Анализ и классификация подходов к развитию LLM-Агентов

Р. Р. Фаткиева[1], А. Е. Сычев[2]

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им. В.И. Ульянова (Ленина)

[1]rikki2@yandex.ru, [2]the220th@gmail.com

***Аннотация*. Современные агентные системы на основе больших языковых моделей требуют механизмов адаптации к новым задачам, данным и средам. В данной работе предлагается унифицированная таксономия подходов к эволюции агентов, разделённая на динамические и статические методы. Проведён анализ архитектуры LLM-агентов, включающей ядро, системный промпт, память и инструменты. К статическим подходам отнесены методы изменения параметров модели и компонентов агента. Динамические подходы включают адаптацию во время выполнения за счёт обновления знаний, управления памятью и взаимодействия с внешней средой. Работа направлена на систематизацию существующих методов и может служить основой для дальнейших исследований и практического применения LLM-агентов.**

Ключевые слова: искусственный интеллект; агентные системы; LLM-агент; систематизация

I. Введение

В последние годы наблюдается стремительное развитие агентных систем в контексте использования больших языковых моделей (large language model, LLM). Такие системы представляют собой новый этап эволюции искусственного интеллекта, позволяя создавать автономные, адаптивные и гибкие решения для широкого круга задач. В отличие от классических агентных систем, основанных на строгих правилах или обучении с подкреплением, LLM-агенты обладают способностью к обобщению, контекстному пониманию, динамическому принятию решений и взаимодействию между собой. В частности, выделяются одноагентные (one agent system) и мультиагентные (multi-agent system) системы. Многоагентной системе присущи кооперация нескольких агентов, способных распределять задачи, обмениваться информацией и совместно достигать целей.

Однако ключевым аспектом развития подобных систем является их способность к эволюции в процессе работы. Под эволюцией понимается совокупность методов и механизмов, позволяющих агенту адаптироваться к новым условиям, улучшать свои стратегии и повышать эффективность выполнения задач. В данной статье рассматриваются основные подходы к эволюции агентных систем, а также существующие проблемы и перспективы развития.

Агентные системы на базе больших языковых моделей обычно состоят из набора взаимосвязанных компонентов (см. рисунок 1): ядро (core LLM), системный промпт (system prompt), память (memory), инструменты (tools). Ядро — это базовая архитектура и набор параметров нейросети. Эта модель может функционировать в разных режимах: как унимодальная система (обрабатывающая один тип данных, например текст) или как мультимодальная (способная работать с текстом, изображениями, аудио и другими форматами) [1]. Важно отметить, что современная тенденция предполагает, что большие языковые модели, которые агентные системы будут использовать как ядро, должны обладать способностью вызова инструментов (tool-use). Системный промпт — один из ключевых механизмов управления поведением агента. Здесь задаются такие параметры, как «социальная роль» (social place), идентичность (identity), цели (goals), ограничения (constraints), а также политика использования инструментов (tool-use policy) [2]. Именно через системный промпт формируется контекст, в рамках которого модель интерпретирует запросы и принимает решения. Под социальной ролью (social place) понимается кем агент должен себя представлять: ассистент, эксперт, консультант, редактор и т. п. Идентичность (identity) определяет стабильные поведенческие ожидания: тон, этику и тому подобное. Целью (goals) задаётся основное назначение агента, а ограничения (constraints) показывают, что агенту делать нельзя. Политика использования инструментов (tool-use policy) предписывает агенту, как именно вызывать инструменты, а также предоставляет список таких инструментов. Память (memory) отражает способность агента учитывать текущий контекст и накопленные знания или опыт. Память подразделяется на краткосрочную (short-term) и долгосрочную (long-term). Краткосрочная описывает текущий контекст, а долгосрочная — позволяет агенту аккумулировать факты и знания. Инструменты (tools) обеспечивают взаимодействие с внешними системами, API и источниками данных, позволяя агенту взаимодействовать с окружающей средой.

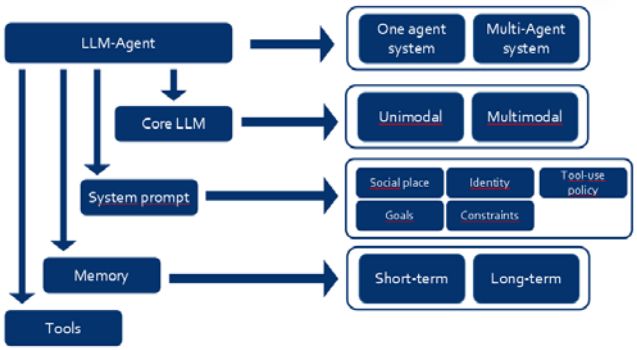

Рис. 1. Набор компонентов агентных систем

Большие языковые модели продемонстрировали выдающиеся способности в генерации текста, рассуждении (reasoning) и работе с инструментами (tool-

use). Однако их ключевое ограничение — статичность знаний, полученных во время обучения.

Агентные системы от запуска к запуску могут развиваться (evolution), то есть становиться более эффективными. Это достигается за счёт накопления опыта, адаптации стратегий, оптимизации памяти и улучшения взаимодействия с внешними инструментами и средой.

К основным проблемам агентных систем можно отнести:

- отсутствие в архитектуре трансформеров долгосрочной памяти [3];
- некорректный вызов инструментов или выбор неподходящего инструмента [4].

Первая проблема решается с помощью различных методов, количество и разнообразие которых значительно возросло за последние годы. Вторая проблема решается через развитие LLM-агентов, для чего также предложено множество подходов.

В результате исследователи сталкиваются с трудностями систематизации этих методов и подходов. Настоящая работа направлен на попытку упорядочить существующее разнообразие технологий. В работе рассматриваются ключевые методы, позволяющие агентам развиваться, что может помочь как исследователям, так и практикам более осознанно подходить к выбору инструментов для решения своих задач.

II. Подходы развития LLM-агентных систем

Agent evolution — это совокупность подходов к развитию агентов. Можно выделить два принципиально разных направления:

- динамическое (dynamic) развитие (inference-time adaptation) — модель не меняется по весам, а адаптируется во время работы;
- статическое (static) развитие (training-time adaptation) — изменения происходят через модификацию параметров.

A. Статические методы

Статические подходы изменяют модель через обучение или тонкую настройку параметров агента. То есть агент останавливается, и меняются либо параметры ядра LLM, либо системный промпт и инструменты. Можно выделить 2 основных направления: изменение параметров LLM-модели и изменение других компонентов агента. Первое направление обычно подразделяется на Fine-tuning и Reinforcement Learning (RL). Второе — на изменение системного промпта и изменение инструментов.

Изменение системного промпта — это процесс корректировки инструкций, которые управляют поведением агента. Промт-инженер может добавлять, убирать или изменять правила, чтобы модель справлялась с задачей более эффективно. Под изменение инструментов понимается добавление новых, удаление существующих и редактирования инструкций вызова старых.

Fine-tuning представляет собой процесс дообучения предобученной модели на специализированном датасете с целью адаптации к конкретной задаче или домену [5]. В контексте больших языковых моделей этот подход позволяет перенастроить внутренние представления, сформированные на общем корпусе данных, под более узкие распределения. Fine-tuning сопряжён с рядом ограничений: он требует значительных вычислительных ресурсов и может приводить к утрате ранее усвоенных знаний.

Reinforcement Learning (RL) используется для обучения агентов через взаимодействие со средой с целью максимизации награды. В контексте LLM-Агентов RL стал ключевым инструментом для выравнивания (alignment) поведения с человеческими ожиданиями и обучения сложного поведения: вызова инструментов (tool-use), планирования (planning) или утилизации памяти (memory management).

Обучение с подкреплением (RL) для alignment направлен на согласование поведения модели с человеческими ожиданиями [6]. В таких подходах, как Reinforcement Learning from Human Feedback (RLHF), Reinforcement Learning from AI Feedback (RLAIF) и Direct Preference Optimization (DPO), модель обучается не только на данных, но и на предпочтениях, которые задают, какие ответы считаются более полезными (helpful), безопасными (harmless) и честными (honest). Награда формируется так, чтобы поощрять желаемое поведение (например, правдивость, полезность, отказ от вредных инструкций) и штрафовать нежелательное. Это позволяет сместить распределение выходов модели в сторону более надёжных и контролируемых ответов, даже если такие предпочтения слабо представлены в исходных данных.

Обучение с подкреплением (RL) для tool-use фокусируется на обучении модели правильно выбирать, вызывать и комбинировать внешние инструменты (API, базы данных, калькуляторы, поисковые системы) для решения задач [4]. В этом контексте агент учится не только генерировать текст, но и принимать решения о том, когда и какой инструмент использовать, а также как интерпретировать его результат. Для обучения вызова инструментов также используется и подход само-контролируемого обучения (self-supervised learning), как это было, например, в ToolFormer.

Обучение с подкреплением (RL) для memory management обучает модель эффективно использовать и обновлять память. Это включает выбор того, какую информацию сохранять, когда её извлекать и как забывать нерелевантные данные [7]. В сценариях с долгим контекстом или многосессионным взаимодействием агент должен балансировать между полнотой памяти и её компактностью, так как контекстное окно не бесконечно.

В случае, если вычислительных ресурсов не хватает для вышепредставленных методов, то используют подход Parameter-Efficient Fine-Tuning (PEFT). Это набор методов дообучения больших нейросетей (в том числе и больших языковых моделей), которые изменяют только небольшие дополнительные модули или низкоразмерные параметры, а большая часть параметров модели остаётся неизменной. То есть PEFT решает ключевую проблему fine-tuning — стоимость. К основным методам данной

категории можно отнести: LoRA, Adapters, Prefix Tuning и Prompt Tuning.

LoRA аппроксимирует обновление весов через низкоранговые матрицы, один из самых влиятельных методов PEFT. Вместо обновления полной матрицы весов, LoRA фиксирует исходные веса и добавляет разложение. Это резко уменьшает число обучаемых параметров, вычислительную стоимость и обеспечивает почти тот же уровень качества, что и, например, full fine-tuning, при существенно меньших ресурсах.

Adapters представляют собой небольшие нейронные модули, вставляемые между слоями трансформера. Основная модель замораживается, а обучаются только эти вставленные блоки. Adapters для обеспечения того же качества, что предоставляет LoRA, обычно требуют больше параметров, но у adapters есть преимущество — модульность.

Prefix-Tuning предполагает обучение специальных префиксных векторов, которые добавляются к входу каждого слоя трансформера. В этом случае эти векторы интерпретируются как «виртуальные токены», влияющие на поведение модели без изменения её весов. Более легковестное решение, чем adapters, но для сложных задач, где требуется глубокая модификация внутренних представлений модели, подход менее эффективен.

Ещё более простое решение, чем Prefix-Tuning — это Prompt-Tuning. В этом методе обучаются только векторные представления (embedding) специальных входных токенов, добавляемых к началу последовательности. То есть эти параметры влияют только на вход, а не на все слои модели. Метод крайне приятный с точки зрения памяти и прост в реализации, но менее эффективен по сравнению с вышеперечисленными.

B. Динамические методы

Динамические подходы адаптируют модель во время работы агента (inference), не изменяя её параметры большой языковой модели.

LLM-агенты могут эволюционировать во время работы. В какой-то момент агент останавливает выполнение текущей задачи, анализирует всё, что произошло до этого, и делает выводы. Такое направление можно назвать «knowledge updating», которое глобально подразделяется на: abstraction, self-reflection и environment exploration.

Abstraction — это процесс перехода от конкретных наблюдений или фактов, полученных во время работы, к более общим, высокоуровневым представлениям. В LLM-агентах это проявляется в способности выделять паттерны, инварианты и скрытые структуры, игнорируя нерелевантные детали. Абстракция нужна для переноса знаний (generalization). То есть агент может применять ранее полученный опыт к новым задачам. В долгоживущих агентах abstraction также используется и для компрессии памяти — замены множества частных эпизодов их обобщённым представлением, но ключевой момент здесь, что новые полученные знания сохраняются отдельно, и агент может получить к ним доступ при необходимости даже спустя несколько контекстных окон. Чтобы получать новые знания, агент может составлять цепочку рассуждений путём комбинирования уже имеющихся фактов, представлений или промежуточных выводов. Получается цепочка рассуждений, последний вывод из которой может агенту помочь решить текущую проблему . Можно улучшить данный подход и заставить агента генерировать несколько путей рассуждения и выбрать лучший, или попробовать несколько и выбрать лучший. Современные достижения в агентных системах позволяют агенту самому решать, когда включить данный механизм.

Self-reflection — это механизм, при котором агент анализирует собственные ответы, ошибки и рассуждения с целью их последующего улучшения [2]. Self-reflection вводит дополнительный мета-уровень: модель сначала решает задачу, затем оценивает качество своего решения и при необходимости пересматривает его. Этот подход тесно связан с идеями итеративного рассуждения и демонстрирует значительное улучшение качества без изменения параметров модели.

Environment exploration описывает активное взаимодействие агента с его окружающей средой с целью получения новых знаний, проверки гипотез и соответственно улучшения своего поведения. Агент сам инициирует действия, которые изменяют состояние среды, и ему предоставляется обратная связь. Причём такие действия могут негативно сказываться на прогрессе решения текущей задачи агента. Этот подход тесно связан с reinforcement learning, но в этом контексте (динамические методы) реализуется без явного изменения весов, а через специальное накопление опыта во внешней памяти и его повторное использование.

LLM ограничены контекстным окном, что требует стратегий управления памятью. К основным подходам в оптимизации памяти агентов можно отнести:

- FIFO [8];
- Summarization [3];
- External Storage [9, 10];
- Task Delegation [10].

FIFO/truncation представляет собой базовую стратегию управления контекстным окном, при которой самые старые элементы диалога или памяти удаляются по мере заполнения доступного пространства. Этот подход не требует дополнительных вычислений или моделей и обеспечивает предсказуемое использование памяти. Главный недостаток — это отсутствие семантической чувствительности: важная информация может быть удалена только из-за своей «давности», что приводит к деградации качества и эффективности [8]. Несмотря на примитивность, FIFO часто используется как базовый механизм в более сложных сценариях оптимизации памяти, где он комбинируется с другими стратегиями фильтрации.

Summarization — это стратегия компрессии контекста, при которой накопленная история взаимодействия преобразуется в более компактное представление с сохранением ключевой информации [3]. В отличие от FIFO, этот подход учитывает семантику и позволяет поддерживать долгосрочную когерентность диалога. При многократной компрессии возможна потеря деталей и накопление ошибок.

External storage переносит память за пределы контекстного окна агента, используя внешние хранилища: векторные базы данных, файловые системы. Вместо хранения всей информации в краткосрочной памяти, агент извлекает релевантные фрагменты по мере необходимости. Можно использовать отдельный файл, в котором содержится оглавление, чтобы агент мог себе в контекстное окно добавлять нужные знания сам (dynamic context discovery) [9]. Также агент может явным образом сам записывать информация в файл для себя же в будущем или для других агентов [10]. Это позволяет масштабировать память практически неограниченно и поддерживать долговременное хранение знаний.

Task delegation предполагает разбиение сложной задачи на подзадачи и передачу их специализированным агентам, моделям или инструментам. Это позволяет преодолеть ограничения одной модели в контекстном окне [10]. Делегирование может быть статическим (предопределённые роли) или динамическим (решения принимаются во время выполнения). Такой подход лежит в основе многоагентных систем. У нововызванного агента есть своя задача и своё контекстное окно, а результат, который он вернёт, будет меньше занимать места в памяти, чем если бы изначальный агент выполнял задачу сам.

Во время работы агент может получать новые знания явным образом. Это могут быть как ранее им же накопленные знания, так и заранее заготовленные, так и потенциальные. То есть агент из долгосрочной (long-term) памяти переводит знания в краткосрочную (short-term) с помощью следующих подходов: Retrieval-Augmented Generation (RAG), databases, API и Expert.

Retrieval-Augmented Generation (RAG) представляет позволяет объединить параметрические знания языковой модели с внешним непараметрическим хранилищем. С помощью RAG динамически извлекаются релевантные документы (обычно через векторный (embedding-based) поиск в базе знаний) и включает их в память агента [11]. Это позволяет существенно повысить актуальность, проверяемость и доменную точность ответов, а также уменьшить галюцинации. Обычно RAG используется для работы с неструктурированной информацией: сырые текстовые файлы, документы и так далее.

Работа с databases фокусируется на извлечении и манипуляции структурированной информацией. Данные строго нормализованы и требуют высокой точности в ответе от агента. Взаимодействие с базами данных часто дополняется итеративным уточнением запросов [12]. То есть агент обычно пользуется данным подходом, когда заранее известно какие операции будут выполняться, какие таблицы и поля доступны. Результат запроса агента к таким структурированным данным напрямую «вставляется» в его краткосрочную (short-term) память.

Интеграция API позволяет агенту выполнять реальные действия, согласно API. Запрос делается во внешние сервисы и источники. В отличие от RAG и databases, ресурсы, доступные по API будут более актуальные (например, текущая погода, текущее время или релевантные статьи из новостных источников) [13].

Подход Expert предполагает использование экспертных систем. Агент задаёт вопрос специализированным моделям, обученным на узкой доменной области, которые зачастую имеют на порядок больше параметров. Такие модели обладают глубокой «экспертизой» и зачастую более высокой точностью в своей области [14]. Таким образом более тяжёлая модель будет вызываться редко, в то время как более «шустрый» и универсальный агент будет работать далее после получения ответа от эксперта. LLM-Агент в таком сценарии выполняет роль мета-контроллера: он определяет, когда задача требует экспертного знания, формирует запрос и агрегирует результат. Этот подход может сработать, где агентская модель недостаточно надежна. В данном случае возникает композиционная система, где агент со способностью вызывать инструменты (tool-use) дополняется узкоспециализированными компонентами.

III. Обсуждение

Проведенный анализ развития агентов на базе больших языковых моделей [1–14], позволил сформировать их классификацию способов развития в виде графовой модели, представленной на рис. 2.

Рис. 2. Классификация LLM-агентов по способам их развития

Разработанная классификация развитие агентных систем на базе больших языковых моделей (рис. 2), а также произведенный анализ, сформированный на основании 14 источников, выявил следующие закономерности:

- проблема ограниченного контекста актуальна до сих пор, модели плохо удерживают длинную историю взаимодействия, дополнительное применение внешней памяти только частично решает проблемы, а риск утраты деталей диалога и результатов обработки сохраняется;
- ограничение вычислительных ресурсов требует дальнейшего развития динамических методов (inference-time adaptation);
- сохранение тенденций дообучении моделей с помощью обучения с подкреплением;

- увеличение интереса исследователей к развитию методов построения гибкой архитектуры LLM-агентов и способов их взаимодействия через элементы динамической модификации структуры.

IV. Заключение

Агентные системы на базе больших языковых моделей представляют собой важный этап развития искусственного интеллекта, позволяя создавать гибкие, адаптивные и автономные решения. В ходе работы было показано, что эволюция таких систем может осуществляться как за счёт изменения параметров моделей (статические методы), так и через адаптацию во время выполнения (динамические методы).

Статические подходы, включая fine-tuning, reinforcement learning и PEFT, обеспечивают глубокую настройку поведения модели, однако требуют значительных вычислительных ресурсов и не подходят для быстрой адаптации. В то же время динамические методы позволяют агентам развиваться в процессе работы за счёт накопления и обработки опыта, управления памятью и взаимодействия с внешними источниками знаний.

Особую роль играют механизмы работы с памятью и внешними ресурсами, такие как RAG, API и базы данных, которые позволяют преодолеть ограничения контекстного окна и статичность знаний LLM. Многоагентные архитектуры и делегирование задач также открывают новые возможности для масштабирования и повышения эффективности.

Список литературы

[1] Yin S. et al. A survey on multimodal large language models // National Science Review. 2024. vol. 11. no. 12. pp. nwae403. DOI: 10.1093/nsr/nwae403.

[2] Yao S. et al. ReAct: Synergizing reasoning and acting in language models // Proceedings of the Eleventh International Conference on Learning Representations (ICLR), Kigali, Rwanda, 1–5 May 2023. OpenReview – 2023. DOI: 10.48550/arXiv.2210.03629.

[3] Zhong W. et al. MemoryBank: Enhancing Large Language Models with Long-Term Memory // Proceedings of the Thirty-Eighth AAAI conference on artificial intelligence, Vancouver, Canada, February 20–27, 2024. Washington, DC, USA: AAAI Press – 2024. vol. 38. no. 17. pp. 19724–19731. DOI: 10.1609/aaai.v38i17.29946.

[4] Feng J. et al. ReTool: Reinforcement Learning for Strategic Tool Use in LLMs // arXiv preprint arXiv:2504.11536. – 2025. DOI: 10.48550/arXiv.2504.11536.

[5] Radford A. et al. Improving language understanding by generative pre-training // OpenAI technical report. – 2018.

[6] Ouyang L. et al. Training language models to follow instructions with human feedback // Proceedings of the Advances in Neural Information Processing Systems (NeurIPS 2022), New Orleans, USA, 28 Nov – 9 Dec 2022. – Red Hook, NY, USA: Curran Associates, Inc – 2022. vol. 35. pp. 27730–27744. DOI: 10.48550/arXiv.2203.02155.

[7] Zhou Z. et al. MEM1: Learning to Synergize Memory and Reasoning for Efficient Long-Horizon Agents // arXiv preprint arXiv:2506.15841. – 2025. DOI: 10.48550/arXiv.2506.15841.

[8] Beltagy I., Peters M. E., Cohan A. Longformer: The Long-Document Transformer // arXiv preprint arXiv:2004.05150. – 2020. DOI: 10.48550/arXiv.2004.05150.

[9] Jediah Katz (Dynamic context discovery) Available at: https://cursor.com/blog/dynamic-context-discovery (accessed 15 March 2026).

[10] Anthropic (Effective harnesses for long-running agents) Available at: https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents (accessed 15 March 2026).

[11] Fan W. et al. A Survey on RAG Meeting LLMs: Towards Retrieval-Augmented Large Language Models // Proceedings of the Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, Barcelona, Spain. New York, NY, USA: Association for Computing Machinery – 2024. pp. 6491–6501. DOI: 10.1145/3637528.3671470.

[12] Hu C. et al. ChatDB: Augmenting LLMs with Databases as Their Symbolic Memory // arXiv preprint arXiv:2306.03901. – 2023. DOI: 10.48550/arXiv.2306.03901.

[13] Nakano R. et al. Webgpt: Browser-assisted question-answering with human feedback // arXiv preprint arXiv:2112.09332. – 2021. DOI: 10.48550/arXiv.2112.09332.

[14] Youwai S. et al. Large language model-based multi-agent systems for automated foundation design: router-driven task classification and expert selection framework // AI in Civil Engineering. 2026. vol. 5. no. 1. pp. 5. DOI: 10.1007/s43503-026-00088-8.