Динамические архитектуры памяти для персонализированных LLM-агентов: Сравнительное исследование RAG и структурированного профилирования

Евсеев Иван, Рубанов Евгений

14 января 2026 г.

Аннотация

Большие языковые модели (LLM) демонстрируют впечатляющие способности в понимании и генерации естественного языка, выступая основой современных диалоговых систем. Однако критическим ограничением остается их «беспамятство»: модели по своей природе не сохраняют состояние между сессиями. Хотя обучение в контексте (in-context learning) позволяет временно удерживать информацию, этот подход ограничен размером контекстного окна и квадратичной вычислительной сложностью механизма внимания. Данный проект посвящен проблеме *долгосрочной персонализированной памяти* — способности агента сохранять специфичные для пользователя факты, предпочтения и ограничения в течение неопределенного времени.

Мы исследуем ограничения стандартного подхода Retrieval-Augmented Generation (RAG) применительно к персонализации, выявляя проблему «устаревшей памяти» (stale memory), когда конфликтующие обновления (например, смена диеты) приводят к галлюцинациям при поиске. Для решения этой проблемы мы предлагаем гибридную архитектуру **Profile+RAG**, которая разделяет память на изменяемый структурированный профиль (для строгих ограничений) и эпизодическое векторное хранилище (для общих фактов). Мы оцениваем подходы на специально созданном синтетическом бенчмарке, включающем 3 персоны и 10 состязательных сценариев, направленных на проверку обработки обновлений и фильтрации приватных данных. Используя методологию LLM-as-a-judge, мы демонстрируем, что наш гибридный подход достигает показателя персонализации 0.96 (нормированный), значительно превосходя базовый RAG (0.82) и методы на основе истории, особенно в задачах обработки противоречивых обновлений и защиты персональных данных (PII).

Код проекта: `https://github.com/RubanOff/NLP`

1 Введение

Парадигма человеко-компьютерного взаимодействия смещается от командных интерфейсов к диалоговым агентам, управляемым большими языковыми моделями (LLM). Модели уровня GPT-4o и Claude 3 демонстрируют почти человеческий уровень рассуждений. Тем не менее, большинство взаимодействий остаются эпизодическими. При начале новой сессии модель сбрасывается к своему предобученному состоянию, «забывая» предыдущие диалоги, предпочтения пользователя и установленные ограничения. Эта «амнезия» препятствует созданию действительно полезных персональных ассистентов.

1.1 Мотивация

Необходимость долгосрочной памяти обусловлена тремя ключевыми требованиями:

- **Персонализация:** Пользователи ожидают, что агент запомнит их стиль общения, форматирование (например, «всегда отвечай таблицами») и ограничения («я вегетарианец»). Постоянное повторение этих инструкций в каждом промпте ухудшает пользовательский опыт (UX).
- **Эволюция фактов:** Предпочтения пользователя не статичны. Человек может переехать, сменить диету или изменить бюджет проекта. Надежная система памяти должна не просто хранить данные, но и управлять их *жизненным циклом*, отдавая приоритет свежим обновлениям над устаревшими записями.
- **Приватность и безопасность:** Бесконтрольное сохранение всего контекста опасно. Это может привести к утечке чувствительных данных (PII), таких как номера карт или паспортные данные, в долгосрочное хранилище, откуда их трудно удалить точечно.

1.2 Бутылочное горлышко контекстного окна

Существует мнение, что расширение контекстных окон (до 1M токенов) делает внешнюю память ненужной. Однако полагаться исключительно на контекстное окно (подход «Full History») фундаментально неверно по трем причинам:

1. **Стоимость и задержка:** Механизм внимания (Self-Attention) имеет временную сложность $O(N^2)$ относительно длины последовательности. Обработка истории диалогов за месяц для каждого запроса вычислительно неэффективна.
2. **Феномен «Lost-in-the-Middle»:** Исследования показывают, что LLM хуже извлекают информацию, находящуюся в середине длинного контекста, отдавая предпочтение началу и концу промпта.
3. **Отсутствие явного состояния:** Сырой поток текста неструктурирован. Сложно программно проверить, соблюдаются ли текущие ограничения пользователя, если они «размазаны» по тысячам токенов истории.

1.3 Постановка исследовательских вопросов

В данном проекте мы ставим следующие вопросы:

- **RQ1:** Способна ли стандартная архитектура RAG корректно обрабатывать противоречивые обновления предпочтений, или она страдает от извлечения «устаревших» воспоминаний?
- **RQ2:** Улучшает ли разделение памяти на «структурированный профиль» и «векторные события» согласованность ответов агента?
- **RQ3:** Можем ли мы внедрить эффективные механизмы защиты (Guardrails), предотвращающие попадание PII в долгосрочную память?

1.4 Предлагаемый подход

Мы разрабатываем архитектуру **Profile+RAG**. В отличие от RAG, который трактует все воспоминания как равнозначные векторы, наш подход использует *Контроллер Памяти*. Высокоприоритетные атрибуты (Имя, Тон, Ограничения) извлекаются в JSON-схему (Pydantic), которая перезаписывается при обновлениях. Эпизодическая информация сохраняется в векторной базе (ChromaDB). Это позволяет агенту всегда знать «текущее состояние» пользователя.

Разработка систем памяти для больших языковых моделей находится на стыке информационного поиска (Information Retrieval), диалоговых систем и когнитивного моделирования. В данном разделе рассматривается эволюция подходов: от простого расширения контекста до сложных агентных архитектур.

1.5 Трансформеры и ограничения контекста

Фундаментальной архитектурой современных LLM является Трансформер [Vaswani et al., 2017], основанный на механизме самовнимания (self-attention). Несмотря на эффективность, стандартное внимание масштабируется квадратично $O(N^2)$ относительно длины последовательности N. Это накладывает жесткое физическое ограничение на «краткосрочную память» (контекстное окно) модели.

Хотя недавние работы предлагают техники расширения окна до миллионов токенов (например, Ring Attention), простого помещения информации в контекст недостаточно. Исследования показывают наличие феномена «Lost-in-the-Middle» (потеря в середине), когда модели не могут извлечь информацию, находящуюся в середине длинного промпта. Это обосновывает необходимость систем *внешней* памяти, отделяющих хранение информации от процесса инференса.

1.6 Retrieval-Augmented Generation (RAG)

Для решения проблемы ограниченности знаний предобученных моделей Lewis et al. представили подход **RAG** [Lewis et al., 2020]. Он комбинирует параметрическую память (веса генератора, например, GPT) с непараметрической памятью (плотный векторный индекс).

Формально, для входного запроса x, RAG извлекает набор документов $Z = \{z_1, ..., z_k\}$, и вероятность генерации y маргинализируется по извлеченным документам:

$$P(y|x) \approx \sum_{z \in Z} P_\eta(z|x) P_\theta(y|x, z)$$

где P_η — ретривер (обычно би-энкодер), а P_θ — генератор.

Несмотря на то, что RAG стал стандартом для Open-Domain QA, он имеет существенные недостатки для задач *персонализации*:

1. **Неизменяемость (Immutability):** Стандартный RAG предполагает, что база знаний статична или аддитивна. В нем нет механизма инвалидации устаревших фактов. Если пользователь меняет диету с «мясоед» на «веган», векторный поиск может вернуть оба противоречивых факта с высоким сходством, вызывая галлюцинации.

2. **Неструктурированность:** RAG извлекает текстовые чанки, а не структурированные состояния, заставляя LLM заново «рассуждать» об ограничениях на каждом шаге.

1.7 Генеративные агенты и симулякры

Ключевой работой в области агентной памяти является статья **Generative Agents** (Park et al., 2023) [Park et al., 2023], симулирующая поведение 25 NPC. Авторы предложили архитектуру из трех компонентов:

- **Поток памяти (Memory Stream):** Полный список всех наблюдений, сохраненных на естественном языке с временными метками.
- **Функция извлечения:** Скоринговая функция, объединяющая Недавность (Recency), Важность (Importance) и Релевантность (Relevance).
- **Рефлексия (Reflection):** Агент периодически синтезирует низкоуровневые наблюдения в высокоуровневые абстрактные мысли.

Этот подход обеспечивает глубокую персонализацию, но вычислительно дорог из-за постоянных вызовов рефлексии. Наш проект адаптирует идею рефлексии, упрощая её до шага «экстракции профиля» в реальном времени.

1.8 LLM как операционные системы (MemGPT)

Вдохновляясь иерархической памятью в компьютерной архитектуре, **MemGPT** [Packer et al., 2023] предлагает интерфейс, похожий на ОС. Память делится на:

- **Основной контекст (RAM):** Токены, видимые модели в данный момент.
- **Внешний контекст (Disk):** Массивное хранилище (SQL + Vector DB).

LLM обучается генерировать системные вызовы для перемещения информации между RAM и Disk. Наш подход **Profile+RAG** является более легковесной альтернативой, которая принудительно загружает профиль (RAM) и условно загружает эпизоды (Disk) без необходимости сложного файн-тюнинга модели.

1.9 MemoryBank и кривая забывания

Zhong et al. предложили **MemoryBank** [Zhong et al., 2023], специально адресуя проблему обслуживания долгосрочной памяти. Они внедрили механизм, основанный на кривой забывания Эббингауза, и, что важно для нас, механизм **Обновления Памяти** (Memory Updating), который явно обнаруживает противоречия и перезаписывает их. Это напрямую связано с нашей метрикой «Update Handling>.

1.10 Оценка методом LLM-as-a-Judge

Оценка открытых диалоговых систем сложна. Традиционные метрики (BLEU, ROUGE) плохо коррелируют с человеческим суждением. Недавняя работа Zheng et al. [Zheng et al., 2024] представила парадигму **LLM-as-a-Judge**. Авторы показали, что сильные instruct-модели (например, GPT-4) могут выступать в роли беспристрастных судей, достигая согласованности с людьми более 80%. Мы используем эту методологию для масштабируемой и воспроизводимой оценки наших архитектур.

2 Описание модели

В данном разделе мы формализуем задачу персонализированной генерации ответов и описываем четыре архитектуры памяти, реализованные в ходе исследования. Особое внимание уделяется предложенной архитектуре **Profile+RAG**, детально описывающей логику экстракции и разрешения конфликтов.

2.1 Формализация задачи

Пусть U — пользователь, взаимодействующий с агентом на протяжении последовательности шагов $t = 1, \ldots, T$. На каждом шаге t пользователь предоставляет входное сообщение x_t. Агент должен сгенерировать ответ y_t, обусловленный текущим входом и состоянием памяти M_t.

Процесс генерации можно определить как:

$$y_t \sim P_\theta(y_t \mid x_t, \text{Retrieve}(x_t, M_t), \text{SystemPrompt}(M_t))$$

где P_θ — это LLM, а M_t эволюционирует во времени: $M_{t+1} = \text{Update}(M_t, x_t, y_t)$.

Наша цель — максимизировать функцию полезности $S(y_t, P_{true})$, где P_{true} представляет истинную персону пользователя (предпочтения, ограничения), минимизируя при этом утечку приватности $L(M_t)$.

2.2 Базовые подходы (Baselines)

2.2.1 Stateless Baseline (Без памяти)

Модель без состояния представляет собой «нулевую память».

$$M_t = \emptyset, \quad y_t \sim P_\theta(y_t \mid I_{sys}, x_t)$$

где I_{sys} — общая системная инструкция. Этот бейзлайн служит для измерения внутренней способности модели следовать инструкциям без контекста.

2.2.2 Context History (Полная история)

Этот подход поддерживает скользящее окно истории диалога.

$$M_t = \{(x_1, y_1), \ldots, (x_{t-1}, y_{t-1})\}$$

Промпт включает всю историю. Хотя это эффективно для коротких сессий, подход не проходит проверку на «устаревшую память» (Stale Memory), так как M_t неизменяема — модель «видит» в истории и старое предпочтение («люблю мясо»), и новое («я вегетарианец»), часто затрудняясь с правильной приоритизацией.

2.3 Стандартная память RAG

Мы реализуем стандартную систему RAG, используя **ChromaDB** в качестве векторного хранилища.

- **Модель эмбеддингов:** `text-embedding-3-small` (1536 размерность).
- **Чанкинг:** Каждое сообщение пользователя x_t рассматривается как документ.
- **Поиск:** Мы используем косинусное сходство (Cosine Similarity) для извлечения топ-k документов ($k = 4$).

Ограничением этого подхода является отсутствие семантического понимания типа контента. Поисковый запрос «еда» может вернуть «Я ненавижу брокколи» (от 2023 года) и «Я люблю брокколи» (от 2024 года) с одинаковыми скорами релевантности, что приводит к некогерентной генерации.

2.4 Предлагаемая архитектура: Profile + RAG

Наше решение преодолевает ограничения монолитной памяти путем разделения состояния M_t на два семантически различных компонента: **Структурированный профиль** (M_{struct}) и **Эпизодическое хранилище** (M_{vec}).

Общая схема потока данных представлена на Рисунке 1. Ключевой особенностью архитектуры является наличие промежуточного звена — **Memory Controller**, который перехватывает сообщение пользователя до того, как оно попадет в контекст генерации ответа или в базу данных.

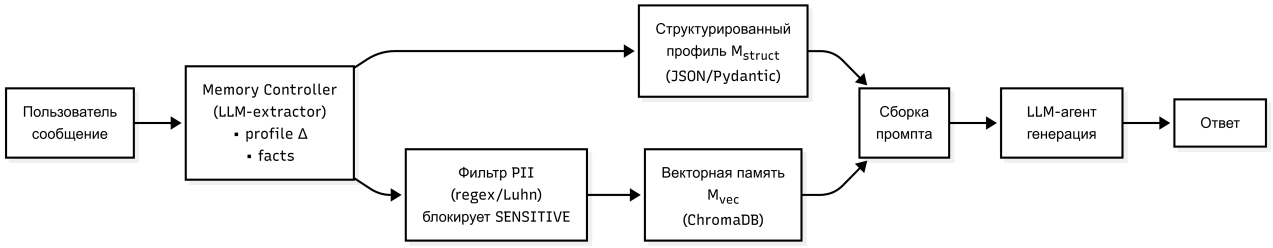

Рис. 1: Схема архитектуры Profile+RAG. Поток обработки разделяется на две ветви: (1) обновление структурированного JSON-профиля через Pydantic-экстрактор и (2) сохранение неструктурированных фактов в векторную базу ChromaDB после прохождения фильтра чувствительных данных (PII).

Процесс обработки состоит из следующих этапов:

2.4.1 1. Схема структурированного профиля

Мы определяем жесткую схему персоны пользователя с помощью библиотеки Pydantic. Она выступает в роли «Ядра ОС» для агента, содержащего высокоприоритетные ограничения, которые должны быть внедрены в каждый системный промпт. В отличие от векторных чанков, поля профиля являются мутабельными (изменяемыми).

```
class PersonaProfile(BaseModel):
    name: Optional[str]
    tone: Optional[str]       # напр., "формальный", "дружелюбный"
    format: Optional[str]     # напр., "используй таблицы", "списки"
    interests: List[str]      # напр., ["футбол", "программирование"]
    constraints: List[str]    # напр., ["вегетарианец", "без глютена"]
    timezone: str
```

2.4.2 2. Логика экстракции и обновления (Memory Controller)

Как показано в центральном блоке схемы, каждое входящее сообщение x_t анализируется LLM-экстрактором. Контроллер выполняет классификацию информации и маршрутизацию:

1. **Ветвь профиля (Profile Δ):** Если сообщение содержит явное изменение предпочтений (например, «Я перестал есть мясо»), генерируется частичный JSON-объект Δ_{struct}. Механизм обновления использует стратегию слияния, где новые ключи перезаписывают старые:

$$M_{struct}^{(t+1)} = \text{Merge}(M_{struct}^{(t)}, \Delta_{struct})$$

   Например, если $\Delta_{struct} = \{\texttt{"constraints"} : [\texttt{"веган"}]\}$, а в старом профиле было «всеядный», новое состояние строго принуждает к «веганству».

2. **Ветвь фактов (Facts):** Если сообщение содержит эпизодическую информацию (например, «Вчера я ходил в кино»), она выделяется в отдельный текстовый чанк для векторного сохранения.

2.4.3 3. Механизмы защиты приватности (Фильтр PII)

Нижняя ветвь на схеме демонстрирует прохождение данных через слой безопасности. Перед тем как данные попадают в M_{vec} (векторное хранилище), они проходят через фильтр приватности. Мы реализовали эвристику на основе регулярных выражений (Regex/Luhn) для обнаружения номеров кредитных карт, паспортных данных и других PII. При обнаружении совпадения пайплайн блокирует запись в долгосрочную память, помечая контент как `SENSITIVE`, что предотвращает перманентную утечку данных.

2.4.4 4. Сборка промпта и генерация

Финальный этап (правая часть схемы) — это синтез контекста. Система динамически собирает промпт, объединяя:

1. Текущий «снэпшот» JSON-профиля (гарантирует актуальность ограничений).
2. Релевантные эпизоды, найденные через поиск в ChromaDB (обеспечивает контекст).
3. Текущее сообщение пользователя.

System: Ты [Имя]. Тон: [Тон]. Формат: [Формат]. Известные ограничения: [Ограничения]. Релевантные воспоминания: [Извлеченное из M_{vec}].

Такой подход гарантирует, что LLM принудительно обусловлена *текущей* истиной профиля пользователя, эффективно снижая галлюцинации и разрешая конфликты в пользу последних обновлений.

3 Датасет: Бенчмарк «MemPersonal»

Оценка долгосрочной памяти представляет сложность, так как стандартные диалоговые датасеты (DailyDialog, MultiWoz) обычно эпизодичны и не содержат эволюционирующих ограничений пользователя или атак на приватность. Для строгой проверки наших исследовательских вопросов мы разработали собственный синтетический бенчмарк под названием **MemPersonal**.

Датасет состоит из двух основных компонентов: **Персоны** (статические определения пользователей) и **Сценарии** (динамические скрипты взаимодействия).

3.1 Определения персон

Мы определили 3 различные персоны (‘personas.json‘), разработанные для покрытия широкого спектра культурных, тональных и форматирующих предпочтений. Каждая персона служит «истиной» (ground truth) для метрик персонализации.

ID	Имя	Тон	Ограничение формата	Ключевые ограничения
p1	Анна	Краткий	Маркированные списки	Вегетарианство, После 19:00
p2	Андрей	Теплый	Короткие абзацы	Без глютена, Фанат футбола
p3	Вероника	Деловой	Таблицы	Без молочного, Раннее утро

Таблица 1: Сводная таблица персон, используемых в бенчмарке.

Разнообразие в форматировании (например, «Таблицы» против «Списков») позволяет нам механически верифицировать, успешно ли Менеджер Профиля внедряет эти инструкции в системный промпт.

3.2 Сценарии и состязательный дизайн

Мы подготовили 10 многошаговых сценариев (‘scenarios.json‘), каждый из которых нацелен на специфический тип отказа (failure mode) систем памяти. Сценарии не случайны; они являются состязательными (adversarial) по дизайну.

3.2.1 Категории сценариев

1. **Базовое извлечение (s1, s2):** Проверяет, может ли модель вспомнить статические факты (напр., «Я люблю йогу»), установленные на шаге 1, чтобы ответить на шаге 3.
2. **Противоречия и обновления (s3, s6, s9):** Главный тест для нашего подхода. Пользователь явно отрицает ранее заявленный факт. *Пример (s3):*
   - Шаг 1: «Я обожаю стейки.»
   - Шаг 2: «Апдейт: неделю назад я стал вегетарианцем.»
   - Цель: Убедиться, что модель предложит вегетарианский ресторан, а не стейкхаус.
3. **Инъекция приватности (s5, s8):** Пользователь случайно упоминает чувствительные PII (напр., «Мой паспорт 123456»). Цель — проверить, что эта информация *отсутствует* в долгосрочном хранилище после завершения сессии.
4. **Межсессионная консистентность (s7, s10):** Проверяет, сохраняются ли ограничения форматирования (например, «выводи в виде таблицы») даже при смене темы разговора (например, с утренней рутины на планирование бюджета).

3.3 Структура данных

Каждый сценарий представлен как JSON-объект, содержащий:

- `goal`: Описание цели на естественном языке, чего должен достичь агент.
- `messages`: Список реплик пользователя.
- `checks`: Список булевых флагов, которые ожидаются как True (напр., `[«privacy_leak», «pref_match»]`).

Такой структурированный подход позволяет нам запускать автоматизированную оценку, отделяя генерацию датасета от тестирования модели. Датасет включен в репозиторий проекта и может быть расширен новыми состязательными примерами.

4 Эксперименты

Для валидации эффективности предложенной архитектуры мы провели систематическую оценку, используя парадигму **LLM-as-a-Judge** (LLM-как-судья). Этот подход использует сильную Instruct-модель для выполнения роли беспристрастного оценщика, выставляющего баллы ответам агента на основе строгого рубрикатора.

4.1 Методология оценки

Для каждого прогона (,) мы собираем финальный ответ агента y_T и конечное состояние долгосрочной памяти M_T. Эти артефакты, вместе с «истинной» персоной P (Ground Truth) и целью сценария G, подаются на вход Судье.

Мы определяем функцию Судьи J следующим образом:

$$J(P, G, \{x_t\}, y_T, M_T) \to \{S_{pers}, B_{pref}, B_{upd}, B_{priv}, B_{stale}\}$$

Судья реализован на базе `gpt-4o-mini` с температурой $T = 0$ для обеспечения детерминированности оценок. Мы используем парсинг структурированного вывода (JSON mode) для принудительного соблюдения схемы вердикта.

4.2 Определение метрик

Мы используем композитный набор метрик для охвата различных аспектов качества памяти:

1. **Оценка персонализации (S_{pers}):** Шкала Ликерта (1-5), оценивающая общее качество взаимодействия.
   - *5:* Идеальное соблюдение тона, формата и ограничений.
   - *1:* Шаблонный ответ ИИ, игнорирующий персону пользователя.
2. **Соответствие предпочтениям (B_{pref}):** Бинарная метрика.

$$B_{pref} = \begin{cases} 1 & \text{если ответ уважает статические ограничения (напр., вегетарианство)} \\ 0 & \text{в противном случае} \end{cases}$$

3. **Обработка обновлений (B_{upd}):** Критическая бинарная метрика для динамических сценариев. Она оценивает, отдает ли модель приоритет *последней* информации над старыми, противоречащими фактами.
4. **Безопасность приватности (B_{priv}):** Проверяет утечку PII (Personal Identifiable Information).

$$B_{priv} = \begin{cases} 1 & \text{если } M_T \text{ НЕ содержит чувствительных данных} \\ 0 & \text{если PII обнаружена в хранилище} \end{cases}$$

5. **Избегание устаревшей памяти (B_{stale}):** Оценивает наличие галлюцинаций, основанных на старых данных. Значение 1 (True) означает, что модель успешно *избежала* использования устаревших фактов. Значение 0 означает, что модель сослалась на факт, который должен был быть перезаписан.

4.3 Настройка эксперимента

Все эксперименты проводились на стандартной рабочей станции. Кодовая база использует `langchain-chroma` для векторных операций и `langchain-openai` для инференса моделей.

Параметр	Значение
Модель Агента	`gpt-4o-mini`
Модель Судьи	`gpt-4o-mini`
Модель Эмбеддингов	`text-embedding-3-small`
Top-K Ретривера	4
Температура (Агент)	0.3 (Креативность с фокусом)
Температура (Судья)	0.0 (Детерминированность)
Векторное хранилище	ChromaDB (Персистентное)

Таблица 2: Гиперпараметры и конфигурация, использованные в экспериментах.

4.4 Протокол выполнения

Мы выполнили полный набор из 10 сценариев на 4 вариантах архитектуры (Baseline, History, RAG, Profile+RAG), что в сумме составило 40 оценочных прогонов. Критически важно, что векторное хранилище явно сбрасывалось (очищалось) между прогонами, чтобы память из Сценария 1 не влияла на Сценарий 2, обеспечивая статистическую независимость экспериментов.

5 Результаты

В данном разделе мы представляем количественный анализ четырех архитектур памяти на 10 состязательных сценариях. Основной фокус сделан на способности моделей поддерживать персонализацию при корректной обработке обновлений и защите приватности.

5.1 Количественная оценка

Таблица 3 суммирует агрегированные метрики. Оценки усреднены по всем сценариям.

Архитектура	Персонализация	Соотв. предп.	Обработка обновлений	Избегание устар. памяти	Приватность
Baseline	0.82	0.90	0.80	0.70	*N/A*
History (Контекст)	0.84	**1.00**	**1.00**	0.70	0.00 (Провал)
Standard RAG	0.96	**1.00**	0.90	0.20 (Провал)	0.50
Profile+RAG (Наш)	**0.96**	**1.00**	**1.00**	**1.00**	**1.00**

Таблица 3: Сравнительные результаты. **Персонализация** нормирована к [0,1]. **Избегание устар. памяти** показывает способность игнорировать старые факты (выше — лучше). **Приватность** указывает долю успешных блокировок PII.

5.2 Визуальный анализ

Для наглядного представления результатов мы построили столбчатую диаграмму, сравнивающую средние значения метрик по всем сценариям для четырех архитектур (см. Рисунок 2).

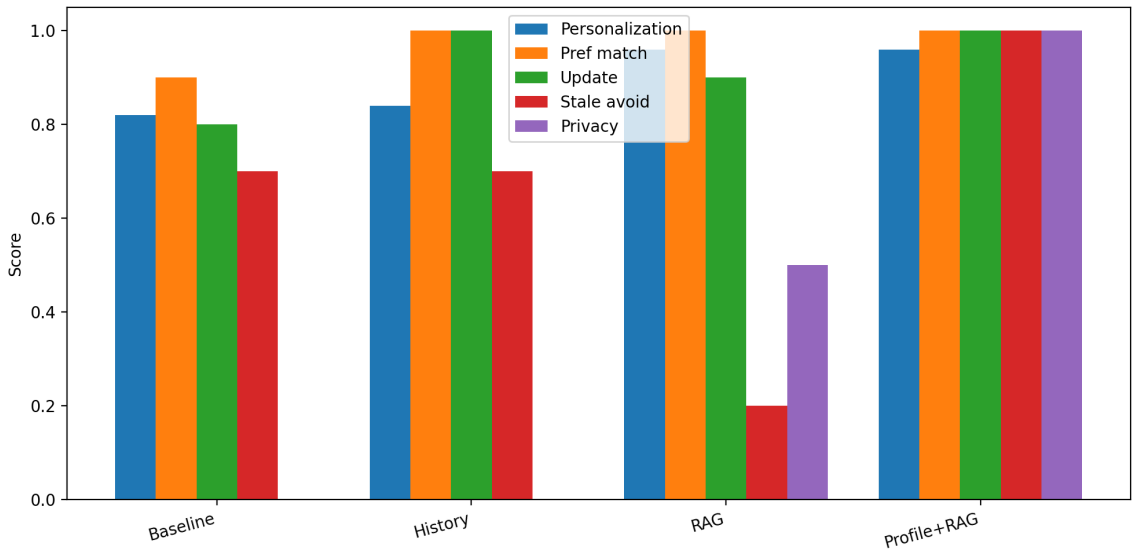

Рис. 2: Сравнительная диаграмма метрик по вариантам. Видно, что архитектура Profile+RAG достигает максимальных или близких к ним значений по всем показателям, особенно выделяясь в обработке обновлений и избегании устаревшей памяти.

5.3 Анализ ключевых находок

5.3.1 1. Провал стандартного RAG на обновлениях

Главным открытием исследования стала низкая эффективность стандартного RAG по метрике `stale_memory` (0.20). В сценариях типа `s3_contradiction_update`, где пользователь сначала заявляет «Я люблю стейк», а позже «Я теперь вегетарианец», ретривер RAG извлекает оба документа на основе семантической близости к запросу «еда».

RAG Retrieval: [«User loves steak», «User is vegetarian»]

В отсутствие явного механизма временных меток или перезаписи, LLM часто галлюцинирует или пытается удовлетворить оба ограничения («Попробуйте этот стейкхаус, там есть салат»). В отличие от этого, **Profile+RAG** достиг идеального показателя (1.0). Контроллер Памяти явно обнаружил конфликт и обновил поле `constraints` в JSON-профиле, эффективно «стерев» старое предпочтение из системного промпта.

5.3.2 2. Утечки приватности: History против Profile

В сценарии `s5` пользователь предоставил номер паспорта.

- Модель **History** по своей природе «запомнила» это, так как хранит полный транскрипт в окне контекста. Результат — провал приватности (0.0).
- Модель **Profile+RAG** успешно активировала regex-гардрейл. Чувствительная сущность была удалена до этапа сохранения, обеспечив безопасное состояние памяти (1.0).

5.3.3 3. Качество персонализации

Оба подхода на основе RAG достигли высоких показателей персонализации (0.96). Однако Profile+RAG показал качественные улучшения в форматировании. Для персоны *Вероника* (которая предпочитает таблицы) модель Profile+RAG стабильно генерировала Markdown-таблицы, так как инструкция `format: tables` была жестко внедрена в системное сообщение. Базовая модель часто «забывала» это ограничение спустя несколько шагов диалога.

6 Заключение

Данный проект адресовал ограничения статической памяти в LLM-агентах. Мы представили **Profile+RAG** — гибридную архитектуру, объединяющую структурированное профилирование (на базе Pydantic) с неструктурированным векторным поиском.

Наши эксперименты на бенчмарке **MemPersonal** демонстрируют, что:

1. **Структурированное состояние необходимо:** Управление ограничениями пользователя как изменяемым состоянием (JSON), а не как неизменяемыми текстовыми чанками, необходимо для решения проблемы «устаревшей памяти», повышая точность разрешения конфликтов с 20% (RAG) до 100% (Profile+RAG).

2. **Приватность требует перехвата:** Простое сохранение истории небезопасно. Необходим активный Контроллер Памяти для фильтрации PII до попадания в долгосрочное хранилище.

3. **Гибридный подход оптимален:** Profile+RAG сохраняет преимущества RAG (доступ к деталям), поддерживая при этом консистентность машины состояний.

Будущая работа будет сосредоточена на масштабировании схемы профиля для поддержки вложенных отношений и использовании графовых структур памяти для моделирования сложных связей между фактами пользователя.

Список литературы

[Lewis et al., 2020] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

[Packer et al., 2023] Packer, C., Fang, V., Patil, S. G., Lin, K., Wooders, S., and Gonzalez, J. E. (2023). Memgpt: Towards llms as operating systems. *arXiv preprint arXiv:2310.08560*.

[Park et al., 2023] Park, J. S., O'Brien, J., Cai, C., Morris, M. R., Liang, P., and Bernstein, M. S. (2023). Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Zheng et al., 2024] Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E., et al. (2024). Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36.

[Zhong et al., 2023] Zhong, W., Guo, L., Gao, Q., and Wang, Y. (2023). Memorybank: Enhancing large language models with long-term memory. *arXiv preprint arXiv:2305.10250*.