Архитектурные паттерны для постоянной памяти в агентных системах на базе LLM: сравнительный анализ

Краткий обзор

Распространение больших языковых моделей (LLM) стало преобразующим фактором, однако их базовая архитектура по своей сути является stateless (без сохранения состояния). Это ограничение, часто описываемое как «разговорная амнезия», проистекает из конечных окон контекста, которые ограничивают их способность сохранять информацию в ходе длительных взаимодействий.[1]
Этот единственный фактор остается самым значительным препятствием на пути создания по-настоящему постоянных, совместных и персонализированных ИИ-агентов, способных выполнять сложные, длительные задачи.
В ответ на эту проблему отрасль вышла за рамки простой stateless-архитектуры Retrieval-Augmented Generation (RAG) и разработала сложные архитектурные паттерны для долговременной памяти.

В этом отчете определены и проанализированы четыре различные архитектурные философии, которые стали современным стандартом в готовых к производству системах:

1. **Парадигма операционной системы (MemGPT):** Подход, который рассматривает память как управляемый ресурс, виртуализируя контекст LLM для создания иллюзии бесконечной емкости.
2. **Глобальный механизм персонализации (OpenAI):** Продукто-ориентированная архитектура, которая рассматривает память как функцию для создания бесшовного, персонализированного пользовательского опыта во всех взаимодействиях.
3. **Модель явного определения контекста (Claude):** Дизайн, который отдает приоритет контролю пользователя и строгой изоляции данных, реализуя память как инструмент для проектной работы.
4. **Инструментарий разработчика (агентные фреймворки):** Гибкий, компонентный

подход, который предоставляет разработчикам строительные блоки для создания пользовательских систем памяти.

Для обеспечения строгой аналитической основы в этом отчете каждая система оценивается по концептуальному эталону: **Память как услуга (MaaS)**. Концепция MaaS определяется как проектно-ориентированная, многоканальная и stateful-система памяти, которая хранит информацию в виде структурированных символических сущностей (например, факты, решения) и использует оркестрованный многоагентный конвейер для курирования знаний.

Выводы высокого уровня указывают на явный компромисс в текущих системах между автоматическим удобством, предлагаемым глобальными моделями персонализации, и явным контролем, предоставляемым архитектурами с проектной областью видимости.

Более продвинутые паттерны, примером которых являются MemGPT и возможности агентных фреймворков, указывают на будущее автономной, самоуправляемой памяти.

Однако ни одна производственная система сегодня не реализует полностью структурированное, совместное и реляционное видение концепции MaaS. Необходимые компоненты теперь доступны, но их интеграция в единое целое остается следующим рубежом в развитии агентного ИИ.

Анализ системы I: MemGPT — парадигма операционной системы

MemGPT представляет собой фундаментальный архитектурный сдвиг, переосмысливая проблему памяти с управления контентом на управление ресурсами. Черпая прямое вдохновение из дизайна компьютерных операционных систем, он рассматривает конечное окно контекста LLM не как жесткое ограничение, а как форму быстрой, энергозависимой памяти (вроде ОЗУ), которой необходимо разумно управлять в сочетании с большим, постоянным уровнем хранения (вроде жесткого диска).[1]

Основная архитектура: управление виртуальным контекстом

Ядром MemGPT является его иерархическая система памяти, прямая аналогия с архитектурой памяти в обычной ОС.[2] Эта архитектура разделена на два основных уровня:

- **Основной контекст (ОЗУ):** Это промпт LLM фиксированного размера, представляющий единственную информацию, к которой процессор LLM может

напрямую получить доступ или «увидеть» во время цикла вывода. Этот уровень далее подразделяется на три отдельных раздела: статический системный промпт, содержащий основные инструкции и схемы функций; динамический рабочий контекст, который действует как черновик для мыслей агента и промежуточных результатов; и очередь First-In, First-Out (FIFO), которая содержит самые последние сообщения в разговоре.[3]

- **Внешний контекст (дисковое хранилище):** Это внеконтекстный, фактически бесконечный уровень хранения, к которому LLM не может получить прямой доступ. Он состоит из двух подуровней: **хранилище для вызова (Recall Storage)**, поисковая база данных (часто простое хранилище документов), содержащая полную историю всех событий и сообщений для точного, буквального извлечения, и **архивное хранилище (Archival Storage)**, долгосрочное хранилище, обычно векторное, используемое для больших документов или для хранения обобщенных знаний, которые можно извлечь с помощью семантического поиска.[3]

Поток данных регулируется строгим механизмом подкачки. Процессор LLM работает исключительно с основным контекстом. Для доступа к информации, находящейся во внешнем контексте, LLM должен автономно генерировать и выполнять явные вызовы функций, такие как conversation_search или archival_memory_search.[2] Результат этих вызовов функций затем «подкачивается» в основной контекст, часто вытесняя более старую, менее релевантную информацию из очереди FIFO. Этот процесс эффективно виртуализирует контекст, создавая иллюзию неограниченного окна памяти при работе в физических ограничениях базовой модели.[9] Извлечение из внешнего контекста может выполняться с использованием текстового, временного или семантического поиска на основе вложений.[7]

Процесс формирования памяти: самоуправляемый цикл «обратной записи»

Процесс фиксации информации в долговременной памяти — цикл «обратной записи» — это не непрерывная фоновая задача, а событийно-управляемый процесс, который функционирует как прерывание ОС.[1] Основным триггером для этого процесса является **давление на память**. Когда количество токенов в основном контексте приближается к операционному лимиту LLM (например, 70% емкости), в очередь контекста вставляется системное оповещение.[3] Это предупреждение действует как прерывание, побуждая процессор LLM приостановить свою текущую задачу и инициировать цикл управления памятью.

Определяющей характеристикой MemGPT является то, что сам LLM отвечает за это управление памятью. Получив предупреждение о давлении на память, LLM анализирует свой собственный основной контекст — недавний разговор и свой рабочий черновик — чтобы решить, какая информация является наименее важной. Затем он генерирует необходимые вызовы функций для обобщения этой информации и перемещения ее на

соответствующий уровень внешнего контекста.[2] Это полностью автоматизированный, саморефлексивный цикл, в котором агент управляет своими собственными когнитивными ресурсами. Эта возможность саморедактирования также позволяет агенту динамически исправлять свою память, например, перезаписывая ранее сохраненный факт новой информацией, предоставленной пользователем.[4]

Инструменты и технологический стек

Архитектура MemGPT чаще всего реализуется с использованием следующего стека:

- **Основной фреймворк:** Эталонная реализация — это фреймворк Python с открытым исходным кодом, первоначально названный MemGPT, который с тех пор превратился в фреймворк Letta.[10]
- **LLM:** Архитектура не зависит от модели, но наиболее эффективна в паре с мощными моделями, поддерживающими вызов функций, такими как GPT-4 и GPT-3.5 от OpenAI.[1] Она также обеспечивает поддержку работы с локальными и открытыми моделями.[10]
- **Векторные базы данных:** Уровень архивного хранилища обычно реализуется с использованием векторной базы данных для обеспечения семантического поиска. Распространенными вариантами, упоминаемыми в документации и примерах сообщества, являются Chroma, LanceDB и pgvector.[10]
- **Постоянное хранение:** Хранилище для вызова, которое содержит необработанную историю событий, часто представляет собой более простой уровень постоянного хранения, такой как локальная база данных или файловая система.[11]

Сравнительный анализ с концепцией «MaaS»

- **Сходства:** И MemGPT, и концепция MaaS предусматривают автоматизированную, агентную систему, в которой операции с памятью являются неотъемлемой частью основного цикла рассуждений агента. Обе системы предназначены для поддержки постоянных, длительных взаимодействий.
- **Различия:**
    - **Структура:** Наиболее существенное различие заключается в структуре хранимой памяти. Память MemGPT в основном **неструктурирована** и состоит из необработанных фрагментов текста, журналов сообщений и фрагментов документов. В отличие от этого, MaaS требует **структурированной символической памяти**, где информация хранится в виде типизированных сущностей, таких как факты, решения и вопросы.
    - **Конвейер обработки:** Формирование памяти в MemGPT — это монолитный и реактивный процесс, обрабатываемый одним процессором LLM в ответ на

системные прерывания. MaaS предлагает **оркестрованный многоагентный конвейер** (например, агенты-нормализаторы, линкеры, экстракторы), который проактивно работает над курированием и структурированием знаний.
    - **Область применения и фокус:** MemGPT в первую очередь ориентирован на пользователя, предназначен для предоставления одному агенту долговременной памяти для персонализации или для анализа большого корпуса документов.[8] MaaS явно **проектно-ориентирован и предназначен для совместной работы**, задуман как общее хранилище знаний для команд.
- **Уникальные преимущества/недостатки:**
    - **Преимущество:** Основное преимущество MemGPT — это его элегантное и мощное решение проблемы конечного окна контекста. Абстрагируя управление памятью в функцию, подобную ОС, он эффективно создает «неограниченный контекст» для LLM.[1]
    - **Недостаток:** Опора на одного агента как для выполнения задач, так и для управления памятью создает «когнитивную нагрузку». Поскольку память агента неструктурирована, выполнение сложных реляционных запросов (например, «Какие решения были приняты под влиянием фактов из источника X?») невозможно без значительной постобработки — ограничение, которое архитектура MaaS специально разработана для преодоления.

Дизайн MemGPT, где LLM управляет своей собственной памятью, выявляет критический компромисс в производительности. Задачи управления памятью — разбор предупреждений, обобщение контекста для вытеснения и генерация вызовов функций — все это потребляет ценные токены и вычислительные ресурсы в ограниченном окне контекста LLM.[2] Это означает, что в любом данном цикле обработки часть «мыслительной» способности агента посвящена логистике памяти, а не непосредственному решению основного запроса пользователя. Это создает когнитивную нагрузку, когда преимущество практически бесконечного контекста достигается за счет снижения когнитивной способности для основной задачи в каждом шаге.

Анализ системы II: Память OpenAI — глобальный механизм персонализации

Функция памяти OpenAI для ChatGPT представляет собой продукто-ориентированную архитектуру, направленную на создание бесшовного, глубоко персонализированного пользовательского опыта. Она резко контрастирует с другими подходами, реализуя глобальную, ориентированную на пользователя память, которая сохраняется во всех разговорах, стремясь сделать ИИ-помощника постоянно осведомленным и полезным с

минимальными усилиями со стороны пользователя.[15]

Основная архитектура: гибридное хранилище фактов и семантики

Архитектура представляет собой сложную гибридную систему, предназначенную для захвата как явных инструкций, так и неявного контекста из взаимодействий с пользователем.[15] Она состоит из двух основных компонентов:

- **Сохраненные воспоминания (явное хранилище фактов):** Этот уровень функционирует как специфичная для пользователя база данных, подобная структурированному «блокноту» или хранилищу ключ-значение. Он содержит дискретные факты, предпочтения и инструкции, которые пользователь явно просит ChatGPT запомнить или которые система автоматически определяет и помечает как важные.[15] Эта коллекция фактов фактически добавляется в начало контекста каждой новой сессии, обеспечивая постоянную осведомленность об основных атрибутах пользователя.[19]
- **Ссылка на историю чатов (неявное семантическое хранилище):** Этот уровень работает как мощная система RAG над всей историей разговоров пользователя. Когда пользователь вводит промпт, система выполняет семантический поиск по всем прошлым разговорам, чтобы найти релевантные фрагменты информации.[15] Эти извлеченные фрагменты затем вставляются в контекст, обеспечивая релевантный фон, даже если информация никогда не была явно сохранена.

Поток данных для генерации ответа начинается с запроса к обоим компонентам памяти. Система извлекает все релевантные «Сохраненные воспоминания» и одновременно выполняет семантический поиск по «Истории чатов». Эта объединенная контекстная информация затем упаковывается с текущим промптом пользователя и передается основной LLM (например, GPT-4) для генерации персонализированного и контекстуально осведомленного ответа.[16]

Процесс формирования памяти: управляемый пользователем и автоматизированный ИИ

Цикл обратной записи для памяти OpenAI — это двухрежимный процесс, который сочетает в себе контроль пользователя с автоматизированным интеллектом:

- **Явные команды:** Самый прямой метод формирования памяти — это команда пользователя, например, «Запомни, что я живу в Сан-Франциско и предпочитаю краткие резюме».[15] Это запускает процесс, который разбирает инструкцию и сохраняет информацию непосредственно в слое «Сохраненные воспоминания».
- **Автоматическое извлечение:** Система также использует автоматизированный фоновый процесс. Специализированная LLM или тонко настроенная модель

классификации постоянно анализирует разговоры для выявления и извлечения важной информации, которая кажется значимой для будущих взаимодействий, такой как профессия пользователя, цели или повторяющиеся предпочтения.[16] Эти автоматически извлеченные факты затем предлагаются или добавляются в хранилище «Сохраненные воспоминания».

- **Контроль и вето пользователя:** Важнейшим аспектом дизайна является то, что пользователь сохраняет окончательный контроль. Специальный интерфейс настроек позволяет пользователям просматривать, редактировать и удалять любое конкретное воспоминание, которое было сохранено, будь то вручную или автоматически.[15] Это обеспечивает жизненно важный механизм для исправления, управления конфиденциальностью и тонкой настройки личности ИИ.

Инструменты и технологический стек

Весь технологический стек для этой функции является проприетарным и внутренним для инфраструктуры OpenAI. Однако, основываясь на стандартных отраслевых практиках для таких систем, стек, вероятно, включает:

- Высокопроизводительную, масштабируемую векторную базу данных для функции семантического поиска по «Истории чатов».
- Надежную базу данных ключ-значение или документов (аналогичную Amazon DynamoDB или Google Firestore) для управления структурированными «Сохраненными воспоминаниями».
- Внутренний конвейер моделей классификации и извлечения, предназначенный для автоматизированного процесса формирования памяти.
- Все компоненты тесно интегрированы в основную архитектуру сервиса ChatGPT для обеспечения низкой задержки и высокой доступности.

Сравнительный анализ с концепцией «MaaS»

- **Сходства:** Обе системы нацелены на создание постоянной базы знаний из взаимодействий с пользователем. Автоматическое извлечение фактов в системе OpenAI концептуально схоже с агентом «Write-back Extractor», предложенным в конвейере MaaS.
- **Различия:**
    - **Область применения и аренда:** Это самое фундаментальное различие. Память OpenAI **ориентирована на пользователя и глобальна**. Она создает единый, монолитный профиль пользователя, который применяется ко всем разговорам, независимо от темы.[16] MaaS, напротив, **проектно-ориентирован и многопользовательский**, предназначен для профессионального сотрудничества со строгой изоляцией данных между

различными проектами.
    - **Структура:** Хотя «Сохраненные воспоминания» OpenAI являются полуструктурированными (вероятно, хранятся как текстовые факты или пары ключ-значение), MaaS требует гораздо более богатой, **символической структуры** с типизированными сущностями (факт, решение) и явными отношениями между ними.
    - **Сотрудничество:** Память OpenAI предназначена для персонализации одного пользователя. MaaS в своей основе **предназначен для совместной работы**, задуман как общий мозг для команды, работающей над общей целью.
- **Уникальные преимущества/недостатки:**
    - **Преимущество:** Архитектура превосходно создает бесшовный и почти «волшебный» пользовательский опыт. ИИ-помощник кажется постоянно персонализированным и интеллектуальным без каких-либо усилий по настройке или управлению со стороны пользователя, что очень эффективно для массового потребительского продукта.[15]
    - **Недостаток:** Глобальный, ориентированный на пользователя характер делает его по своей сути непригодным для многих профессиональных и корпоративных контекстов. Риск «утечки контекста» — когда конфиденциальная информация из разговора о клиенте А может непреднамеренно повлиять на более поздний разговор о клиенте Б — является серьезной проблемой, которую архитектура MaaS специально разработана для предотвращения.[20]

Дизайн системы памяти OpenAI — это не просто технический выбор, а прямое отражение ее основной бизнес-стратегии. Основной целью ChatGPT является массовая, глобальная аудитория потребителей и просьюмеров.[17] Для этой демографической группы ключевыми факторами принятия и долгосрочного вовлечения являются простота использования и сильное чувство персонализации. Глобальная, автоматизированная система памяти, которая создает единый, всеобъемлющий профиль пользователя, напрямую служит этой цели. Она минимизирует трение для пользователя, устраняя необходимость управлять проектами или контекстами, и максимизирует ощущение непрерывных, развивающихся отношений с ИИ. Напротив, функции, центральные для концепции MaaS, такие как строгая изоляция данных, символические структуры знаний и многоагентные конвейеры, внесли бы уровень сложности, который не нужен для основной бизнес-модели. Таким образом, архитектура памяти OpenAI является прямым проявлением ее потребительской стратегии.

Анализ системы III: Память Claude — модель явного определения контекста

Подход Claude к памяти представляет собой философский контрапункт глобальной

модели персонализации OpenAI. Anthropic отдает приоритет контролю пользователя, явной активации и строгой контекстуальной изоляции, что приводит к созданию системы памяти, которая менее автоматизирована, но более предсказуема и лучше подходит для профессиональных и разработческих рабочих процессов, где важна компартментализация данных.[23]

Основная архитектура: резюме в рамках проекта и контекст на основе файлов

Функциональность памяти Claude реализуется через многогранную архитектуру, которая сочетает в себе официальные функции с возникающими, управляемыми сообществом паттернами:

- **Официальная память (планы Team/Enterprise):** Для платящих клиентов Claude предлагает **резюме памяти в рамках проекта**.[25] В интерфейсе Claude пользователи могут создавать отдельные «проекты». Каждый проект поддерживает свое собственное, отдельное, редактируемое резюме ключевых фактов, инструкций и контекста. Это резюме затем автоматически включается в промпт для всех разговоров, которые происходят в рамках этого проекта. Этот дизайн обеспечивает жесткое разделение контекстов; информация из «Проекта А» никогда не будет доступна или использована в «Проекте Б», обеспечивая конфиденциальность и релевантность.[24]
- **Память, управляемая сообществом (CLAUDE.md):** В сообществе разработчиков, особенно для задач кодирования, появился мощный паттерн. Он заключается в размещении простого файла markdown с именем CLAUDE.md в корневом каталоге проекта.[26] Когда пользователь взаимодействует с Claude в этом каталоге, система автоматически считывает содержимое этого файла и включает его в контекст. Это позволяет командам определять и контролировать версии статического контекста, такого как архитектурные принципы, соглашения о кодировании и спецификации API, непосредственно рядом с их кодом.[26] Этот метод функционирует скорее как «контекстуальное извлечение», чем как динамическая система RAG, поскольку весь файл загружается в окно контекста.[27]
- **Извлечение на основе инструментов:** Обратная инженерия поведения Claude предполагает, что когда пользователь явно просит его вспомнить информацию (например, «Что мы обсуждали о дизайне API на прошлой неделе?»), он использует внутренние инструменты в реальном времени, такие как conversation_search и recent_chats. Эти инструменты выполняют поиск по необработанной истории разговоров *только текущего проекта*, а не полагаются на предварительно вычисленное резюме или векторный индекс.[23]

Процесс формирования памяти: преимущественно курируемый

пользователем

В резком контрасте с автоматизированной системой OpenAI, формирование памяти Claude почти полностью явное и управляемое пользователем. Новый чат начинается с чистого листа, если он не инициирован в проекте, который уже имеет резюме памяти или файл CLAUDE.md.[23] Цикл «обратной записи» — это ручной процесс:

- Для официальной функции памяти пользователь должен явно общаться с Claude, чтобы проинструктировать его, как обновить резюме памяти проекта.[25]
- Для паттерна на основе файлов пользователь или разработчик напрямую редактирует файл CLAUDE.md с помощью стандартного текстового редактора.[26]

Память не является «всегда включенной». Ее использование часто инициируется пользователем, явно побуждающим Claude ссылаться на прошлую информацию, что, в свою очередь, активирует внутренние инструменты поиска.[23]

Инструменты и технологический стек

- **Платформа Anthropic:** Официальное резюме памяти в рамках проекта является проприетарной функцией, интегрированной в веб-приложение и API Claude, доступной пользователям планов Team и Enterprise.[25]
- **Файловая система и контроль версий:** Паттерн CLAUDE.md не требует специальных технологий, кроме стандартной файловой системы, текстовых редакторов и систем контроля версий, таких как Git.[26]
- **Внешние коннекторы:** Появляется экосистема сторонних инструментов для соединения Claude с внешними системами памяти. Например, Pieces CLI предоставляет коннектор, который добавляет в Claude инструменты, такие как ask_pieces_ltm (запросить долговременную память) и create_pieces_memory, позволяя ему взаимодействовать с локальным, постоянным хранилищем памяти.[29]

Сравнительный анализ с концепцией «MaaS»

- **Сходства:**
  - **Проектно-ориентированный фокус:** Самая сильная точка соприкосновения между Claude и MaaS — это их фундаментально **проектно-ориентированный** характер. Обе архитектуры построены на принципе, что в профессиональной работе контекст должен быть строго изолирован для конкретных проектов или инициатив.[24]
- **Различия:**
  - **Автоматизация и структура:** Формирование памяти Claude является ручным

и хранит неструктурированный (или полуструктурированный markdown) текст. MaaS определяет **автоматизированный, многоагентный конвейер**, предназначенный для создания высоко **структурированных, символических знаний**.
    - **Сотрудничество:** Хотя память проекта Claude может быть разделена внутри команды, процесс курирования не определен как формальный, многоагентный рабочий процесс. MaaS определяет **оркестрованный конвейер** специально для совместного построения знаний.
- **Уникальные преимущества/недостатки:**
    - **Преимущество:** Подход Claude обеспечивает максимальный **контроль, предсказуемость и прозрачность**.[23] Для профессионалов и разработчиков, работающих с конфиденциальными данными клиентов или с отдельными, несвязанными проектами, эта строгая изоляция является критически важной функцией, а не ограничением.
    - **Недостаток:** Сильная зависимость от ручного курирования накладывает высокую когнитивную нагрузку на пользователя и плохо масштабируется. Память системы так же хороша, как и усердие пользователя в ее обновлении, поскольку она не учится и не развивается самостоятельно.[27]

Архитектура памяти Claude, которая часто включает в себя вставку большого блока текста в окно контекста, подчеркивает критическую проблему современных LLM, известную как проблема «иголки в стоге сена». Хотя модели, такие как Claude, обладают очень большими окнами контекста (например, 200 000 токенов), их способность надежно находить конкретную, релевантную деталь в этом огромном контексте уменьшается по мере его роста.[27] Пользователи наблюдали это явление как «угасающую память»: по мере того, как файл
CLAUDE.md или резюме памяти становится больше и загроможденнее, способность модели точно вспоминать конкретные детали из него ухудшается. Это показывает, что просто наличие большого окна контекста не является панацеей от проблемы памяти. Структура, релевантность и краткость информации *внутри* контекста так же, если не более, важны. Это ограничение напрямую подтверждает архитектурные принципы RAG и концепции MaaS, которые отдают приоритет извлечению небольших, высокорелевантных фрагментов информации, а не без разбора загружают всю память в промпт.

Анализ системы IV: Агентные фреймворки (LangChain и Microsoft Autogen) — инструментарий разработчика

LangChain и Microsoft Autogen — это не готовые системы памяти, а мощные фреймворки с открытым исходным кодом, которые предоставляют разработчикам необходимые

компоненты для создания собственных, сложных архитектур памяти. Они представляют собой подход «собери сам», предлагая гибкость для реализации продвинутых паттернов, включая те, которые могут приближаться к концепции MaaS.

Основная архитектура: компонуемая и протокольно-ориентированная

Эти фреймворки предлагают модульный, протокольно-ориентированный подход к памяти, позволяя разработчикам подключать различные компоненты по мере необходимости.

- **Модульная память LangChain:** LangChain предоставляет богатую библиотеку компонуемых классов памяти, которые могут быть интегрированы в агентные рабочие процессы («цепочки» и «агенты»).[30] Ключевые архитектурные паттерны включают:
    - **Буферная память:** Простая кратковременная память для хранения последних сообщений в разговоре (например, ConversationBufferMemory, ConversationBufferWindowMemory).
    - **Память-резюме:** Использует LLM для периодического обобщения истории разговора, экономя токены в длительных взаимодействиях (ConversationSummaryBufferMemory).
    - **Память сущностей:** Извлекает и хранит структурированную информацию о конкретных сущностях (людях, организациях, концепциях), упомянутых в разговоре (ConversationEntityMemory).
    - **Память графа знаний:** Самый продвинутый паттерн, ConversationKGMemory, который динамически строит граф знаний из узлов (сущностей) и ребер (отношений) из разговора, позволяя выполнять сложные реляционные запросы.[30]
- **LangGraph для stateful-агентов:** LangGraph, расширение LangChain, специально разработан для создания stateful, многоагентных систем. Он представляет рабочие процессы в виде циклических графов, где память является явной и постоянной частью состояния графа, переносимой между шагами и даже целыми сессиями. Это делает его идеальной основой для реализации решений долговременной памяти.[33]
- **Протокол памяти Autogen:** Фреймворк Autogen от Microsoft предоставляет общий протокол Memory и продвигает RAG-центричный паттерн, где агенты запрашивают внешнее хранилище памяти для дополнения своего контекста.[35] Хотя он менее предвзят в отношении конкретной структуры памяти, он обеспечивает надежную основу для многоагентной коммуникации, необходимой для взаимодействия с такой системой. Фреймворк расширяем, с интеграциями, предоставляемыми сообществом, для векторных баз данных, таких как ChromaDB, и сторонних сервисов памяти, таких как Mem0.[35]

Процесс формирования памяти: логика, реализуемая разработчиком

В отличие от продуктовых систем от OpenAI и Anthropic, эти фреймворки не поставляются с автоматическим циклом обратной записи по умолчанию. Разработчик полностью отвечает за проектирование и реализацию этой критически важной логики в цикле рассуждений агента.[37] Формирование памяти обычно реализуется как инструмент или выделенный шаг в агентном рабочем процессе. Распространенный паттерн включает:

1. Агент завершает задачу или шаг в разговоре.
2. Затем агент вызывает инструмент «извлечения памяти» (который сам может быть другим вызовом LLM с определенным промптом) для обработки недавнего диалога и извлечения ключевых фактов, сущностей или отношений в структурированном формате.
3. Наконец, агент вызывает инструмент «хранения памяти» для записи этой структурированной информации в выбранную базу данных (например, векторное хранилище или графовую базу данных).

Этот определенный разработчиком процесс отражает оркестрованный многоагентный конвейер концепции MaaS, но должен быть построен с нуля.[34]

Инструменты и технологический стек

- **Основные фреймворки:** Python и JavaScript/TypeScript являются основными языками как для LangChain, так и для Autogen.[33]
- **Оркестрация:** LangChain, LangGraph и Microsoft Autogen предоставляют основные возможности оркестрации.
- **Подключаемые базы данных:** Ключевым преимуществом этих фреймворков является их независимость от баз данных. Разработчики могут интегрировать практически любую базу данных в качестве бэкенда памяти, включая:
    - **Векторные хранилища:** Pinecone, Chroma, Weaviate, LanceDB для семантической памяти.
    - **Графовые базы данных:** Neo4j, Memgraph, Kuzu для структурированной, реляционной памяти.[39]
    - **Хранилища ключ-значение/документов:** Redis, MongoDB для хранения данных сущностей или состояния сессии.

Сравнительный анализ с концепцией «MaaS»

- **Сходства:**
  - **Архитектурная философия:** Агентные фреймворки наиболее философски согласованы с концепцией MaaS. Оба в своей основе основаны на идее **оркестрованных, многоагентных систем**, где память является первоклассным компонентом, управляемым через определенный, программируемый процесс.[41] LangGraph, в частности, предоставляет stateful, циклическую структуру графа, необходимую для реализации рабочего процесса, подобного MaaS.
- **Различия:**
  - **«Создать» против «Купить»:** Критическое различие заключается в том, что MaaS определяется как полный, оркестрованный сервис, тогда как LangChain и Autogen — это **инструментарии для *создания*** такого сервиса. Они предоставляют необходимые компоненты (агенты, модули памяти, коннекторы к базам данных), но не готовый, полностью интегрированный конвейер.
- **Уникальные преимущества/недостатки:**
  - **Преимущество:** Эти фреймворки предлагают непревзойденную **гибкость и настраиваемость**. Разработчики могут выбрать точную структуру памяти, технологию базы данных и агентную логику, необходимую для их конкретного случая использования, что позволяет им создавать системы, которые намного сложнее и более адаптированы, чем любой готовый продукт.
  - **Недостаток:** Эта гибкость достигается за счет высокой **сложности реализации**. Создание надежной, масштабируемой и надежной системы долговременной памяти требует значительных усилий по разработке, глубокого архитектурного проектирования и отраслевых знаний. Разработчик несет полную ответственность за создание цикла обратной записи, управление состоянием и обеспечение согласованности данных.[37]

Эволюция паттернов памяти в этих фреймворках указывает на четкую траекторию развития отрасли. В то время как ранние системы памяти полагались на хранение необработанного текста и использование векторного поиска для извлечения, этот подход ограничен. Он может отвечать на вопросы на основе семантического сходства, но не справляется со сложными, реляционными запросами, такими как «Кто работает с Алисой над проектами, которые в настоящее время заблокированы?». Концепция MaaS определяет хранение структурированных, символических знаний, что по своей сути означает представление сущностей и их отношений. Графы знаний являются естественной и наиболее мощной структурой данных для этой цели, моделируя сущности как узлы, а отношения как ребра.[39] Растущая популярность компонентов, таких как
ConversationKGMemory от LangChain, и интеграция графовых баз данных (таких как Neo4j) в сервисы памяти [40] указывают на то, что отрасль выходит за рамки простого семантического поиска. Этот сдвиг в сторону графовой памяти представляет собой следующий рубеж, поскольку это самый многообещающий путь к достижению глубокого,

реляционного понимания, необходимого для действительно продвинутых, совместных агентов, и является наиболее близкой практической реализацией основных принципов концепции MaaS.

Синтез и стратегический прогноз

Анализ этих четырех различных подходов к долговременной памяти LLM показывает яркую и быстро развивающуюся картину. Каждая архитектурная философия предлагает уникальный набор компромиссов, отражающих различные приоритеты, которые варьируются от бесшовного удобства для пользователя до детального контроля разработчика, и от хранения неструктурированного текста до высокоструктурированного представления знаний.

Четыре философии: повторный взгляд

Четыре парадигмы можно расположить на спектре. На одном конце **память OpenAI** отдает приоритет беспроблемному пользовательскому опыту через глобальный, автоматизированный механизм персонализации. На противоположном конце **память Claude** отстаивает явный контроль пользователя и изоляцию данных через вручную курируемые, проектно-ориентированные контексты. Занимая более продвинутый, системный уровень, **MemGPT** рассматривает память как технический ресурс, которым должен автономно управлять агент, подобный ОС, для преодоления аппаратных ограничений. Наконец, **агентные фреймворки**, такие как LangChain и Autogen, предлагают мета-уровневый подход, предоставляя мощный, но сложный инструментарий для разработчиков для создания индивидуальных систем памяти, адаптированных к любым потребностям.
Следующая матрица представляет собой сравнительное резюме для помощи в принятии стратегических решений.

Характеристика	MemGPT	Память OpenAI	Память Claude	Агентные фреймворки (паттерн)	MaaS (эталон)
Основная парадигма	Виртуальный контекст в стиле ОС	Глобальная персонализация пользователя	Явное определение контекста	Инструментарий разработчика / Компонуемые компоненты	Оркестрованное многоагентное курирование знаний
Структура памяти	Неструктурированные текстовые	Гибридная: факты ключ-значени	Неструктурированное текстовое	Определяется разработчиком (вектор,	Структурированный символически

	уровни	е + семантическая история	резюме / Markdown	граф и т.д.)	й граф (факты, решения и т.д.)
Триггер формирования	Автоматический (давление на память)	Гибридный (команда пользователя + автоматическое извлечение)	Ручное курирование	Логика, реализуемая разработчиком	Автоматизированный многоагентный конвейер
Область применения	Пользователь / Агент	Глобальный пользователь	Проект / Команда	Специфично для приложения	Проект / Команда
Основная технология хранения	Векторная БД	Проприетарное KV + векторное хранилище	Проприетарная / Файловая система	Подключаемая (векторная БД, графовая БД и т.д.)	Графовая БД
Ключевое преимущество	Неограниченный контекст	Бесшовная персонализация	Максимальный контроль и изоляция данных	Абсолютная гибкость и настраиваемость	Глубокое реляционное понимание и сотрудничество
Ключевой недостаток	Когнитивная нагрузка / Неструктурированность	Риск утечки контекста	Высокая нагрузка на пользователя / Ручной труд	Высокая сложность реализации	Концептуальная / Высокая архитектурная нагрузка

Траектория развития памяти LLM

Эволюционный путь памяти LLM ясен. Он начался с простого наполнения контекста и базового RAG, который остается базовым уровнем. Следующий этап, примером которого является OpenAI, представил гибридные автоматизированные системы, которые сочетают явное хранение фактов с семантическим извлечением. Одновременно системные решения, такие как MemGPT, решали фундаментальную проблему конечного контекста. Однако современный уровень развития определяется агентными фреймворками, которые позволяют создавать полностью структурированные, определяемые разработчиком системы памяти. Растущее внедрение графов знаний в этих фреймворках знаменует собой последний и самый значительный шаг в этом прогрессе, переходя от простого извлечения информации к подлинному синтезу знаний.

Конвергенция к парадигме MaaS

Хотя ни одна система в настоящее время не воплощает в себе полное видение концепции «Память как услуга» (MaaS), тенденции всей отрасли сходятся на ее основных принципах. Автономный, самоуправляемый характер MemGPT, строгая проектная ориентация Claude, автоматическое извлечение фактов в системе OpenAI и мощные структурированные графовые компоненты, доступные в LangChain, — все это отдельные части головоломки MaaS. Будущее продвинутой, постоянной памяти для агентов LLM, несомненно, будет гибридной архитектурой, которая интегрирует эти разрозненные сильные стороны. Эта будущая система, вероятно, будет проектно-ориентированной, совместно управляемой конвейером специализированных агентов и построенной на временном графе знаний, что, наконец, позволит ИИ-системам не просто вспоминать информацию, но и понимать, связывать и рассуждать о ней во времени.

Источники

1. This article delves into MemGPT, a novel system developed by researchers at UC Berkeley to address the limited context window issue prevalent in Large Language Models (LLMs). By drawing inspiration from traditional operating system memory management, MemGPT introduces a hierarchical memory architecture allowing LLMs to handle extended contexts effectively. This piece explores the core concepts, implementation, evaluations, and the implications of MemGPT in advancing the capabilities of LLMs. - GitHub Gist, дата последнего обращения: сентября 28, 2025, https://gist.github.com/cywf/4c1ec28fc0343ea2ea62535272841c69
2. Inside MemGPT: An LLM Framework for Autonomous Agents ..., дата последнего обращения: сентября 28, 2025, https://towardsai.net/p/artificial-intelligence/inside-memgpt-an-llm-framework-for-autonomous-agents-inspired-by-operating-systems-architectures
3. MemGPT: Towards LLMs as Operating Systems - arXiv, дата последнего обращения: сентября 28, 2025, https://arxiv.org/pdf/2310.08560
4. what is memGPT?. making large language models better... | by michael raspuzzi | Medium, дата последнего обращения: сентября 28, 2025, https://michaelraspuzzi.medium.com/what-is-memgpt-cf344d88139f
5. Research Paper Summary: MemGPT Towards LLMs as Operating Systems - Medium, дата последнего обращения: сентября 28, 2025, https://medium.com/@Temitope_Oladokun/research-paper-summary-memgpt-towards-llms-as-operating-systems-c0f85ec8e73b
6. Revolutionizing Conversational AI with MemGPT's Context Expansion - Arun Addagatla, дата последнего обращения: сентября 28, 2025, https://arunaddagatla.medium.com/revolutionizing-conversational-ai-with-memgpts-context-expansion-0775e5299308
7. Unveiling MemGPT: The Evolution of Memory-Augmented Language Models | by

Ajay Verma | Artificial Intelligence in Plain English, дата последнего обращения: сентября 28, 2025, https://ai.plainenglish.io/unveiling-memgpt-the-evolution-of-memory-augmented-language-models-ebda3e762bcd
8. MemGPT: Unlocking Unlimited Memory for Conversational AI | by Tarek AbdELKhalek, дата последнего обращения: сентября 28, 2025, https://medium.com/@tiro2000/memgpt-unlocking-unlimited-memory-for-conversational-ai-6932002ea8e5
9. MemGPT with Real-life Example: Bridging the Gap Between AI and OS | DigitalOcean, дата последнего обращения: сентября 28, 2025, https://www.digitalocean.com/community/tutorials/memgpt-llm-infinite-context-understanding
10. MemGPT: OS inspired LLMs that manage their own memory - LanceDB Blog, дата последнего обращения: сентября 28, 2025, https://blog.lancedb.com/memgpt-os-inspired-llms-that-manage-their-own-memory-793d6eed417e/
11. MemGPT - Letta, дата последнего обращения: сентября 28, 2025, https://docs.letta.com/concepts/memgpt
12. MemGPT | Infinite Memory Superpower | Hands-on and In-Depth Discussion | Welcome to the Future - YouTube, дата последнего обращения: сентября 28, 2025, https://m.youtube.com/watch?v=p2GREqyxEYw
13. MemGPT, дата последнего обращения: сентября 28, 2025, https://research.memgpt.ai/
14. MemGPT: Towards LLMs as Operating Systems - UC Berkeley 2023 - Is able to create unbounded/infinite LLM context! : r/agi - Reddit, дата последнего обращения: сентября 28, 2025, https://www.reddit.com/r/agi/comments/17cojcp/memgpt_towards_llms_as_operating_systems_uc/
15. What is Memory? | OpenAI Help Center, дата последнего обращения: сентября 28, 2025, https://help.openai.com/en/articles/8983136-what-is-memory
16. How does ChatGPT's memory work? | Medium, дата последнего обращения: сентября 28, 2025, https://medium.com/@jay-chung/how-does-chatgpts-memory-feature-work-57ae9733a3f0
17. Memory in ChatGPT: How OpenAI's New Feature Creates Continuous Context and Transforms User Experience - Quantilus Innovation, дата последнего обращения: сентября 28, 2025, https://quantilus.com/article/memory-in-chatgpt-how-openais-new-feature-creates-continuous-context-and-transforms-user-experience/
18. Memory FAQ - OpenAI Help Center, дата последнего обращения: сентября 28, 2025, https://help.openai.com/en/articles/8590148-memory-faq
19. ELI5: How does ChatGPT's memory actually work behind the scenes? : r/OpenAI - Reddit, дата последнего обращения: сентября 28, 2025, https://www.reddit.com/r/OpenAI/comments/1jy3e6z/eli5_how_does_chatgpts_memory_actually_work/

20. OpenAI is Adding Memory Capabilities to ChatGPT to Improve Conversations - InfoQ, дата последнего обращения: сентября 28, 2025, https://www.infoq.com/news/2024/02/openai-chatgpt-memory/
21. Memory and new controls for ChatGPT - OpenAI, дата последнего обращения: сентября 28, 2025, https://openai.com/index/memory-and-new-controls-for-chatgpt/
22. Building ChatGPT-Like Memory: OpenAI's New Feature and How to Create Your Own | by Prasad Thammineni - Medium, дата последнего обращения: сентября 28, 2025, https://medium.com/agentman/building-chatgpt-like-memory-openais-new-feature-and-how-to-create-your-own-3e8e3594b670
23. Claude Memory: A Different Philosophy | Shlok Khemani, дата последнего обращения: сентября 28, 2025, https://shloked.com/writing/claude-memory
24. How Claude Solved the Memory Problem - Unite.AI, дата последнего обращения: сентября 28, 2025, https://www.unite.ai/how-claude-solved-the-memory-problem/
25. Claude introduces memory for teams at work \ Anthropic, дата последнего обращения: сентября 28, 2025, https://www.anthropic.com/news/memory
26. Claude Code's Memory: Working with AI in Large Codebases - Medium, дата последнего обращения: сентября 28, 2025, https://medium.com/@tl_99311/claude-codes-memory-working-with-ai-in-large-codebases-a948f66c2d7e
27. Claude Memory: A Deep Dive into Anthropic's Persistent Context ..., дата последнего обращения: сентября 28, 2025, https://skywork.ai/blog/claude-memory-a-deep-dive-into-anthropics-persistent-context-solution/
28. Comparing the memory implementations of Claude and ChatGPT - Simon Willison's Weblog, дата последнего обращения: сентября 28, 2025, https://simonwillison.net/2025/Sep/12/claude-memory/
29. How to use AI Memory in Claude desktop (and why It changes everything), дата последнего обращения: сентября 28, 2025, https://pieces.app/blog/ai-memory-for-claude
30. Advanced Memory in LangChain - Comet, дата последнего обращения: сентября 28, 2025, https://www.comet.com/site/blog/advanced-memory-in-langchain/
31. Building LangChain Applications: From Basics to Advanced Patterns — II | by Badru Siddique | Sep, 2025 | Medium, дата последнего обращения: сентября 28, 2025, https://medium.com/@badru.siddique_1465/building-langchain-applications-from-basics-to-advanced-patterns-ii-a256d6181023
32. Understanding LangChain Memory and Its Use Cases - Rajiv Gopinath, дата последнего обращения: сентября 28, 2025, https://www.rajivgopinath.com/real-time/trending/technology/understanding-langchain-memory-and-its-use-cases
33. LangGraph - LangChain, дата последнего обращения: сентября 28, 2025,

https://www.langchain.com/langgraph
34. A Long-Term Memory Agent | 🦜 LangChain, дата последнего обращения: сентября 28, 2025, https://python.langchain.com/docs/versions/migrating_memory/long_term_memory_agent/
35. Memory and RAG — AutoGen - Microsoft Open Source, дата последнего обращения: сентября 28, 2025, https://microsoft.github.io/autogen/stable//user-guide/agentchat-user-guide/memory.html
36. Mem0: Long-Term Memory and Personalization for Agents | AutoGen 0.2, дата последнего обращения: сентября 28, 2025, https://microsoft.github.io/autogen/0.2/docs/ecosystem/mem0/
37. AutoGen 0.4 Unpacked: A Thorough Analysis and a Wishlist for What's Next - Medium, дата последнего обращения: сентября 28, 2025, https://medium.com/@writetopavan/autogen-0-4-unpacked-a-thorough-analysis-and-a-wishlist-for-whats-next-058f5e4d8e75
38. microsoft/autogen: A programming framework for agentic AI - GitHub, дата последнего обращения: сентября 28, 2025, https://github.com/microsoft/autogen
39. Modeling Agent Memory - Graph Database & Analytics - Neo4j, дата последнего обращения: сентября 28, 2025, https://neo4j.com/blog/developer/modeling-agent-memory/
40. Graph Memory - Mem0 Documentation, дата последнего обращения: сентября 28, 2025, https://docs.mem0.ai/open-source/graph_memory/overview
41. Multi-agent - Docs by LangChain, дата последнего обращения: сентября 28, 2025, https://docs.langchain.com/oss/python/langchain/multi-agent
42. Memory Proposal · Issue #4564 · microsoft/autogen - GitHub, дата последнего обращения: сентября 28, 2025, https://github.com/microsoft/autogen/issues/4564
43. Building AI Agents with Knowledge Graph Memory: A Comprehensive Guide to Graphiti | by Saeed Hajebi | Medium, дата последнего обращения: сентября 28, 2025, https://medium.com/@saeedhajebi/building-ai-agents-with-knowledge-graph-memory-a-comprehensive-guide-to-graphiti-3b77e6084dec