Qwen

Архитектурный Анализ Самоусовершенствующейся Агентной Системы: Двойная Модель, Человеческий Контроль и Замкнутый Цикл Обучения

Фундаментальная Архитектура: Разделение Ответственности между Моделями Размышлений и Кода

Создание сложной, самообучающейся агентной системы, способной самостоятельно разрабатывать и исполнять планы для решения комплексных задач, требует тщательного проектирования архитектуры. Предложенная пользователем концепция двух специализированных моделей — одна для размышлений и планирования, а другая — для создания исполняемого кода — является отражением передовых практик в области разработки автономных агентов [16] [54]. Такой подход, известный как **dual-model** или **dual-process architecture**, позволяет разделить когнитивные и исполнительные функции, что повышает надежность, управляемость и потенциал для глубокого обучения системы. Этот принцип основан на классическом разделении ролей, где один компонент отвечает за стратегию (что делать), а другой — за тактику (как это сделать). В рамках современных **AI**-систем эта парадигма адаптируется для работы нейронных сетей, создавая гибридные архитектуры, которые сочетают силу символьного вывода с гибкостью генеративных моделей [16].

Основой такой системы является модель "Размышлений", которая выполняет функции когнитивного модуля или "мозга" агента [41]. Ее основная задача — не выполнять действия напрямую, а понимать, декомпозировать и планировать их. Когда пользователь задает сложный запрос, например, "найди и сравни все цены на **iPhone** с **13**го по **15**й", модель-планировщик должна сначала интерпретировать этот запрос, выявляя скрытые намерения и ограничения [41].

Затем она разбивает эту макро-задачу на последовательность более простых и выполнимых микро-задач. Например, в случае с анализом цен на смартфоны, план может включать следующие шаги: определить точные названия моделей **(iPhone 13, iPhone 13 Pro, iPhone 13 Pro Max, ..., iPhone 15 Pro Max)**; составить список доверенных веб-сайтов для сбора данных; спроектировать структуру результата (например, **JSON**-схему для таблицы сравнения); сформулировать промпт для модели-генератора кода, который будет писать скрипт для веб-скрапинга; и, наконец, определить логику для анализа полученных данных [69] [73]. Этот процесс декомпозиции является критически важным, поскольку он преобразует абстрактную цель в детальный, машинно-интерпретируемый набор инструкций. Исследования показывают, что эффективное планирование и декомпозиция задач являются ключевыми факторами успеха в сложных агентных системах [34] [73].

Вторым компонентом является модель "Кода", которая выступает в роли исполнительного модуля или "руки" системы [41]. Эта модель не обладает самостоятельными планирующими способностями; ее единственная задача — преобразовать текстовое описание действия, полученное от модели-планировщика, в конкретный, исполняемый код [50]. Например, после того как модель-планировщик сгенерирует промпт, содержащий инструкцию "напиши **Python**-скрипт, который использует библиотеку **BeautifulSoup** для парсинга сайта **X** и извлекает цену товара **Y**", модель-кодогенератор должна создать корректный и безопасный код, который можно запустить в песочнице **(sandbox)** для получения данных [37]. Важнейшим аспектом этого взаимодействия является формализация протокола вызова функций **(Function Calling)** [48] [49]. Это означает, что модель-планировщик не генерирует произвольный текст, а создает структурированный **JSON**-объект, который явно указывает на необходимость вызова определенного инструмента (например, `code_execution_tool`) с конкретными параметрами (например, `"code": "import requests..."`). Такой подход обеспечивает четкое разделение ответственности: **LLM** отвечает только за логику и инструкции, но не за безопасное выполнение кода [51]. Это значительно снижает риски, связанные с выполнением вредоносного или некорректного кода, и делает систему предсказуемой и контролируемой.

Такое разделение ответственности является распространенной практикой в многоагентных системах, где один тип агента **(Planner Agent)** занимается высоким уровнем стратегического мышления, а другой **(Executor Agent)** отвечает за реализацию этих стратегий [34] [61]. Эта дихотомия также находит

отражение в различных фреймворках для построения агентов. Например, в **AutoGen** агенты могут быть настроены на выполнение разных ролей, таких как "**Planner**" и "**Developer**"[91], а в **CrewAI** система команд организуется вокруг агентов со специализированными ролями, например, "**researcher**" (исследователь) и "**writer**" (писатель)[53][88]. В контексте предлагаемой системы, модель-планировщик играет роль "**manager**" или "**orchestrator**", который координирует работу всех компонентов, включая саму модель-генератор кода, если это требуется для решения задачи[54]. Это создает иерархическую структуру, где каждый агент отвечает за свою область компетенции, что упрощает отладку, верификацию и дальнейшее обучение каждого компонента в отдельности. Кроме того, такой подход позволяет использовать разные модели для разных задач: для планирования можно выбрать модель, хорошо знакомую с алгоритмами и логикой, а для генерации кода — модель, специально обученную на большом корпусе программного кода. Это разделение позволяет оптимизировать производительность и качество на каждом этапе рабочего процесса. Таким образом, фундаментальная архитектура, основанная на разделении функций между моделью-планировщиком и моделью-генератором кода, является не просто удобным решением, а ключевой инженерной практикой, обеспечивающей надежность, масштабируемость и потенциал для развития сложной автономной системы.

Компонент	Основная Функция	Задачи и Ответственности	Технологический Протокол
Модель "Размышлений" **(Planner)**	Стратегическое Планирование и Координация	Интерпретация исходного запроса пользователя; декомпозиция сложной задачи на подзадачи; создание детального плана действий; выбор и согласование использования внешних инструментов и агентов; управление циклом выполнения и самокоррекции.	Генерация структурированных **JSON**-планов и промптов для модели-кодогенератора и других инструментов. [34] [41] [73]
Модель "Кода" **(Coder)**	Генерация Исполняемого Кода	Преобразование текстовых инструкций из плана в исполняемый код (например, **Python**, **SQL**); обеспечение соответствия сгенерированного кода заданным требованиям и схемам; генерация безопасного и корректного кода для выполнения в песочнице.	Получение промптов от модели-планировщика и возвращение сгенерированного кода в формате строки. [37] [50] [51]
Инструменты **(Tools)**	Выполнение Внешних Действий	Предоставление доступа к **API**, базам данных, файловой системе и другим внешним сервисам; выполнение кода в безопасной среде **(sandbox)**; предоставление результатов выполнения обратно в систему.	Реализация через стандартные **API**, библиотеки или встроенные функции, вызываемые через протокол **Function Calling**. [48] [49] [67]
Валидатор **(Validator)**	Оценка и Верификация Результатов	Проверка корректности сгенерированного кода (синтаксис, логика, безопасность); оценка качества и полноты результатов, полученных от инструментов; предоставление обратной связи о успешности или неудаче выполнения задачи.	Могут использоваться различные подходы: человек **(human-in-the-loop)**, специализированные **LLM (LLM-as-a-judge)**, автоматизированные системы (статический анализатор кода, тесты). [7] [8] [42]

Принцип Безопасности и Надежности: Интеграция **Human-in-the-Loop (HITL)** на Критических Шагах

Запрос пользователя на обязательное "внешнее подтверждение на критические шаги" является фундаментальным требованием для построения доверенной и безопасной автономной системы. Этот принцип, известный как **Human-in-the-Loop (HITL)**, представляет собой не просто дополнительную функцию, а целостный дизайн-подход, при котором человек активно участвует в работе, надзоре и принятии решений в автоматизированной **AI**-системе [1]. Цель **HITL** — объединить эффективность и скорость автоматизации с точностью, нюансом и этическим рассуждением человека [1]. В контексте предлагаемой системы, **HITL** становится критическим механизмом контроля, который предотвращает необратимые ошибки, гарантирует соответствие регуляторным нормам и обеспечивает возможность для долгосрочного обучения на основе реальных

данных [4] [81]. Без такого механизма система, даже будучи высокоэффективной, остается черным ящиком, что недопустимо в любом критически важном применении.

Существует несколько моделей реализации **HITL**, каждая из которых подходит для разных сценариев и требований к безопасности [3] [79]. Наиболее прямой и часто используемый метод — это**Approval Gates** (гейты-контроль). Этот подход предполагает, что выполнение определенных, заранее определенных действий требует явного одобрения от человека [82]. **Amazon Bedrock Agents** предлагает два встроенных фреймворка для **HITL**: `User Confirmation`, который представляет собой простой бинарный вопрос "да/нет" перед выполнением действия, и `Return of Control` **(ROC)**, более глубокий режим, где система передает полный контекст параметров человеку для редактирования и повторной отправки [2]. Например, в задаче по сравнению цен на **iPhone**, операции, такие как запись данных в базу или изменение файла с результатами, должны быть помечены как высокорисковые и требовать одобрения. Это соответствует паттерну, когда система просит у пользователя подтвердить, что собирается перезаписать существующие данные о ценах на **iPhone 15** [82]. Другой подход —**Confidence-Based Routing**. В этом сценарии система оценивает свою собственную уверенность в правильности решения или действия. Если уровень уверенности падает ниже заданного порога, действие автоматически отправляется на проверку человеку [4] [77]. Это позволяет балансировать между полной автономией и необходимостью человеческого вмешательства, направляя внимание человека на наиболее сложные и неопределенные случаи [79]. Например, если модель-парсер столкнулась с новым форматом веб-страницы, который не встречал ранее, ее уверенность в извлечении данных будет низкой, и она должна будет запросить помощь человека [82].

Еще одним важным паттерном являетсяInteractive Clarification. Он применяется, когда запрос пользователя является неясным, многозначным или содержит противоречия [82]. Вместо того чтобы принимать поспешное решение, система должна самостоятельно идентифицировать эту неопределенность и инициировать диалог для уточнения [3]. Например, если пользователь говорит "сравни цены на **iPhone**", система может спросить: "Имеются в виду все модели **iPhone** или только последние три поколения? Также, какие торговые площадки вас интересуют?". Это не только повышает качество результата, но и демонстрирует системе эвристические способности и готовность к

сотрудничеству [82]. Наконец, существует **Asynchronous Authorization**, который направлен на улучшение пользовательского опыта **(UX)**. Система может отправить запрос на одобрение (например, **push**-уведомление или **email**) и продолжить работу над другими задачами, не дожидаясь ответа. Пользователь может дать свое решение в любое удобное время, что особенно полезно для длительных процессов [5]. Все эти механизмы требуют тщательного проектирования интерфейсов, которые предоставляют человеку достаточно контекста для принятия осознанного решения, включая описание проблемы, потенциальные риски и историю предыдущих действий системы [82].

Интеграция **HITL** имеет далеко идущие последствия для всей архитектуры системы. Во-первых, каждое вмешательство человека должно быть тщательно залогировано для обеспечения прозрачности и возможности аудита [4]. Эти логи служат бесценным источником данных для обучения. Во-вторых, система должна иметь возможность сохранять свое состояние на момент паузы и восстанавливать его после получения ответа от человека, что требует наличия механизма проверки состояния **(checkpointer)** [83]. Ярким примером такой реализации является **LangGraph**, который позволяет легко внедрять точки паузы (`interrupt()`) в графовый рабочий процесс, сохраняя текущее состояние и возобновляя выполнение после ввода человека [83]. В-третьих, наличие **HITL** меняет саму природу управления. Вместо того чтобы полностью делегировать всю власть системе, управление распределяется между человеком и **AI**, где человек сохраняет окончательное слово по критическим вопросам [17]. Это соответствует требованиям многих регуляторных актов, таких как **EU AI Act**, который обязывает системы высокого риска обеспечивать "значительный человеческий надзор", позволяющий людям интерпретировать выводы, вмешиваться и отменять действия системы [81]. Таким образом, **HITL** — это не просто дополнительная кнопка "Подтвердить", а комплексная экосистема, включающая протоколы взаимодействия, механизмы сохранения состояния, системы логирования и регуляторные рамки, которая превращает автономную систему из потенциально опасного инструмента в надежного партнера.

Метод **HITL**	Описание	Пример Применения в Системе	Технологические Реализации
Approval Gates	Автоматическая остановка выполнения действия до получения явного одобрения от человека.	Перед отправкой платежа или удалением файла.	**Amazon Bedrock Agents (User Confirmation, ROC), Zapier ('Request approval' step),** кастомные **UI**-компоненты. [2] [4] [82]
Confidence-Based Routing	Автоматическое направление на проверку действий, для которых модель имеет низкую степень уверенности.	Если модель-парсер не смогла найти цену на **15 Pro Max** на нескольких сайтах, запросить помощь человека.	Калибровка метрик уверенности **(ECE, Brier Score),** настройка порогов для разных категорий риска. [4] [77] [82]
Interactive Clarification	Автоматическое уточнение неоднозначных или противоречивых запросов пользователя.	При запросе "сравни цены на **iPhone**" система спрашивает: "Какие модели и сайты вас интересуют**?**".	**Claude** (реактивные вопросы), **GitHub Copilot** (запрос уточнений), кастомная логика в модели-планировщике. [3] [23] [82]
Asynchronous Authorization	Отправка запроса на одобрение без блокировки основного потока выполнения.	После начала процесса сбора данных, отправить **push-**уведомление пользователю с предложением остановить или продолжить.	**CIBA (Client Initiated Backchannel Authentication), SMS/email** уведомления, **Push-**уведомления. [3] [5]
Batch Review	Автоматическое выполнение в реальном времени с периодической ручной проверкой выборочных результатов.	Пользователь ежедневно просматривает сводку всех успешно завершенных задач, чтобы выявить системные проблемы.	**Zapier ('Collect Data' action), CRM-**системы с возможностью ручной верификации. [4] [79]

Движущая Сила Развития: Создание Замкнутого Цикла Обратной Связи для Непрерывного Улучшения

Отсутствие механизма обратной связи, как справедливо отметил пользователь, делает любую систему, стремящуюся к "самосовершенствованию", статичной и неспособной адаптироваться к новым условиям. Концепция замкнутого цикла обучения **(Closed-Loop Feedback)** является ядром всего проекта, поскольку именно она обеспечивает движущую силу для развития и улучшения системы со временем [8] [43]. Этот цикл представляет собой непрерывный процесс, в котором результаты выполнения задач системой используются для ее последующего обучения и доработки, создавая эффект снежного кома, где каждая новая задача добавляет знаний и опыта [7]. Без этого цикла система будет постоянно повторять те же ошибки, тогда как замкнутый цикл позволяет ей учиться на своих неудачах и постепенно повышать свой уровень компетенций [30].

Элементы замкнутого цикла можно представить в виде последовательности шагов. Первый шаг —Выполнение и Оценка: система берет на себя выполнение задачи, используя свои текущие модели и процедуры. Второй шаг — Валидация: полученный результат не считается окончательным. Он проходит через механизм проверки, который может быть представлен как человек-валидатор или автоматизированная система (например, тестовый набор, статический анализатор кода) [7] [8]. Этот валидатор определяет, был ли результат успешным, частично успешным или полностью неудачным. Третий шаг —Сбор Обратной Связи: вся информация, связанная с результатом, собирается и структурируется. Если результат неудачен, валидатор предоставляет подробное объяснение причины сбоя. Это может быть как "промпт для кода был неверно сформулирован", так и "сгенерированный скрипт выдал ошибку времени выполнения **X**" или "парсер нашел **0** товаров на сайте **Y**" [28] [29]. Четвертый и самый важный шаг —Обновление Модели/Промптов: собранная обратная связь используется для целенаправленного улучшения системы. Данные о неудачах становятся частью обучающей выборки. Эти данные могут использоваться для переобучения **(fine-tuning)** моделей, но гораздо более мощным подходом является использование методов обучения с подкреплением **(Reinforcement Learning, RL)** [13] [25]. Например, система **PREFACE** использует **RL** для обучения малому агенту, который генерирует новые промпты на основе ошибок, сообщаемых формальным верификатором кода, что позволяет повысить процент успешной генерации верифицируемого кода [25]. Аналогично, **HeraldLight** использует качественные оценки критика **(critic)** для обучения агента-генератора **(agent)** через ранжирование, что значительно снижает количество "галлюцинаций" [13].

Формы обратной связи могут быть различными.Человеческая коррекцияявляется наиболее ценной, так как она предоставляет "золотой стандарт" для обучения [77]. Каждое вмешательство человека, будь то исправление кода или отзыв о качестве таблицы сравнения, должно быть детально зафиксировано с указанием причин, чтобы служить учебным материалом для системы [77]. Однако для масштабируемости необходимо внедрятьавтоматизированные валидаторы. Например, после того как модель-код генерирует скрипт для веб-скрапинга, его можно автоматически прогнать через статический анализатор кода **(SonarQube)** для поиска потенциальных уязвимостей и ошибок [42]. Затем скрипт можно запустить в изолированной среде **(sandbox)**, и результат его работы (например, **JSON**-файл с данными) можно проверить на соответствие заранее определенной **Pydantic**-схеме, чтобы

убедиться в его структурной корректности[98]. Для еще более высокого уровня надежности можно использоватьформальную верификацию. **Frameworks like ASTROGATOR** используют формальную верификацию **Ansible**-скриптов, чтобы гарантировать, что сгенерированный код выполняет именно те действия, которые были запрошены, а не допускает побочных эффектов[24]. **PropertyGPT** использует компилятор и статический анализ в качестве внешнего оракула для итеративного улучшения сгенерированных свойств для смарт-контрактов, прежде чем они будут проверены формальным доказателем[27].

Ключевым аспектом эффективного обучения является структурирование обратной связи. Система должна уметь отличать причины сбоев и получать детализированные метаданные. Вместо общего сигнала "не удалось", система должна получить информацию типа: "Ошибка типа '**AttributeError**' в строке **15** скрипта, потому что переменная '**price_element**' равна **None**", или "Данные, извлеченные со страницы, содержат **50** элементов вместо ожидаемых **100**, что указывает на проблему с **CSS**-селектором". Такая детализация позволяет системе проводить целенаправленный поиск и исправление ошибок[28]. Например, если модель-генератор кода постоянно создает скрипты с определенным типом ошибок, система может выявить эту закономерность и скорректировать свой промпт, чтобы сделать акцент на избегании этой ошибки в будущем. Механизмы, такие как **Metaprompt Agent** в **Self-Evolving Agent Architecture**, специально созданы для анализа отказов и генерации точных, технически обоснованных инструкций для обновления основных промптов[28]. Важно отметить, что не только неудачи, но и успешные выполнения задач также являются источником информации. Они служат положительным сигналом для обучения и помогают системе лучше понять, какие стратегии и промпты работают хорошо. Таким образом, замкнутый цикл обучения превращает каждое взаимодействие с пользователем в возможность для роста, позволяя системе не просто выполнять задачи, а постоянно совершенствовать свой собственный способ их выполнения.

Технологическая Реализация: Выбор Оркестраторов, Протоколов и Валидаторов

Выбор правильного технологического стека и архитектурных паттернов имеет решающее значение для реализации амбициозной системы, описанной в

запросе. Необходимо выбрать инструменты, которые позволят эффективно оркестрировать взаимодействие между двумя моделями, стандартизировать их коммуникацию с внешними инструментами и обеспечить надежную проверку результатов. Современные фреймворки для создания агентов предлагают различные подходы, и выбор зависит от требований к детерминизму, гибкости и сложности рабочего процесса.

Для оркестрации рабочих процессов в данной системе наиболее подходящим инструментом является**LangGraph.** Его основное преимущество заключается в том, что он моделирует агентские рабочие процессы как графы состояний, что идеально подходит для последовательных и итеративных операций, характерных для задачи пользователя[89] [91]. В **LangGraph** каждый агент или шаг (например, "**planning_node**", "**coding_node**") является узлом, а переходы между ними — ребрами. Это позволяет создавать сложные, циклические рабочие процессы, необходимые для самокоррекции. Например, после того как модель-код генерирует скрипт, он может быть передан для проверки, и если валидатор сообщает об ошибке, выполнение может вернуться к узлу генерации кода с новой, скорректированной инструкцией. **LangGraph** имеет встроенную поддержку состояния **(state)**, которое является общим для всех узлов и хранит всю необходимую информацию (историю сообщений, текущий шаг, результаты выполнения), что упрощает управление сложным контекстом[89]. Более того, **LangGraph** предоставляет встроенные механизмы для реализации **Human-in-the-Loop**, позволяя создавать специальные узлы, которые явно останавливают выполнение и ждут ввода от человека, сохраняя при этом текущее состояние для возобновления работы[83] [104]. Примеры использования **LangGraph** для **RAG**-систем, где происходит итеративная проверка на "галлюцинации" и запрос на дополнительную информацию, демонстрируют его применимость к задаче проверки качества данных[99]. Хотя другие фреймворки, такие как **AutoGen** от **Microsoft** и **CrewAI**, также мощны, они имеют свои особенности. **AutoGen** лучше подходит для сценариев, требующих свободного диалогового взаимодействия между агентами, где они могут спорить, соглашаться и обсуждать решение[89] [91]. **CrewAI**, в свою очередь, ориентирован на создание "команд" агентов с четко определенными ролями и ограниченной областью ответственности, что также может быть полезно для разделения задач в системе[53] [87].

Центральным элементом коммуникации между всеми компонентами системы должен стать формализованный протокол**Function Calling** (или **Tool Calling**)[48] [49]. Этот протокол определяет, как **LLM** взаимодействует с внешними инструментами. Вместо того чтобы генерировать произвольный текст, **LLM**

возвращает структурированный JSON-объект, который указывает, какой инструмент нужно вызвать и с какими аргументами [51]. Это разделяет логику принятия решения (которую выполняет LLM) и фактическое выполнение действия (которое выполняется внешним инструментом). Для дальнейшей стандартизации взаимодействия рекомендуется использовать открытый стандартModel Context Protocol (MCP) [48]. MCP предназначен для создания универсальных интерфейсов между AI-моделями и данными/инструментами, решая проблему "N×M" количества кастомных соединителей [48]. Использование MCP позволит легко интегрировать различные инструменты и, что особенно важно, обеспечить возможность для внешней системы (валидатора) перехватывать и проверять вызовы функций до их фактического выполнения [47]. Это создает надежный барьер безопасности и контроля.

Наконец, необходимо продумать архитектуру валидатора. Запрос на "отдельного валидатора" является критически важным для обеспечения надежности и предотвращения "галлюцинаций" LLM. Валидатор не должен быть просто еще одной LLM, так как это не решает проблему самосовершенствования на основе ошибок. Вместо этого следует использовать многоуровневую систему валидации. 1.Автоматизированный валидатор (первый уровень):Этот уровень предназначен для быстрой проверки промежуточных результатов. Например, после того как модель-код сгенерировала скрипт, его можно проверить с помощью:Статического анализатора кода (SAST):Инструменты, такие как SonarQube или ESLint, могут проверить код на соответствие стандартам кодирования, поиск потенциальных уязвимостей и ошибок [42].Синтаксической проверки:Простая проверка на корректность синтаксиса языка (например, для Python).Проверки структуры данных:Если инструмент должен вернуть данные в определенном формате (например, JSON с определенной Pydantic-схемой), эта проверка может быть автоматизирована [98]. 2.Исполнительный валидатор (второй уровень):Этот уровень выполняет сгенерированный код в безопасной, изолированной среде — "песочнице" (sandbox). Это позволяет проверить код на предмет безопасности (например, попыток доступа к файловой системе или сети) и на корректность его работы в реальных условиях [59]. Результаты выполнения (stdout, stderr, выходные файлы) собираются для дальнейшей проверки. 3.Логический валидатор (третий уровень):Этот уровень проверяет, действительно ли результат, полученный от исполнителя, решает поставленную задачу. Здесь может быть задействован человек или более мощная LLM, которая выступает в роли эксперта. Например, человек может проверить, что сгенерированная таблица сравнения цен содержит актуальные данные и нет никаких очевидных ошибок. 4.Формальный

валидатор (четвертый уровень, для критически важных частей):*Для самых важных частей системы, например, для алгоритмов расчета стоимости доставки или финансовых расчетов, можно использовать подходы формальной верификации, где код проверяется на соответствие формальным спецификациям, что гарантирует его математическую корректность [24] [25].

Такая многоуровневая архитектура валидации обеспечивает баланс между скоростью, автоматизацией и надежностью, позволяя снизить нагрузку на человека, но сохраняя человеческий контроль над наиболее важными и сложными аспектами работы системы.

Управление и Исправление Ошибок: Распределение Компетенций и Роль Валидатора

Вопрос о том, кто управляет, кто что исправляет и как система учится на ошибках, является центральным для понимания динамики работы самообучающейся агентной системы. Ответ на него лежит в тщательном распределении компетенций между различными компонентами системы: моделью-планировщиком, моделью-генератором кода, валидатором и механизмом обратной связи. Каждый из этих элементов играет свою уникальную и незаменимую роль в достижении конечной цели — точного и надежного выполнения задачи.Управление процессомв такой системе осуществляется преимущественномоделью "Размышлений" **(Planner)**. Она несет ответственность за весь жизненный цикл задачи: от первоначального понимания запроса до синтеза конечного результата. Модель-планировщик управляет последовательностью действий, выбирая, какую модель-генератор кода использовать, какие инструменты вызвать и в каком порядке. Она также управляет циклом самокоррекции: если валидатор сообщает об ошибке, модель-планировщик должна проанализировать сообщение об ошибке, понять его причину и сгенерировать новый, скорректированный план действий для модели-кодогенератора. Эта роль аналогична роли менеджера проекта, который отслеживает прогресс, выявляет проблемы и перепланирует ресурсы для достижения цели [69]. В некоторых архитектурах, таких как **SAGE**, эта роль может быть разделена между несколькими агентами, где **Meta agent** отвечает за стратегическое планирование, **Executor** — за тактическое исполнение, а **Evaluator** — за краткосрочную коррекцию [61] [63].Исправление ошибок— это

процесс, который начинается с момента, когдавалидаторвыявляет несоответствие между ожидаемым и фактическим результатом. Роль валидатора абсолютно критична, и ее нельзя недооценивать. Как было отмечено ранее, **LLM** демонстрируют значительную склонность к "**intrinsic self-correction failure**", то есть систематически предпочитают свою первоначальную, даже неверную, генерацию [23]. Они могут распознавать свои же ответы с высокой точностью **(**более **70%)** и им доверять, игнорируя очевидные ошибки, что делает внутреннюю самопроверку почти бесполезной [23]. Поэтомувнешняя, независимая валидация является абсолютной необходимостью. Исправление может происходить несколькими путями: **1.**Человеческое исправление**:**В случаях, когда система не может самостоятельно исправить ошибку или когда требуется человеческий опыт, экспертное суждение или учет контекста, вмешивается человек. Это может быть как простое одобрение исправленного варианта, так и ручное исправление кода или данных. Человеческое вмешательство также является основным источником данных для обучения системы через замкнутый цикл обратной связи [77]**. 2.**Автоматическое исправление**:**Система может быть спроектирована для автоматического исправления некоторых типов ошибок. Например, если валидатор сообщает, что сгенерированный **SQL-**запрос содержит синтаксическую ошибку, модель-планировщик может сгенерировать новый промпт для модели-кодогенератора с указанием на ошибку, и тот может попытаться исправить ее. Это основа для итеративных циклов самокоррекции, таких как **Reflexion** или **ToRA** [30] [66]**. 3.**Исправление через переобучение**:**Длинно-временное исправление ошибок происходит через механизм обратной связи. Собранные данные об ошибках используются для переобучения или дообучения **(fine-tuning)** моделей-генераторов, чтобы они в будущем избегали совершения тех же ошибок [13] [25]. Например, если система постоянно ошибается при парсинге данных с сайта **Z,** а человек исправляет эти ошибки, то эти исправленные примеры могут быть добавлены в обучающую выборку для модели-генератора кода.

Таким образом, распределение компетенций можно описать следующим образом:

Компонент	Основная Роль в Управлении и Исправлении	Как Участвует в Процессе Самосовершенствования
Модель "Размышлений" **(Planner)**	Главный Управляющий:Определяет цель, создает план, координирует работу, управляет циклом самокоррекции.	Анализирует сообщения об ошибках от валидатора и генерирует новые, исправленные планы. Участвует в формировании данных для обучения. [34] [41] [73]
Модель "Кода" **(Coder)**	Исполнитель:Выполняет действия, описанные в плане, генерируя код.	Не участвует в исправлении напрямую, но ее выходные данные (код) являются объектом для проверки валидатором. [50] [51]
Валидатор	Контролер и Диагност:Проверяет корректность кода и результатов его выполнения. Выявляет ошибки и предоставляет детальное описание причин.	Его выходные данные (оценки и сообщения об ошибках) являются основным источником обратной связи для всей системы. [7] [8] [42]
Механизм Обратной Связи **(Feedback Loop)**	Директор и Учитель:Собирает данные обо всех взаимодействиях, анализирует неудачи и целенаправленно обновляет модели и промпты.	Это ядро системы, которое использует данные о неудачах для обучения и улучшения моделей-генераторов и планировщиков. [13] [25] [43]
Человек **(Human-in-the-Loop)**	Финальный Окончательный Контроллер:Принимает окончательное решение по критическим шагам, исправляет сложные ошибки и предоставляет золотой стандарт для обучения.	Его одобрение и исправления являются высшим приоритетом и служат "золотым стандартом" для обучения моделей. [1] [4] [77]

В заключение, управление и исправление ошибок в такой системе представляют собой сложный, многоуровневый процесс, в котором каждый компонент выполняет свою специфическую роль. Модель-планировщик управляет процессом, валидатор выступает в роли незаменимого контролера, а человек сохраняет финальное слово по самым важным решениям. Именно совместная работа этих компонентов, подпитываемая информацией из замкнутого цикла обратной связи, позволяет системе не просто выполнять задачи, а постоянно развиваться, становиться умнее и надежнее с каждым новым вызовом.

Синтез и Практические Рекомендации: От Теории к Реальной Реализации

На основе всестороннего анализа предоставленных материалов и запроса пользователя, можно сформулировать ряд ключевых рекомендаций и синтезировать практическую картину реализации предложенной самообучающейся агентной системы. Успех проекта будет зависеть не столько от выбора конкретных моделей, сколько от тщательного проектирования

архитектурных паттернов, которые обеспечивают надежность, безопасность через **Human-in-the-Loop (HITL)** и эффективное обучение через замкнутый цикл обратной связи.Первоочередной рекомендацией является принятие архитектуры с разделением ответственности **(Dual-Model Architecture)**. Систему следует разделить на два специализированных агента:**Planner**(для размышлений и планирования) и**Coder**(для генерации кода) [16] [54]. Такое разделение не только упрощает управление и отладку, но и является лучшей практикой для повышения надежности. Модель-планировщик будет отвечать за логику, стратегию и управление циклом выполнения, в то время как модель-генератор кода будет выполнять только одну функцию — преобразование текстовых инструкций в исполняемый код [34]. Это четкое разделение компетенций снижает риск ошибок и позволяет оптимизировать каждую модель под свою специфику.В качестве основы для оркестрации рабочих процессов настоятельно рекомендуется использовать фреймворк **LangGraph**. Его способность моделировать состояния, управлять циклами и встраивать точки паузы для **HITL** идеально соответствует требованиям задачи [89] [91]. **LangGraph** позволит реализовать сложный, итеративный процесс, где модель-генератор кода может многократно перезапускаться с новыми инструкциями, пока валидатор не подтвердит корректность результата. Взаимодействие между всеми компонентами системы должно быть стандартизировано с помощью протокола**Function Calling**, а для дальнейшей унификации интеграции с инструментами и валидаторами следует рассмотреть использование открытого стандарта**MCP (Model Context Protocol)** [47] [48].Реализация системы **HITL** должна быть многоуровневой и гибкой. Вместо одного общего правила "спрашивать человека" следует применять несколько паттернов. Для критически важных операций, таких как изменение данных или отправка уведомлений, следует использовать**Approval Gates** [82]. Для задач, где модель демонстрирует низкую уверенность, следует внедрить**Confidence-Based Routing**для автоматической отправки на проверку [4]. Кроме того, система должна быть способна инициировать**Interactive Clarification**, чтобы уточнять неоднозначные запросы пользователя, прежде чем приступать к выполнению [82]. Каждое вмешательство человека должно быть тщательно зафиксировано в логах, так как это является основным источником данных для обучения [4] [77].Ключевым элементом, обеспечивающим развитие системы, является замкнутый цикл обратной связи. Необходимо спроектировать систему таким образом, чтобы каждый результат выполнения задачи, особенно неудачный, шел на вход для обучения. Собранные данные об ошибках должны быть детализированы, чтобы позволить системе целенаправленно их исправлять. Эти данные могут использоваться для дообучения моделей или, что более эффективно, для обучения с

подкреплением (RL), где система получает обратную связь от валидатора в виде наград или штрафов, что позволяет ей научиться выбирать более успешные стратегии [13] [25]. Этот цикл превращает каждое взаимодействие с пользователем в возможность для роста и улучшения.Наконец, роль валидатора должна быть четко определена и расширена за пределы одного LLM. Запрос на "отдельного валидатора" является одним из самых важных пунктов. Валидатор должен быть многоуровневой системой. Для повышения эффективности следует внедритьавтоматизированных валидаторовна промежуточных шагах, таких как статический анализ кода, проверка структуры данных и выполнение unit-тестов [42] [98]. Это позволит быстро отсеивать очевидные ошибки и снизить нагрузку на человека. Человек должен оставаться финальным контролером, особенно для сложных и неоднозначных задач, где требуется экспертное суждение. Только такой гибридный подход, сочетающий автоматизацию и человеческий контроль, позволит создать систему, которая одновременно и быстра, и надежна.

В заключение, предложенная пользователем система является передовым примером современной AI-агента. Ее реализация потребует значительных усилий в области системного проектирования, но следование вышеописанным принципам позволит создать мощную, безопасную и способную к непрерывному развитию платформу, способную решать сложные, многокомпонентные задачи автономно.

Справка

1. **What Is Human In The Loop (HITL)?** https://www.ibm.com/think/topics/human-in-the-loop
2. **Implement human-in-the-loop confirmation with ...** https://aws.amazon.com/blogs/machine-learning/implement-human-in-the-loop-confirmation-with-amazon-bedrock-agents/
3. **Why AI still needs you: Exploring Human-in-the-Loop ...** https://workos.com/blog/why-ai-still-needs-you-exploring-human-in-the-loop-systems
4. **Human-in-the-loop in AI workflows: Meaning and patterns** https://zapier.com/blog/human-in-the-loop/

5. **Secure “Human in the Loop” Interactions for AI Agents** https://auth0.com/blog/secure-human-in-the-loop-interactions-for-ai-agents/
6. **Humans in the Loop: The Design of Interactive AI Systems** https://hai.stanford.edu/news/humans-loop-design-interactive-ai-systems
7. **Human-in-the-Loop AI: Why It Matters in the Era of GenAI** https://www.tredence.com/blog/hitl-human-in-the-loop
8. **Why Human in the Loop is Critical for Reliable AI Solutions** https://www.openxcell.com/blog/human-in-the-loop/
9. **From Human-in-the-loop to Human-in-power - PMC** https://pmc.ncbi.nlm.nih.gov/articles/PMC11384285/
10. **Human-In-The-Loop Systems** https://thedecisionlab.com/reference-guide/computer-science/human-in-the-loop-systems
11. **Designing a neuro-symbolic dual-model architecture for ...** https://www.nature.com/articles/s41598-025-27076-9
12. **A Dual-Model Architecture with Grouping-Attention-Fusion ...** https://www.mdpi.com/2072-4292/13/3/433
13. **A Dual Large Language Models Architecture with Herald ...** https://arxiv.org/html/2511.00136v1
14. **(PDF) Designing a neuro-symbolic dual-model architecture ...** https://www.researchgate.net/publication/398084922_Designing_a_neuro-symbolic_dual-model_architecture_for_explainable_and_resilient_intrusion_detection_in_IoT_networks
15. **Dual-process theories of thought as potential architectures ...** https://www.frontiersin.org/journals/cognition/articles/10.3389/fcogn.2024.1356941/pdf
16. **Agentic AI: A Comprehensive Survey of Architectures ...** https://arxiv.org/html/2510.25445v1
17. **A Conceptual Framework for AI-based Decision Systems in ...** https://arxiv.org/html/2504.16133v1
18. **2025 AI Safety Index** https://futureoflife.org/ai-safety-index-summer-2025/
19. **Third-Party Assessments** https://www.frontiermodelforum.org/technical-reports/third-party-assessments/
20. **AI Safety for Field Operations: Best Practices and Protocols** https://wezom.com/blog/ai-safety-for-field-operations-best-practices-and-protocols
21. **AI Risks in Healthcare Incident Response Policies** https://censinet.com/perspectives/ai-risks-in-healthcare-incident-response-policies

22. **How AI challenges the medical device regulation: patient ...** https://academic.oup.com/jlb/advance-article/doi/10.1093/jlb/lsae007/7642716
23. **Why AI Systems Can't Catch Their Own Mistakes - Nova Spivack** https://www.novaspivack.com/technology/ai-technology/why-ai-systems-cant-catch-their-own-mistakes-and-what-to-do-about-it
24. **Towards Formal Verification of LLM-Generated Code from ...** https://arxiv.org/abs/2507.13290
25. **A Reinforcement Learning Framework for Code Verification ...** https://dl.acm.org/doi/10.1145/3716368.3735300
26. **Generating and Structuring Proofs for Formal Verification** https://openreview.net/forum?id=QeoJtLyRsn
27. **PropertyGPT: LLM-driven Formal Verification of Smart ...** https://www.ndss-symposium.org/ndss-paper/propertygpt-llm-driven-formal-verification-of-smart-contracts-through-retrieval-augmented-property-generation/
28. **Self-Evolving Agents - A Cookbook for Autonomous ...** https://cookbook.openai.com/examples/partners/self_evolving_agents/autonomous_agent_retraining
29. **AgentEvolver: Towards Efficient Self-Evolving Agent System** https://arxiv.org/html/2511.10395v1
30. **The Dawn of Self-Evolving AI: How Agents Are Learning to ...** https://www.aiworldtoday.net/p/the-dawn-of-self-evolving-ai-agents
31. **Validating multi-agent AI systems** https://www.pwc.com/us/en/services/audit-assurance/library/validating-multi-agent-ai-systems.html
32. **Unlocking Agentic AI: Risks & Governance | EY - Canada** https://www.ey.com/en_ca/insights/assurance/technology-risk/unlocking-the-potential-of-agentic-ai
33. **HyDRA: A Hybrid-Driven Reasoning Architecture for ...** https://arxiv.org/html/2507.15917v1
34. **AgentX: Towards Orchestrating Robust Agentic Workflow ...** https://arxiv.org/html/2509.07595v1
35. **Adaptive Tool Generation with Models as ...** https://arxiv.org/html/2510.06825v1
36. **Designing LLM-based Multi-Agent Systems for Software ...** https://arxiv.org/html/2511.08475v1
37. **A Survey on Code-Enhanced Reasoning and ...** https://arxiv.org/html/2502.19411v1
38. **A closed-loop architecture with knowledge-of-results ...** https://www.sciencedirect.com/science/article/abs/pii/S095070512501086X

39. **Closed-Loop Development: How AI Agents Build Software ...** https://medium.com/@alexzanfir/closed-loop-development-how-ai-agents-build-software-while-you-sleep-6df42cd05a85
40. **Closed-Loop Intelligence: A Design Pattern for Machine ...** https://learn.microsoft.com/en-us/archive/msdn-magazine/2019/april/machine-learning-closed-loop-intelligence-a-design-pattern-for-machine-learning
41. **Building Agentic AI Architectures: Blueprint for Autonomous ...** https://www.tredence.com/blog/agentic-ai-architectures
42. **Architectural Blueprint for the Closed-Loop Autonomous ...** https://www.linkedin.com/pulse/architectural-blueprint-closed-loop-autonomous-agent-aad-smeyatsky-achcf
43. **Build Feedback Loops in Agentic AI for Digital Growth** https://www.amplework.com/blog/build-feedback-loops-agentic-ai-continuous-transformation/
44. **The Autonomous Enterprise: Principles of Autonomous Systems** https://kalypso.com/viewpoints/entry/the-autonomous-enterprise-part-1
45. **A Structural Output Evaluation of Multimodal LLMs** https://arxiv.org/html/2511.21750v1
46. **Who Validates the Validators? Aligning LLM-Assisted ...** https://people.eecs.berkeley.edu/~bjoern/papers/shankar-validators-uist2024.pdf
47. **Unified Tool Integration for LLMs: A Protocol-Agnostic ...** https://arxiv.org/html/2508.02979v1
48. **Understanding Function Calling: The Bridge to Agentic AI** https://fireworks.ai/blog/function-calling
49. **Function calling - OpenAI API** https://platform.openai.com/docs/guides/function-calling
50. **Tools & Function Calling in LLMs and AI Agents** https://medium.com/@sahin.samia/tools-function-calling-in-llms-and-ai-agents-a-hands-on-guide-3a19f2f21954
51. **Function Calling with LLMs** https://www.promptingguide.ai/applications/function_calling
52. **How function calling and tool use work in advanced AI models** https://www.datastudios.org/post/how-function-calling-and-tool-use-work-in-advanced-ai-models
53. **Demystifying AI Agents: Frameworks and Comparative ...** https://www.linkedin.com/pulse/demystifying-ai-agents-frameworks-comparative-jin-nch4e

54. **Building Effective AI Agents** https://www.anthropic.com/research/building-effective-agents
55. **How do AI agents evaluate the outcomes of their actions?** https://milvus.io/ai-quick-reference/how-do-ai-agents-evaluate-the-outcomes-of-their-actions
56. **The Power of AI Feedback Loop: Learning From Mistakes** https://irisagent.com/blog/the-power-of-feedback-loops-in-ai-learning-from-mistakes/
57. **Agent0-VL: Exploring Self-Evolving Agent for Tool- ...** https://arxiv.org/html/2511.19900v1
58. **SAGE: Self-evolving Agents with Reflective and Memory- ...** https://arxiv.org/html/2409.00872v2
59. **HiVA: Self-organized Hierarchical Variable Agent via Goal ...** https://arxiv.org/html/2509.00189v1
60. **Self-Evolving Multi-Agent Collaboration Networks for ...** https://arxiv.org/html/2410.16946v1
61. **Self-evolving Agents with reflective and memory ...** https://arxiv.org/html/2409.00872v1
62. **An Automated Framework for Evolving Agentic Workflows** https://arxiv.org/pdf/2507.03616
63. **HealthFlow: A Self-Evolving AI Agent with Meta Planning ...** https://arxiv.org/pdf/2508.02621
64. **Average Reward Reinforcement Learning for Omega ...** https://arxiv.org/abs/2505.15693
65. **A Survey on Autonomous Scientific Discovery** https://arxiv.org/html/2508.14111v1
66. **Awesome ICLR 2024 LLM Papers Collection** https://github.com/azminewasi/Awesome-LLMs-ICLR-24
67. **LLM-Based Agents for Tool Learning: A Survey** https://link.springer.com/article/10.1007/s41019-025-00296-9
68. **Survey: Tool Learning with Large Language Models** https://github.com/quchangle1/LLM-Tool-Survey
69. **AI Agent Orchestration Patterns - Azure Architecture Center** https://learn.microsoft.com/en-us/azure/architecture/ai-ml/guide/ai-agent-design-patterns
70. **Top 10 AI Agent Frameworks for Building Autonomous ...** https://www.kubiya.ai/blog/top-10-ai-agent-frameworks-for-building-autonomous-workflows-in-2025

71. **Top 9 AI Agent Frameworks as of November 2025** https://www.shakudo.io/blog/top-9-ai-agent-frameworks
72. **40+ Agentic AI Use Cases with Real-life Examples** https://research.aimultiple.com/agentic-ai/
73. **Autonomous Data Agents: A New Opportunity for Smart Data** https://arxiv.org/html/2509.18710v1
74. **Real-world gen AI use cases from the world's leading ...** https://cloud.google.com/transform/101-real-world-generative-ai-use-cases-from-industry-leaders
75. **Optimizing Agent Planning for Security and Autonomy** https://openreview.net/forum?id=g0aVCDY3gS
76. **A Survey on the Feedback Mechanism of LLM-based AI ...** https://www.ijcai.org/proceedings/2025/1175.pdf
77. **The Pragmatic Guide to Human in the Loop AI - Arafat Tehsin** https://arafattehsin.com/the-pragmatic-guide-to-human-in-the-loop-ai/
78. **Agents with Human in the Loop : Everything You Need to ...** https://dev.to/camelai/agents-with-human-in-the-loop-everything-you-need-to-know-3fo5
79. **AI Agents with Human-in-the-Loop: Safer & Reliable AI** https://www.creatio.com/glossary/human-in-the-loop-ai-agents
80. **Towards Effective Human-in-the-Loop Assistive AI Agents** https://arxiv.org/html/2507.18374v1
81. **Agent vs Human-in-the-Loop in 2025** https://skywork.ai/blog/agent-vs-human-in-the-loop-2025-comparison/
82. **Human-in-the-Loop: Balancing AI Autonomy and Human Control** https://hopx.ai/blog/ai-agents/human-in-the-loop-ai-agents
83. **LangGraph 201: Adding Human Oversight to Your Deep ...** https://towardsdatascience.com/langgraph-201-adding-human-oversight-to-your-deep-research-agent/
84. **Convergence of a Human-in-the-Loop Policy-Gradient ...** https://www.semanticscholar.org/paper/737da08b32cf9b7a448c29c8d192bb47480ad48c
85. **Top Browser Infrastructure Companies by Funding (2025)** https://www.joinmassive.com/blog/browser-infrastructure-companies-funding
86. **25 Best RFP Tools for 2025 | AI-Powered Comparison Guide** https://deeprfp.com/blog/best-rfp-tools-2025-ai-comparison/
87. **Introduction** https://docs.crewai.com/en/introduction

88. **Building Multi-Agent Systems With CrewAI** https://www.firecrawl.dev/blog/crewai-multi-agent-systems-tutorial
89. **LangGraph vs AutoGen vs CrewAI: Complete AI Agent ...** https://latenode.com/blog/platform-comparisons-alternatives/automation-platform-comparisons/langgraph-vs-autogen-vs-crewai-complete-ai-agent-framework-comparison-architecture-analysis-2025
90. **Top AI Agent Frameworks of 2025: LangChain, CrewAI & ...** https://medium.com/@lambert.watts.809/top-10-best-ai-frameworks-for-building-ai-agents-in-2025-137fafb37a46
91. **LangGraph vs AutoGen vs CrewAI: Best Multi-Agent Tool?** https://www.amplework.com/blog/langgraph-vs-autogen-vs-crewai-multi-agent-framework/
92. **Built a Price Monitoring Agent with ScrapeGraphAI and more** https://www.linkedin.com/posts/arindam2004_ive-been-experimenting-with-multi-agent-activity-7370096893609095168-2tc5
93. **What is crewAI?** https://www.ibm.com/think/topics/crew-ai
94. **Benchmarking Agentic AI Frameworks in Analytics Workflows** https://research.aimultiple.com/agentic-analytics/
95. **LLMOps in Production: 457 Case Studies of What Actually ...** https://www.zenml.io/blog/llmops-in-production-457-case-studies-of-what-actually-works
96. **langchain-ai/langgraph: Build resilient language agents as ...** https://github.com/langchain-ai/langgraph
97. **Agentically scrape the web with Firecrawl & LangGraph ...** https://www.youtube.com/watch?v=vSz5-KeRyHs
98. **ScrapeGraphAI/langchain-scrapegraph** https://github.com/ScrapeGraphAI/langchain-scrapegraph
99. **Building RAG Research Multi-Agent with LangGraph** https://ai.gopubby.com/building-rag-research-multi-agent-with-langgraph-1bd47acac69f
100. **Developing a Large Scale Attribute Extractor for E-Commerce** https://medium.com/thedeephub/developing-a-large-scale-attribute-extractor-for-e-commerce-3efc4d7004e
101. **Scaling Catalog Attribute Extraction with Multi-modal LLMs** https://tech.instacart.com/multi-modal-catalog-attribute-extraction-platform-at-instacart-b9228754a527
102. **Building DoorDash's product knowledge graph with large ...** https://careersatdoordash.com/blog/building-doordashs-product-knowledge-graph-with-large-language-models/

103. **EP 1 • E-commerce Product Copy with LangGraph** https://www.youtube.com/watch?v=5EkVyGBl6tQ

104. **Building an AI-Powered Agent with LLMs and LangGraph** https://medium.com/@venku.buragadda/building-an-agent-using-llm-491680706a90

105. **Build Product Knowledge Graph using LLM** https://pub.towardsai.net/build-product-knowledge-graph-using-llm-a0d0354156ac