PROJECT RETROSPECTIVE · ИЮЛЬ 2026

TrifonAI — персональная локальная ИИ-платформа

История разработки: от одного Python-скрипта до агентской платформы с памятью, десятью инструментами, веб-интерфейсом и собственной иконкой в Dock. Написано как ретроспектива — с датами, багами и настоящими причинами.

ВЕРСИЯ	ИНСТРУМЕНТОВ	ДВИЖОК	ЖЕЛЕЗО
v1.2	10	qwen3:8b	M4 · 16 ГБ

1. Вступление

INTRODUCTION · WHAT TRIFONAI IS

TrifonAI — это не чат-бот. Чат-бот — это интерфейс к языковой модели; здесь всё наоборот. Языковая модель — сменная деталь внутри платформы, которая существует независимо от неё.

Ключевая философия проекта формулируется одной фразой: **интеллект, память и инструменты принадлежат платформе, а не модели**. Модель — это вычислительный движок, который сегодня `qwen3:8b` через Ollama, а завтра может быть любым другим. Долговременная память, характер, набор инструментов, правила поведения — постоянны и живут в коде и в базе, а не в весах модели. Заменить движок — не значит потерять ассистента.

Второй принцип — **полная локальность**. Всё работает на MacBook Air M4 с 16 ГБ памяти: модель, база данных, веб-сервер. Наружу не уходит ничего — единственное исключение появилось позже и осознанно: инструмент веб-поиска, который по определению обращается в интернет.

Проект начинался с другой нейросети в роли ассистента разработки. В какой-то момент контекст проекта был передан Claude — со всей историей, архитектурой и решениями, — и дальше работа шла уже с ним, сначала в редакторе Cursor, а с 10 июля 2026 года в Claude Code. Этот документ описывает путь целиком.

> **Почему это важно как инженерная позиция.** Если память и инструменты живут внутри промпта модели, проект умирает вместе с моделью. Если они живут в платформе — модель становится заменяемым компонентом, а накопленное остаётся. Все архитектурные решения ниже — следствие этого выбора.

2. Хронология развития

TIMELINE · HOW THE PROJECT GREW

до 10.07	Начальная архитектура: `main.py`, `models/qwen.py`, структура папок
10.07	`core/assistant.py` + SQLite-память (базовая, по ключевым словам)
10.07	Переход с Cursor на Claude Code — в тот же день, из-за лимита free-плана
10.07 веч.	Извлечение фактов через саму модель + смысловая дедупликация
10.07	Личность: `personality/system.md`
12.07	Инструменты, базовая версия: три штуки — время, список файлов, чтение файла
12.07	Расширение до шести (`write_file`, `search_files`, `recall_memory`), создан `CLAUDE.md`
13.07	Планировщик: задачи и напоминания
13.07	**Исходный план закрыт полностью — веха**
18.07	Обслуживание: удалена старая копия `~/main.py` (v0.2), версия v0.4 → v1.0
23.07	Веб-интерфейс v1: FastAPI + SSE, тёмная/светлая тема, анимации
25.07	Веб расширен: интерактив + `web_search` как десятый инструмент
25.07	Запуск одной иконкой: `~/Applications/TrifonAI.app`
25.07	Подтверждение записи файлов в веб-версии — починено

Глава 1. Начальная архитектура до 10.07.2026

Первая версия — Python и Ollama, и сразу с принципом, который дальше определил всё: **один файл = одна ответственность**. Не «скрипт, который со временем обрастёт», а модульная структура с самого начала: `main.py` как тонкий UI-цикл, `models/qwen.py` как единственное место, знающее про Ollama, отдельные папки под память, инструменты и личность.

Именно это решение позже сделало возможным всё остальное. Когда появился веб-интерфейс, терминальный `main.py` не пришлось переписывать — он остался одним из двух клиентов к одному и тому же ядру. Когда понадобилось искать бинарник Ollama в разных местах системы, менялся один файл.

Глава 2. Память `10.07.2026`

Память в TrifonAI живёт в SQLite (`memory/trifonai.db`) и состоит из двух разных сущностей: **история диалогов** — что говорилось, и **факты** — что из этого стоит помнить всегда.

Развитие механизма извлечения фактов шло тремя шагами, и каждый следующий появлялся потому, что предыдущий упирался в потолок:

1. **Ключевые слова.** Простое правило: если в сообщении встречается «меня зовут», «я работаю», «мне нравится» — сохранить как факт. Работало, но ловило единицы процентов и не понимало перефразировок.
2. **Извлечение через модель.** Тот же движок, который отвечает пользователю, отдельным вызовом решает: есть ли в этом сообщении факт, который стоит запомнить, и как его сформулировать. Качество выросло резко.
3. **Смысловая дедупликация.** Немедленно вылезла новая проблема: «я живу в Москве» и «мой город — Москва» — это один факт в двух формулировках, и строковое сравнение их не поймает. Решение — снова модель: перед сохранением она сравнивает новый факт с уже имеющимися по смыслу и решает, создавать запись или обновлять существующую.

Это, пожалуй, самое концептуальное решение проекта: *модель используется не только как собеседник, но и как инструмент внутри платформы* — как семантический компаратор, у которого нет альтернативы в детерминированном коде.

Глава 3. Смена инструмента разработки `10.07.2026`

Разработка шла в Cursor, пока не закончились лимиты бесплатного плана — в тот же день, 10 июля, произошёл переход на Claude Code, который дальше стал основным инструментом. Вместе с этим 12 июля появился `CLAUDE.md` — контекстный файл проекта, чтобы ассистент разработки при каждом запуске знал архитектуру, принципы и текущее состояние, а не восстанавливал их из переписки.

Глава 4. Личность `10.07.2026`

`personality/system.md` — отдельный файл, а не строка в коде. В нём идентичность (кто такой Трифон), характер, стиль общения и правила работы с памятью: когда обращаться к фактам, как их упоминать, чего не делать. Личность лежит рядом с кодом как редактируемый текст — её можно править, не касаясь логики, и она переживает смену модели.

Глава 5. Инструменты 12.07.2026 → 25.07.2026

Момент, когда ассистент перестал быть говорящей головой. Началось с трёх простых инструментов — текущее время, список файлов, чтение файла — и `tools/registry.py` как реестра с протоколом вызова. В тот же день набор дошёл до шести: запись файлов, поиск по файлам, доступ к собственной памяти.

Ключевое решение здесь — **запись файлов только с обязательным подтверждением человека**. Ассистент, который может писать на диск без спроса, — это не удобство, а риск, и подтверждение было заложено сразу, а не добавлено после инцидента. Позже именно это требование обернулось самой сложной инженерной задачей проекта (см. §3.6).

К 25 июля инструментов стало десять — с задачами и веб-поиском. Инструмент `get_current_time` по ходу был удалён: время оказалось правильнее подавать в контекст всегда, а не заставлять модель запрашивать его вызовом.

Глава 6. Планировщик 13.07.2026

Задачи и напоминания: `tools/tasks.py` для управления, таблица `tasks` в базе, `core/reminders.py` для показа актуальных напоминаний при старте. Простая с виду функция, породившая несколько раундов багов вокруг арифметики относительных дат — «завтра», «послезавтра». Подробности в §3.3.

> MILESTONE · 13.07.2026
>
> **Исходный план закрыт полностью.** Всё, что задумывалось в начале — локальная модель, долговременная память, личность, агентские инструменты, планировщик, — работало. Проект дошёл до состояния, в котором его можно было просто использовать. Всё дальнейшее — уже не выполнение плана, а его продолжение по собственной воле.

Глава 7. Обслуживание 18.07.2026

Небольшая, но показательная дата. Удалена старая копия `~/main.py` (v0.2), болтавшаяся в домашней папке и способная однажды запуститься вместо настоящей версии. Версия переименована из v0.4 в v1.0 — не косметика, а признание, что проект вышел из состояния черновика. Единый источник версии живёт в `core/version.py`, чтобы номер не расползался по файлам.

Глава 8. Веб-интерфейс 23.07 → 25.07.2026

Переход от терминала к браузеру: FastAPI-сервер (`ui/server.py`) и весь фронтенд одним файлом (`ui/static/index.html`) — тот же принцип минимальной структуры, что и в ядре.

Самое интересное здесь — **SSE-поток событий в реальном времени**. Веб-версия не просто показывает финальный ответ: видно, как ассистент думает. Вызов инструмента — событие. Момент срабатывания памяти — событие. Всё это приходит в браузер потоком и отображается вживую, а технические детали можно развернуть, если интересно, и свернуть, если нет. Плюс тёмная и светлая тема, аккуратные анимации и markdown-рендеринг ответов.

25 июля веб-версия получила интерактив и десятый инструмент — `web_search` через пакет `ddgs` (DuckDuckGo). Первое и единственное осознанное исключение из правила «ничего наружу».

Глава 9. Запуск одной иконкой 25.07.2026

Скрипты `scripts/start_web.sh`, `stop_web.sh` и `build_app.sh` превратили проект из «зайти в терминал, активировать окружение, запустить» в `~/Applications/TrifonAI.app` — полноценное macOS-приложение в Dock и Launchpad. Психологически это самый большой сдвиг во всём проекте: то, что запускается иконкой, перестаёт быть экспериментом и становится программой, которой пользуешься. Именно здесь вылезла грабля с macOS TCC (§3.5).

3. Технические трудности

ENGINEERING WAR STORIES · BUGS AND ROOT CAUSES

Самая содержательная часть пути. Здесь собраны баги, где путь от симптома до причины был неочевиден, а правильное решение требовало понимания настоящей причины, а не косметического патча.

3.1 ANSI escape-коды Ollama в сохранённом тексте

СИМПТОМ — В базе и в интерфейсе в тексте ответов попадался мусор — обрывки вроде управляющих последовательностей вокруг нормальных слов.

ГИПОТЕЗА — Проблема кодировки или битый вывод модели.

ПРИЧИНА — Ollama в CLI-режиме подмешивает в поток ANSI escape-коды — для терминала это управление курсором и цветом, для программы, читающей вывод, — часть строки. В терминале мусор был невидим, потому что терминал его исполнял, а не печатал. В базу он попадал как есть.

РЕШЕНИЕ — Отдельный слой очистки — `core/text.py`, функция `clean_model_output()`. Не патч в месте сохранения, а единая точка: всё, что приходит от модели, проходит через неё.

3.2 SQLite LOWER() и кириллица

СИМПТОМ — Поиск по памяти находил не всё: запрос «москва» не находил факт со словом «Москва».

ГИПОТЕЗА — Ошибка в SQL-запросе или в формировании условий поиска.

ПРИЧИНА — Встроенный `LOWER()` в SQLite по умолчанию работает только с ASCII — кириллические буквы он не приводит к нижнему регистру и оставляет как есть. SQL был корректен; неверным было предположение о его семантике.

РЕШЕНИЕ — Приведение регистра перенесено в Python, где Unicode работает как ожидается, — на стороне запроса, а не движка базы.

3.3 Арифметика относительных дат

СИМПТОМ «Напомни послезавтра» ставило напоминание на +3 дня вместо +2. Похожие сдвиги вылезали и на «завтра».

ГИПОТЕЗА Модель неправильно понимает слово.

ПРИЧИНА Двойной сдвиг: смещение прибавлялось и при разборе фразы, и при сохранении задачи. Плюс путаница в точке отсчёта — «от сегодня» против «от завтра». Баг возвращался несколько раз в разных формах, потому что каждый раз правился симптом в одном из двух мест.

РЕШЕНИЕ Единственная точка, где относительная дата превращается в абсолютную, и явная передача текущей даты в контекст. Урок: относительное время нельзя пересчитывать более одного раза за путь запроса.

3.4 clean_model_output() съедал переносы строк в списках

СИМПТОМ В веб-версии списки склеивались в один абзац: пункты шли подряд без переносов.

ГИПОТЕЗА Баг в markdown-рендеринге фронтенда — ведь в терминале всё было нормально.

ПРИЧИНА Ollama ставит два пробела в конце строки; эвристика в `clean_model_output()` принимала это за артефакт разрыва стриминга и убирала перенос. Важная деталь: баг был **общий** — он бил и терминал, и веб, но проявился только в вебе, потому что markdown-рендеринг требует переносов, а терминальный вывод к ним снисходителен.

РЕШЕНИЕ Эвристика уточнена: два пробела в конце строки перестали считаться признаком разрыва. Правка в ядре, а не во фронтенде — и терминальная версия тоже стала корректнее.

3.5 macOS TCC: разрешения привязаны к подписи процесса

СИМПТОМ После сборки `.app` инструменты работы с файлами перестали видеть содержимое папок, хотя «Полный доступ к диску» в системных настройках был выдан.

ГИПОТЕЗА Разрешение выдано не тому объекту — надо указать другую папку или переоткрыть настройки.

ПРИЧИНА Системная особенность macOS TCC: разрешения привязываются к **подписи процесса**, а не к папке `.app` и не к пути. Приложение-обёртка, запускающее скрипт, и сам процесс интерпретатора — с точки зрения TCC разные субъекты. Разрешение было выдано не тому, кто на самом деле обращался к диску.

РЕШЕНИЕ Разбираться с моделью безопасности системы, а не с кодом: понять, какой именно процесс TCC видит как субъект доступа, и выдавать право ему. Ни одной строкой Python это не лечилось.

3.6 input() не существует в браузере

СИМПТОМ Подтверждение записи файлов, работавшее в терминале, в веб-версии не работало вообще.

ПРИЧИНА Не баг, а архитектурное несоответствие. В терминале подтверждение — это `input()`: синхронная блокировка потока до ответа человека. В веб-запросе блокировать поток нельзя, а ответ придёт отдельным HTTP-запросом позже — возможно, из другого соединения.

РЕШЕНИЕ Полностью новый механизм: `core/confirm.py` как абстракция подтверждения с двумя реализациями. Запрос на запись получает **токен**, уходит в браузер SSE-событием, ждёт подтверждения на `/api/writes/{token}/confirm` и только тогда выполняется. Терминальная версия продолжила работать через `input()` — за той же абстракцией.

3.7 threading.local в многопоточном пуле Starlette

СИМПТОМ События терялись: часть вызовов инструментов и срабатываний памяти не доходила до браузера. Непредсказуемо — то доходили, то нет.

ГИПОТЕЗА Проблема на стороне SSE — обрыв соединения, буферизация, потеря в сети.

ПРИЧИНА Канал событий хранился в `threading.local`. Starlette выполняет синхронный код в пуле потоков, поэтому части одного запроса выполнялись в *разных* потоках — и код, публикующий событие, видел пустое локальное хранилище вместо канала. Событие уходило в никуда. Плавающий характер бага — прямое следствие того, какой поток из пула достался.

РЕШЕНИЕ Отказ от привязки контекста к потоку — контекст события передаётся явно, по границе запроса, а не через поток. Самая тонкая находка проекта: симптом указывал на транспорт, причина была в модели исполнения фреймворка.

3.8 Кеш браузера, который выглядел как баг

СИМПТОМ Правка внесена, сервер перезапущен — поведение старое. Похоже, что исправление не работает.

ПРИЧИНА Браузер отдавал закешированный `index.html`. Код был правильный с самого начала.

РЕШЕНИЕ Жёсткая перезагрузка — и вывод на будущее: прежде чем искать баг в логике, убедиться, что запущен именно тот код, который правился. Полдня диагностики может уйти на проблему, которой нет.

Сводка багов

БАГ	НАСТОЯЩАЯ ПРИЧИНА	РЕШЕНИЕ
Мусор в сохранённом тексте	ANSI escape-коды в CLI-выводе Ollama; терминал их исполнял, база — сохраняла	Единая точка очистки: `clean_model_output()`
Поиск по памяти пропускал факты	SQLite `LOWER()` не работает с кириллицей (ASCII-only)	Регистр приводится в Python, а не в SQL
«Послезавтра» = +3 дня	Двойное применение сдвига и расхождение точки отсчёта	Одна точка перевода в абсолютную дату; дата в контексте
Списки склеивались в абзац	Два пробела Ollama в конце строки принимались за разрыв стриминга	Уточнена эвристика в ядре — исправило и терминал, и веб
Нет доступа к диску у `.app`	macOS TCC привязывает права к подписи процесса, а не к папке `.app`	Право выдано реальному субъекту доступа, а не обёртке
Подтверждение записи не работало в вебе	Блокирующий `input()` физически невозможен в HTTP-запросе	Токены + SSE + отдельный эндпоинт подтверждения за общей абстракцией
События терялись по пути в браузер	`threading.local` в пуле потоков Starlette: разные потоки одного запроса	Контекст события передаётся явно, не через поток
Правка «не применялась»	Кеш браузера отдавал старый `index.html`; бага не было	Жёсткая перезагрузка; сначала проверять, что запущен нужный код

4. Архитектура сегодня

ARCHITECTURE · V1.2

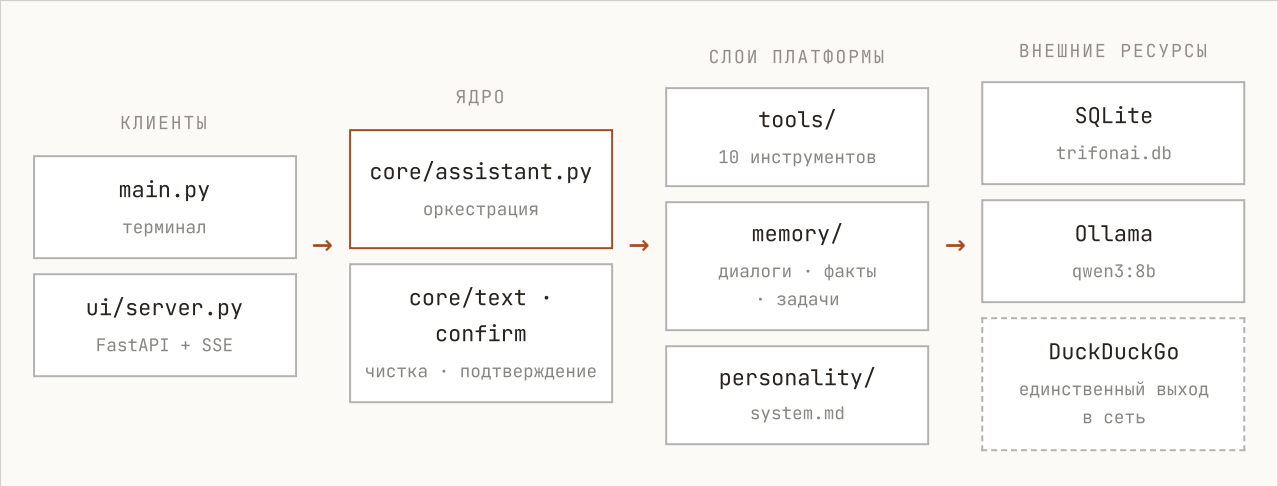

Рис. 1 — Слои платформы. Клиенты взаимозаменяемы; ядро одно.

Карта файлов

ПУТЬ	ОТВЕТСТВЕННОСТЬ
`main.py`	Терминальная версия — тонкий UI-цикл
`core/assistant.py`	Класс `Assistant`: оркестрация модели, памяти и инструментов
`core/text.py`	`clean_model_output()` — чистка ANSI-кодов Ollama
`core/memory_evaluator.py`	Извлечение фактов и смысловая дедупликация через модель
`core/reminders.py`	Показ актуальных напоминаний при старте
`core/confirm.py`	Абстракция подтверждения записи: терминал и веб
`core/version.py`	Единый источник версии (v1.2)
`memory/database.py`	SQLite: таблицы `conversations`, `facts`, `tasks`
`models/qwen.py`	Вызов Ollama; поиск бинарника по PATH с фолбэками
`tools/registry.py`	Реестр инструментов и протокол вызова
`personality/system.md`	Характер, идентичность, правила работы с памятью
`ui/static/index.html`	Весь веб-фронтенд одним файлом
`scripts/*.sh`	Запуск и остановка веб-версии, сборка `.app`
`CLAUDE.md`	Контекст проекта для Claude Code

Десять инструментов

ИНСТРУМЕНТ	МОДУЛЬ	ЧТО ДЕЛАЕТ
`list_files`	`tools/basic.py`	Содержимое папки
`read_file`	`tools/basic.py`	Чтение файла
`write_file`	`tools/basic.py`	Запись файла — **только после подтверждения человеком**
`search_files`	`tools/basic.py`	Поиск по файлам
`recall_memory`	`tools/basic.py`	Доступ к собственной памяти — фактам и истории
`add_task`	`tools/tasks.py`	Новая задача или напоминание
`list_tasks`	`tools/tasks.py`	Список активных задач
`complete_task`	`tools/tasks.py`	Закрыть задачу
`complete_all_tasks`	`tools/tasks.py`	Батч-закрытие — обход слабости модели в длинных цепочках вызовов
`web_search`	`tools/web.py`	Поиск в интернете через `ddgs` (DuckDuckGo)

`get_current_time` был одиннадцатым и удалён осознанно: текущее время подаётся в контекст всегда, отдельный вызов оказался лишним звеном и источником ошибок в датах.

Как работает память

1. Пользователь пишет сообщение → `Assistant` сохраняет реплику в `conversations`
2. `memory_evaluator` отдельным вызовом спрашивает модель: есть ли здесь факт, который стоит помнить?
3. Факт есть → из `facts` достаются похожие записи
4. Второй вызов модели — сравнение по смыслу, а не по строке: дубликат или новое?
5. Новое → `INSERT`; дубликат с новой формулировкой → `UPDATE` существующей записи
6. В веб-версии каждый шаг уходит в браузер SSE-событием — видно, как память срабатывает

Рис. 2 — Путь факта: извлечение → проверка дубликата → сохранение или обновление.

Путь сообщения в веб-версии

Рис. 3 — Поток одного сообщения через веб-версию.

5. Ограничения и компромиссы

KNOWN LIMITATIONS · HONESTLY

- **Маленькая модель не всегда держит стиль.** `qwen3:8b` периодически нарушает инструкции из `system.md` — вставляет эмодзи там, где не надо, или уходит в многословность. Цена локальности.
- **Не тянет длинные цепочки однотипных вызовов.** Десять последовательных вызовов одного инструмента модель не доводит до конца. Решение не в промпте, а в архитектуре: батч-инструменты вместо чейнинга — отсюда `complete_all_tasks`.
- **Нет мультимодальности.** Ассистент не видит изображения — только текст.
- **Несколько моделей одновременно — осознанно отложено.** Архитектура к этому готова (модель отделена в `models/`), но задача не решает текущей боли и ждёт.

6. Что дальше

NEXT · TOWARD A WEAK LOCAL CLAUDE CODE

Цель следующего этапа сформулирована прямо: превратить TrifonAI в «слабую версию Claude Code» — локального агента, который не только читает и пишет файлы, но и работает с проектом по-настоящему. Три направления:

- **Выполнение команд в терминале** — с тем же принципом обязательного подтверждения, что уже отработан на записи файлов.
- **Точечное редактирование файлов** вместо перезаписи целиком — правка фрагмента, а не файла.

- **Более длинный агентский цикл** — способность держать многошаговую задачу, а не один вызов инструмента за ход. Здесь ограничение модели (см. §5) станет главным препятствием, и, вероятно, именно оно определит, когда придёт время для сменного движка помощнее.

Что важно: платформа к этому готова архитектурно. Реестр инструментов расширяется одним файлом, подтверждение уже абстрагировано под два интерфейса, память и личность от модели не зависят. Именно это и было целью с самого начала — и, пожалуй, главный результат двух недель работы: не набор функций, а фундамент, на котором следующий шаг не требует переписывания.