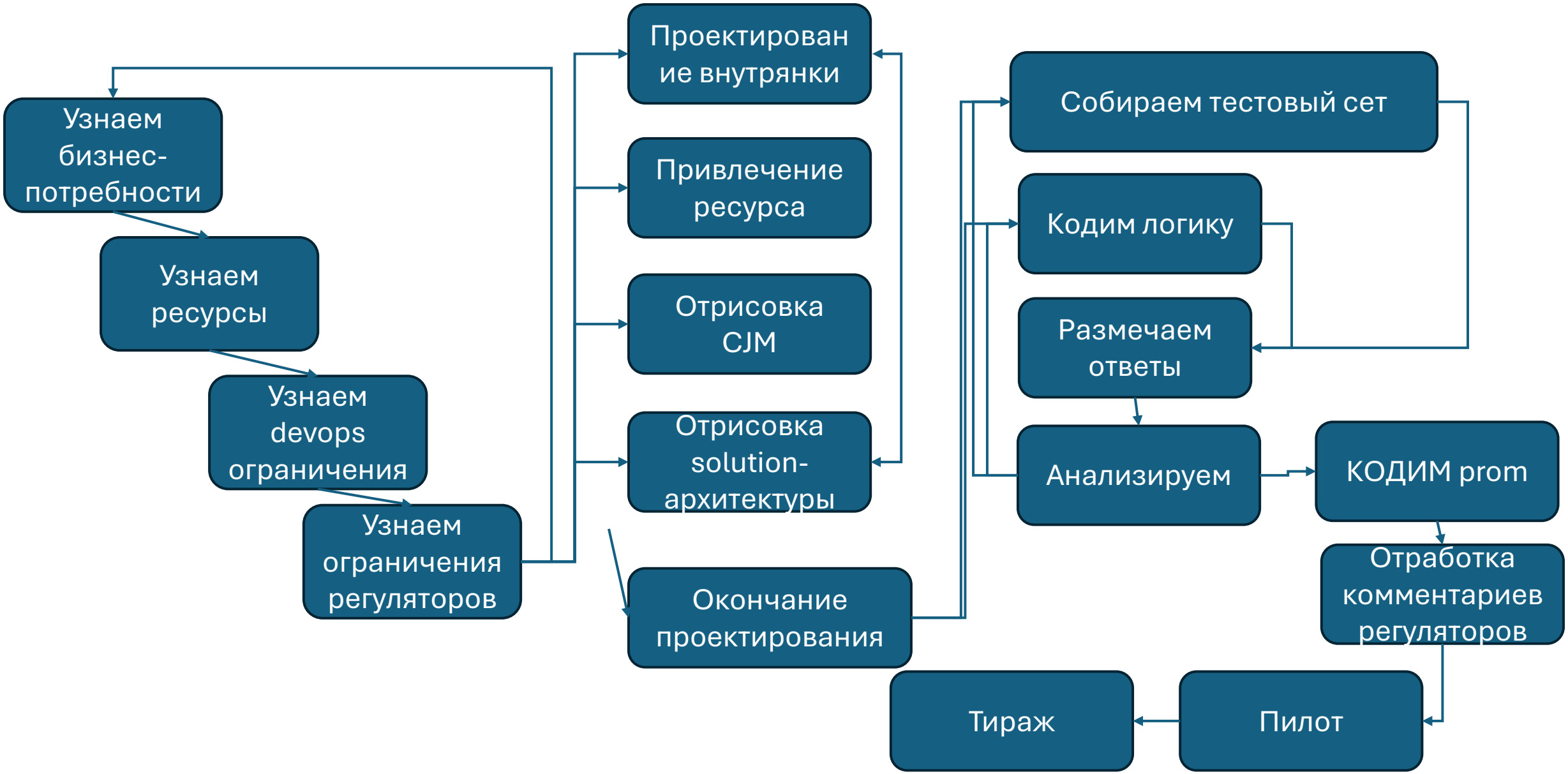
- Сокращение времени решения вопроса клиента
- Снижение ресурса разработчиков на поддержание и обновление статичной логики обработки запроса клиента

У клиента расширена вариативность информации о балансе карты: он может узнать баланс по одной, потом по второй карте. Узнать баланс по нескольким картам суммарно. Узнать хватит ли его денежных средств на покупку, если озвучит ее стоимость.

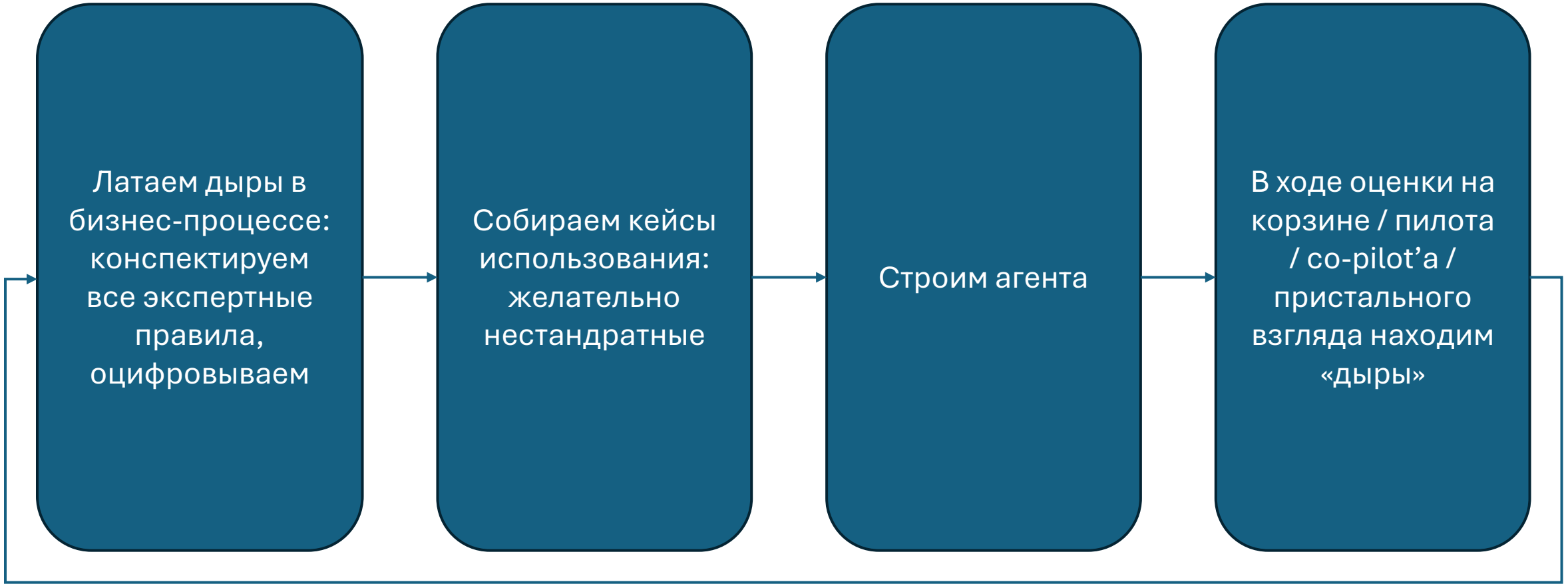
Процесс разработки

Цикл разработки (в т.ч. агента) - ожидание

Цикл разработки (в т.ч. агента) - реальность



Цикл разработки (в т.ч. агента) – реальность спустя время



1. Что выводим?

- MVP – продукт с минимальным функционалом, без защиты и учета краевых кейсов
- MLP – MVP+killer feature
- Полноценный продукт = MLP – недостатки

2. Какие продукты входят в MVP и MLP?

Развитие	Продукт
MVP	Co-Pilot валидора
	Autononom valid (junior)
	Автоверификация С3
MLP	Фичи вне релизного цикла
	Autononom valid (middle)
	Autononom valid (senior)

epic

3. Какие фичи входят в продукты? Какой от них value пользователю? - MVP

task

Продукт	Value для пользователя продукта	Рабочий компонент
Co-Pilot валидора	Повышение удобства использования продукта	Платформа пром
		Платформа тестирования
	Ускорение run, высвобождение времени на альтмод и воспроизведение решения.	Алгоритмизация количественных тестов эндшпиля для ручного запуска
		Автодебют
		Автоформирование Sampler (читает данные)
		Алгоритмизация тестов из дебюта и миттельшпиля данных
		Автоматизация оценки инструкции для асессоров
		Автоформирование Scorer (считает метрику)
		Эндшпиль: формирование списка тестов и обвязки
		Эндшпиль: проведение тестов по fast track
		Автоматизация анализа подхода к моделированию
Autononom valid (junior)	Пройти QG, попасть в ОВМ, корректно измерить качество (редко), найти точки роста для продукта (редко)	Контроль состава, качества и соотвествия формату артефактов для валидации
		Autononom validor
Автоверификация СЗ	Быстрое определение о необходимости валидации (и ее подтверждение), Навигация по корп. налогам	Реализация кода алгоритма в Боте

3. Какие фичи входят в продукты? Какой от них value пользователю? - MLP

task

Продукт	Value для пользователя продукта	Рабочий компонент
Co-Pilot валидора и Autononom valid (junior)	Повышение удобства использования продукта	Настройка логирования
	Ускорение run, высвобождение времени на альтмод и воспроизведение решения.	Эндшпиль: преподготовка данных для тестов, требущих инфреенса И постобработка результатов инфреенса
		Реализация инструмента анализа и автомодификации кода решения
Автоверификация СЗ	Быстрое определение о необходимости валидации (и ее подтверждение), Навигация по корп. налогам	Реализация антифрод-политики на основе LLM
		Помощник по критериям СЗ
Фичи вне релизного цикла	Возможность мониторинга, ускорение разработки новых версий	Co-pilot для DS в построении LLM-as-judge
Autononom valid (middle)	Снижение риска ИОР	Антифрод-политика для агента-валидора
	Более полезно для ВСЗ, Корректно измерить качество, найти точки роста для продукта	Проактивность для агента-валидора – рекомендации, а не только критика
Autononom valid (senior)	Больше пользы для валидатора / поиск нетривиальных ошибок	Советовать и использовать (?) тесты из списка доп. тестов
	Улучшить качество / автономная разработка	Альтмод за счет использования лучших практик моделирования в downstream задаче

4. Для каждой фичи – доп. описание:

- RoadMap:
  - Разработка логики
  - Пилот
  - Дата внедрения
- Ресурсы:
  - GPU
  - Потоки API
  - Люди для разработки / поддержки / асессмента
- Метрика качества фичи (на основании value!):
  - Co-Pilot валидора и Autononom valid (junior):
    - Доступность
    - CSI = #запусков / #негативных отзывов ~ accuracy на тест выборке
    - #запусков / #валидаций в мастер-системе
    - Profit=#валидаций в БМ * зп джуна
  - Autononom valid (middle)
    - Profit = #валидаций в БМ * зп мидла
  - Autononom valid (senior)
    - Profit = #валидаций в БМ * зп сеньора

Описание task

Закупка GPU - sub task

5. Управление рисками (RAID+)

1. **Риски** (RAID):
   - Нельзя запускать сгенерированный код (архитекторы и кибербеза)
2. **Actions** (RIAD):
   - Проведена встреча с предстаивтелями департаментов
   - Можно запускать при условии А и Б
   - Проведена встреча с IT для настройки А
   - Проведена встреча с командой ds для создания видимости Б
   - …
3. **Issues** (RAID):
   - Не пустили в пилот PoC
4. **Decisions** (RAID):
   - Настроить А
   - «Создать» Б
5. **Вероятность рисков**:
   - Высокая (нельзя внедриться без согласия регулятора)
6. **Влияние рисков**:
   - Среднее: 50% функционала будет недостпно
7. **Владелец**:
   1. Product Owener Name

sub task

Задачи, для которых решение на основе LLM будет полезно

- Суммаризация большого пласта текстов
- Выделение именованных сущностей (имена, организации,...)
- Ведение диалога с клиентом (с ролью?)
- QA
- QA+RAG
- Генерация идей
- Генерация неверных вариантов ответов (для составления тестов)

Задачи, для которых решение на основе LLM будет избыточно

- Классификация тона новости
- Рекомендация товара
- Предсказание времени в минутах для выполнения задачи (регрессия)
- Классификация/Регрессия для чего-либо, где фичи представлены табличными данными
- Кривой бизнес-процесс

А как LLM может автоматизировать Ваши бизнес-процессы?

Вопросы

КОНЕЦ ЛЕКЦИИ 02.02