Representation layer: почему вертикальная схема данных дороже самой модели

2026-04-27

Representation layer: почему вертикальная схема данных дороже самой модели

TL;DR. Месячный счёт за инференс у вертикальной AI-команды на порядок меньше, чем разовая инженерная работа по описанию схемы домена — и именно второе создаёт переключательные издержки. Representation layer — это формальная схема сущностей, статусов, инвариантов и правил конкретного бизнеса, на которой работает агент. Модель commodity, схема — нет. Заметка для фаундеров AI-продуктов, руководителей, выбирающих вертикального вендора, и инженеров, выбирающих, где работать.

Цена, которую обычно недооценивают

Возьмём бюджет средней vertical-AI-компании, которая обслуживает auto-dealers или SaaS-поддержку: счёт за инференс к Anthropic или OpenAI — порядка $300–500 в месяц на одного активного клиента. Стоимость инженерной работы по описанию того, что в этом конкретном бизнесе считается лидом, какие у него валидные статусы, что считается дублем, кто имеет право пометить сделку как закрытую, — десятки тысяч долларов разовой работы плюс непрерывная доработка. Соотношение 1:100 в пользу схемы, не модели. И именно эта пропорция объясняет, почему вертикальные AI-продукты, проигрывающие на бенчмарках, выигрывают на удержании.

В предыдущей заметке мы констатировали факт: за один квартал четыре крупнейших провайдера схлопнули agent harness в managed-продукт, и сослались на три слоя, которые остались defensible: representation layer, trajectory data, institutional SOP. Эта заметка — глубокое погружение в первый. Мы утверждаем: стоимость переключения вертикального AI-провайдера прямо пропорциональна глубине интеграции representation layer в операционные процессы клиента, а не качеству модели, на которой он работает.

Что именно мы называем representation layer

Representation layer — это формальная модель домена, в терминах которой работает агент: набор сущностей бизнеса, их атрибутов, валидных состояний, инвариантов и правил перехода между ними, выраженный как явный, версионируемый, машинно-проверяемый артефакт. Это не «структура базы данных» в традиционном смысле и не онтология ради онтологии. Это контракт, по которому модель и операционные процессы понимают одно и то же под одним и тем же словом.

Vertical schema — это representation layer, специализированный под конкретную отрасль: схема для auto-dealers фиксирует, что лид «горячий» при наличии тест-драйва, в SaaS-поддержке — при NPS-сигнале и определённой роли пользователя, в коммерческой недвижимости — при подписанной LOI. Универсальная схема CRM этих различий не делает; vertical schema их кодифицирует, потому что без них агент возвращает правдоподобную, но операционно неверную работу.

Здесь же мы можем зафиксировать ещё два понятия, которые в популярной прессе часто склеиваются. Data moat — это устойчивая разница между тем, что знает про домен ваш продукт, и тем, что знают конкуренты, при условии что разница не воспроизводится за разумные деньги и время извне. И switching cost — это совокупная цена для клиента покинуть текущего вендора: переписать интеграции, восстановить кодифицированные процессы, заново обучить людей и заново накопить операционную историю. У AI-продуктов основной вклад в switching cost даёт не контракт и не стоимость миграции данных, а именно потеря representation layer, которую невозможно «выгрузить в CSV».

Почему модель — заменимая часть стека

Бенчмарки последних восемнадцати месяцев показывают одну и ту же вещь: разрыв между frontier-моделями быстро схлопывается. Anthropic выпустил Managed Agents в апреле 2026 с persistent memory и self-evaluation; OpenAI в это же время отгрузил AgentKit с визуальным Agent Builder; AWS — Bedrock AgentCore с Runtime, Gateway, Memory как managed primitive. Любая команда, которая использует одну из этих платформ, может переключиться на другую за выходные, если речь идёт только об инференсе и harness.

Что не переключается за выходные — это смысловой контракт между текущей моделью и реальной операционной системой клиента. Когда агент в системе поддержки решает, что «тикет требует эскалации», он опирается на вертикальную схему: какие поля в тикете считаются обязательными, какие сочетания статусов означают SLA-риск, какой пользователь имеет право снять эскалацию. Эта схема накапливалась месяцами в коде, в промптах, в валидаторах, в правилах маппинга. Модель — её исполнитель, не носитель.

Jerry Chen из Greylock в эссе «The New Moats: Why Systems of Intelligence Are the Next Defensible Business Model» формулирует это как «системы интеллекта»: в эпоху, когда сами модели становятся горизонтальным API, защита смещается в системы, которые накапливают уникальные данные, feedback loops и контекст их применения. Через пять лет тезис не устарел — изменилась только точность. Уникальные данные сами по себе не moat: их надо описать на языке, который понимает бизнес, и в формате, который понимает агент. Этот язык и есть representation layer.

Из чего реально состоит схема

На уровне артефактов representation layer — это четыре слоя, которые в зрелом продукте живут в репозитории и обновляются как код.

Первый — типы и сущности. JSON Schema или Pydantic-классы для основных объектов домена с явными атрибутами и связями. Например, в SaaS-поддержке: `Account`, `User`, `Ticket`, `Conversation`, `Escalation`, с типизированными ссылками между ними. Этот слой обычно выглядит дёшево — но именно его форма определяет, какие вопросы агент в принципе сможет задавать в данных.

Второй — конечные автоматы и инварианты. Граф валидных переходов состояний (`new → triaged → in_progress → resolved → closed`), правила, при которых переход допустим, и условия, при которых сущность не имеет права существовать. Без этого слоя агент способен сгенерировать «правдоподобный» исход, который на проверке окажется операционно невозможным — и это самая частая причина отказа vertical-AI-продуктов на ранних этапах.

Третий — события и trajectory hooks. Каждое значимое изменение состояния порождает событие с фиксированными `actor`, `timestamp`, `before`, `after`, `reason`. Этот слой непосредственно соединяется с тем, что мы в прошлой заметке называли trajectory data: без формализованной схемы событий замкнутая петля «вход → решение → исход» не собирается. Open-source memory-стек, такой как Graphiti или Letta, даёт хранилище для таких событий и temporal reasoning поверх них, но не даёт самой схемы — её всё равно проектирует команда.

Четвёртый — резолверы и валидаторы. Код, который превращает грязный вход реального мира — почту, чаты, формы, выгрузки — в сущности схемы и обратно. Это самая «грязная» часть representation layer и одновременно самая защищённая: чтобы её воспроизвести, конкурент должен не просто прочитать вашу схему, а заново пройти через все edge cases вашего домена.

Vendep в эссе «Forget the data moat, the workflow is your fortress in vertical SaaS» формулирует это резко: «moat — это не данные сами по себе, а то, как они вшиты в рабочий процесс». На уровне representation layer этот тезис конкретизируется: «вшиты в рабочий процесс» означает, что схема валидируется на каждом шаге, и любое расхождение между моделью и реальностью становится явным, а не молчаливым.

Где это уже работает

Зрелые vertical-SaaS-компании де-факто давно построили representation layer, просто не называли его так. Procore описывает строительный объект через типизированный набор RFI, submittals, change orders с явными статусами и инвариантами; ServiceTitan для home services формализует звонок, work order, наряд бригады и инвойс через типизированные сущности с явной историей переходов. На этом фундаменте AI-функции прирастают как естественное расширение, а не как отдельный продукт. Когда Foundation Capital в эссе «AI Leads a Service-as-Software Paradigm Shift» пишет про переход от продажи

софта к продаже готовой работы, имплицитное условие этой модели — что «работа» формализована достаточно, чтобы её можно было поручить системе. Representation layer и есть инструмент такой формализации; без него service-as-software не выходит за пределы demo.

Обратный пример полезен не меньше. Forbes в марте 2026 описывал, как горизонтальные AI-«обёртки» над generic CRM теряют клиентов в пользу вертикальных продуктов с глубоким пониманием домена — несмотря на то, что качество моделей у обёрток часто выше. Причина прозаична: универсальная CRM-схема не различает реструктуризацию долга в B2B-кредитовании и продление подписки в SaaS, а вертикальный продукт различает, и его агент перестаёт делать абсурдные предложения. Это и есть видимая часть representation layer — она проявляется как «продукт просто понимает наш бизнес».

Что это значит на практике

Для фаундера AI-продукта. Тестовый вопрос для собственного продукта: если завтра OpenAI выпустит модель, которая на ваших задачах работает на 10% лучше, насколько изменится ваша экономика удержания? Если сильно — у вас, скорее всего, нет representation layer, у вас тонкая обёртка над API. Если почти не изменится — у вас есть слой, который компаундируется. Конкретный действие: перенесите одну центральную сущность вашего продукта (лид, тикет, ордер, контракт) из «свободно лежит в промптах и коде» в явную, версионированную схему с инвариантами. Через три месяца оцените, насколько чаще вы исправляете ошибки агента в коде валидаторов, а не в промптах. Этот сдвиг — самый ранний признак, что слой начал формироваться.

Для руководителя, выбирающего вендора. На discovery-встрече задайте один вопрос: «покажите, как у вас описана сущность X в нашем бизнесе». Если ответ — «мы передадим это в промпт модели», вы покупаете harness, и ваша зависимость от вендора будет нулевой, а ценность — кратковременной. Если ответ — «вот версионированная схема, вот инварианты, вот резолверы, вот политика изменений», вы покупаете operating layer, у которого есть и глубина, и аудит, и опционально перенос. Второй важный вопрос: «при расторжении контракта что я получу обратно — дамп таблиц или работающую копию representation layer на стандартных форматах». Ответ на второй вопрос отличает зрелого вендора от оппортуниста.

Для инженера, выбирающего, где работать. В команде, которая накапливает representation layer, ваша работа компаундируется: каждый новый клиент уточняет схему, каждый новый edge case добавляет валидатор, через два года вы один из немногих людей с реальным domain-driven understanding своей вертикали. В команде, которая занята прослойками поверх managed harness, через тот же срок вы — специалист по стеку, который провайдеры съели. На собеседовании смотрите, есть ли у команды отдельный артефакт «schema» или «ontology», кто им владеет, как часто он меняется. Если на этот вопрос команда не понимает, о чём вы спрашиваете, — это сигнал.

На что мы будем смотреть дальше

Если тезис верен, в течение ближайших 12–18 месяцев должны появиться три вещи. Первая — публичные стандарты для vertical schema по отдельным отраслям: не «общий JSON Schema», а конкретные онтологии для логистики, страхования, ритейла, согласованные между несколькими игроками. Сейчас этого нет, и каждый вендор изобретает свою. Вторая — рост сегмента ESB-подобных продуктов вокруг переноса representation layer между провайдерами, по аналогии с тем, как Open Banking стандартизировал API банков. Третья, самая важная — появление кейсов компаний, которые сменили модель и harness без потери operating-слоя: это будет публичным доказательством того, что слой действительно отделим и переносим. Если такие кейсы появятся, рынок vertical-AI окончательно перестанет конкурировать моделями.

Главное

- Representation layer — формальная схема сущностей, статусов и решений конкретного бизнеса — основной источник switching cost у вертикальных AI-продуктов; модель и harness заменимы, схема — нет.
- Слой состоит из четырёх артефактов: типы и сущности, конечные автоматы и инварианты, события и trajectory hooks, резолверы и валидаторы. Все четыре живут в репозитории и обновляются как код.
- Тестовый сигнал для продукта: если улучшение базовой модели не двигает удержание — у вас есть слой; если двигает сильно — вы продаёте обёртку.
- Для покупателя AI-продукта главный вопрос — не «какая там модель», а «как описаны сущности моего бизнеса и что я унесу при расторжении».
- В ближайшие полтора года индикатор зрелости рынка — появление публичных vertical-схем и переносимых operating-слоёв между вендорами.

Вопросы и ответы

Можно ли сделать representation layer переносимым между провайдерами? Часть — да, часть — нет. Схема, инварианты и валидаторы хранятся как код и не зависят от выбора модели. Переносимость ломается на двух стыках — на интеграциях с операционными системами клиента и на управляемой памяти провайдера, где формат событий и trajectory hooks может быть проприетарным. Стандарт переноса появится не раньше, чем будет рыночное давление со стороны крупных покупателей.

Чем representation layer отличается от обычной схемы базы данных? Схема БД описывает форму хранения; representation layer описывает смысловой контракт между бизнесом и агентом. У него обязательно есть инварианты, конечные автоматы и резолверы — слои, которые в традиционной БД-разработке либо живут в коде приложения, либо отсутствуют вовсе. Кроме того, representation layer проектируется так, чтобы его читала и писала и модель, и человек, и валидатор — и расхождения между ними становились видимыми.