12+

СЕРГЕЙ КИРНИЦКИЙ

Право на суждение

АГЕНТНОСТЬ КАК ПРИНЦИП ПРОЕКТИРОВАНИЯ ИИ-СИСТЕМ

Сергей Кирницкий

Право на суждение. Агентность как принцип проектирования ИИ-систем

«Издательские решения»

Кирницкий С.

Право на суждение. Агентность как принцип проектирования ИИ-систем / С. Кирницкий — «Издательские решения»,

В книге рассматривается проектирование систем на основе больших языковых моделей через призму распределения суждений между приложением и моделью. Вводится понятие изъятия агентности, выстраивается его таксономия, анализируются условия возврата суждений модели и сопутствующие риски: недетерминизм, безопасность, стоимость. Предлагается критерий проектирования и процедура аудита существующих систем. Издание рассчитано на инженеров и архитекторов ИИ-систем.

Содержание

Право на суждение
Агентность как принцип проектирования ИИ-систем

Сергей Кирницкий

Оформление обложки Created with Grok

ISBN 978-5-0070-7172-7
Создано в интеллектуальной издательской системе Ridero

Введение. Право на суждение

Где-то в вашей системе есть строка кода, которую давно никто не открывал. В ней записано число — скажем, четыре: сколько фрагментов доставать из базы знаний на каждый вопрос пользователя. Или правило: модели разрешён один проход, второго не будет. Или развилка: вопросы об оплате — в эту ветку, всё остальное — в ту. Или схема, в которую ответ обязан уложиться, каким бы ни оказался вопрос. Строка появилась при проектировании; писали её, возможно, не вы, а того, кто писал, может уже не быть в команде. С тех пор система пережила смену модели, пару миграций и один редизайн — а строка всё там же, и прямо сейчас она выносит решение за очередного пользователя: что для него релевантно, сколько попыток заслуживает его задача, каким его вопросу вообще позволено быть. Почему число именно такое, никто уже не помнит; спросить не у кого, да и незачем — система работает, жалоб немного, панели мониторинга зелёные, и именно поэтому строку никто не открывает. Но интересно не «почему четыре». Интересен вопрос, который я буду задавать каждому такому месту: чьё это суждение? Кто его на самом деле выносит — модель, которая читает живой вопрос в эту секунду, или код, решивший всё однажды и с тех пор ни разу не передумавший?

Долгое время честный ответ был: код — и он был правильным. Прежнюю модель приходилось вести за руку: она теряла нить на втором шаге, путала инструкцию с данными, выдавала правдоподобную форму вместо верного содержания. Отдавать ей решения о собственном пути было не смелостью, а безответственностью, и инженерная культура вокруг моделей закономерно сложилась как культура недоверия — разумного, заслуженного, многократно подтверждённого продакшеном. Интеллект системы жил в коде: в правилах и ветвлениях, в конвейерах обработки, в маршрутизации запросов. Вокруг того, как обойтись наименьшим участием модели, выросла целая дисциплина: пусть модель делает узкую, проверяемую работу, а думает — код. Модель занимала в этой конструкции одну клетку — классифицировала, размечала, отвечала на заранее подготовленный запрос — и возвращала результат туда, где все остальные решения давно были приняты за неё. Чем меньше от неё зависело, тем спокойнее спала команда.

Это устройство мира кончилось — и кончилось быстрее, чем успели измениться привычки. Интеллект системы сместился из статичной логики приложения в саму модель. Сегодняшняя модель — из лучших, доступных в своём поколении, — держит длинную инструкцию и не теряет её к десятому шагу, удерживает план и пересматривает его по ходу, вызывает инструменты и читает их результаты, замечает собственную ошибку и пробует иначе, работает с контекстом, который прежде не поместился бы целиком. Существенно одно: та работа суждения, которую раньше приходилось вынимать из модели и записывать в код, потому что модель её не тянула, теперь всё чаще оказывается тем, что модель делает лучше кода. И тогда прежняя оболочка из страховки превращается в помеху: она уже не компенсирует слабость — она не даёт проявиться силе. А та строка об этом не знает. Способность модели выросла на поколения; права, выданные ей архитектурой, остались прежними. Для инженера это редкий случай: в системе уже лежит незадействованный ресурс — способность, к которой никто не выписал прав. Обычно выигрыш приходится добывать по крохам, вытачивая проценты из конвейера; здесь он заперт одной строкой.

Этот разрыв — между тем, что модель может, и тем, что ей позволено, — и есть предмет книги, начиная с её названия. Каждый раз, когда внешняя логика решает за модель то, что та могла бы решить сама — что искать, что предпринять, повторить ли попытку, что удержать в памяти, действовать ли, — суждение у модели изымается: выносится один раз, на этапе проектирования, и застывает в коде. Я называю это изъятием агентности, а застывшее решение — замороженным суждением; строгие определения подождут — пока достаточно имён.

Замороженное суждение не ошибается. Оно просто не пересматривается — даже когда мир вокруг изменился, а модель, за которую его когда-то заморозили, теперь вынесла бы его лучше. Отсюда и название книги — право на суждение: способность сама по себе не выдаёт прав. Права в системе выдаёт архитектор, и модель будет выносить ровно те суждения, которые он ей оставил, — сколько бы поколений способности ни прошло мимо этой строки. Книга о том, где это право вернуть — потому что способность до него доросла, — и где, так же осознанно, не давать.

Такие замороженные суждения рассыпаны по любой зрелой системе, и почти все они невидимы, потому что выглядят как просто код — как норма, не требующая оправданий. Жёстко заданное число фрагментов, которые всегда достаются из базы, — это решение о том, что релевантно, вынесенное за модель и с тех пор не пересмотренное. Классификатор, раз и навсегда постановивший, какими бывают вопросы пользователей, — решение о природе задачи, застывшее в чьей-то старой таксономии. Единственный разрешённый проход там, где задача просит второго взгляда, — решение о том, сколько усилий заслуживает ответ. Ни одно из этих мест не выглядит как отнятое суждение; все они выглядят как настройки, значения по умолчанию, здравый смысл. В этом и трудность: изъятие не оставляет следов на поверхности — его не видно ни в демо, ни в интерфейсе, ни в метриках, пока запросы похожи на предусмотренные. Всё начинается с навыка замечать такие места: видеть в строке конфигурации не параметр, а решение — чьё-то, старое и до сих пор действующее.

Упрёка тем, кто эти строки писал, здесь нет. Тот, кто обернул слабую модель плотной оболочкой, решал реальную задачу своего времени и решал её правильно. Модель, терявшая нить на втором шаге, не могла вести многоходовый поиск — значит, поиск разумно было свести к одному ходу и задать снаружи; модель, отвечавшая уверенно даже на нерелевантной выборке, не годилась в судьи собственного контекста — значит, контекст решали за неё. Это была не леность мысли, а инженерная честность: точная подгонка архитектуры под то, что модель тогда умела. Само по себе изъятие — законный приём, и отменять его никто не предлагает; упрёк адресован не людям и не старым архитектурам, а инерции — решениям, которые были верны при одних способностях модели и молча пережили их рост. Порок начинается не там, где суждение забирают у модели, а там, где его забирают не выбором, а привычкой. И снять этот упрёк важно раньше любых переворотов: трудно пересматривать выбор, за который тебя не перестали винить.

Главный тезис заявлю сразу и без доказательств. Свобода модели — правильный дефолт проектирования; ограничение — осознанное исключение. Это не призыв дать свободу всему, а дисциплина различения: видеть, какое суждение отдать модели, а какое сознательно оставить приложению, — и настаивать лишь на порядке, в котором эти вопросы задаются, — сначала «что оставить модели», потом «что забрать», а не наоборот. Повод вернуть суждение всегда конкретен: модель способна вынести его лучше замороженного кода — не по определению, а на живых запросах, где код промахивается. Конкретна и цена: вместе с суждением модель получает свободу ошибиться иначе и право пойти путём, которого инженер не закладывал. Разрешите ей повторить попытку, когда первый ответ не сошёлся, — и она вытащит запрос, на котором одноходовый конвейер сдавался; но заплатить придётся лишним раундом и маршрутом, который не расписать заранее. Настоящая ставка возврата — не проценты к метрике, а класс запросов, прежде недосягаемый. А есть решения, которые модели не стоит отдавать вовсе, — недоверенный ввод, недопустимая цена ошибки, требование строгой воспроизводимости. Разморозить суждение — не жест доверия и не идеология, а инженерное решение с поводом и ценой; обе величины останутся на виду до самого конца — ни выгода не спрячется за осторожностью, ни цена за энтузиазмом. Тезис, умалчивающий о своей цене, — лозунг, а не метод.

Я пишу для тех, кто эти системы строит: для инженеров и архитекторов ИИ-систем, техлидов, продакт-инженеров — для всех, кто отвечает за то, где в системе проходит граница между кодом и моделью. От читателя потребуется знакомство с большими языковыми моделями на уровне пользователя API — запрос и ответ, вызов инструмента, контекстное окно; глубокого машинного обучения не нужно, потому что речь не о том, как модели устроены внутри и как их обучают. Предмет — то, что мы строим вокруг готовой модели, и решения, которые в этой обвязке кто-то за кого-то выносит: достаёт ли модель нужные документы сама или ей их подкладывают; разрешён ли второй заход, если первый не дал ответа; чья версия важного остаётся в памяти — модели или того, кто сжал контекст за неё; где систему остановить, а где отпустить. Это не введение в языковые модели с нуля, не туториал под конкретный инструмент и не книга об оркестрации множества агентов между собой — это книга про одну развилку, которая проходит через каждую ИИ-систему, и про то, как проходить её не вслепую. Скорее всего, вы уже выпустили в мир систему, где каждое из этих решений принято, — вопрос лишь в том, кем и глядя ли.

Сквозь книгу проходит одна карта — карта суждений: несколько типовых решений, которые ИИ-система либо выносит за модель, либо доверяет ей, — что и где искать, что делать дальше, повторить ли попытку, как ответить, что помнить, действовать ли самостоятельно. Каждое пройдёт три станции: где его изымают, где его можно вернуть и чего этот возврат стоит. По той же карте движется сквозной пример — ассистент поддержки на базе документации: система, которую хоть раз строил или видел почти каждый и в которой естественно живут все решения карты. Сначала это знакомая версия с изъятой агентностью: подготовленный за модель контекст, классификатор намерений, один проход, шаблонный ответ. Потом она упрётся в потолок на запросе, которого никто не предусмотрел. Потом её перепроектируют, суждение за суждением возвращая модели то, что было заморожено. Потом посчитают цену этой свободы и проверят систему на прочность там, где свобода опасна. И под конец по ней пройдут с чек-листом, как по чужой системе, которую нужно честно оценить. Один и тот же бот через всю книгу — чтобы было видно, как одно и то же решение сначала замораживают, а потом размораживают, и что меняется на каждом шаге.

Осталось одно обещание. Здесь не будет рецептов под конкретный инструмент, протокол или модель; не будет версий, бенчмарков и имён — ничего, что устареет к следующему поколению. Это не осторожность, а позиция: метод переживает модели. Отсюда же — сдержанность со словарём, которым поле описывает само себя. Имена его дисциплин приходят слоями и сменяют друг друга: то, что вчера звалось prompt-инжинирингом, сегодня зовётся context-инжинирингом, а завтра получит новое имя для тех же, по сути, решений. Разговор сознательно пойдёт уровнем выше — на суждениях, а не на терминах сезона; у каждого текущего имени есть мост к тому, как ту же вещь называю я, и эти мосты собраны в глоссарии, чтобы читатель, пришедший с любым словарём, легко нашёл соответствие. Знать имя сезона полезно; строить на нём метод — значит привязать метод к сроку годности имени. Вопрос же «чьё это суждение?» не требует новой модели, чтобы его задать, и не устареет вместе со старой: он останется, когда сегодняшние библиотеки, протоколы и имена дисциплин сменятся неузнаваемо. Сегодня этот вопрос звучит так: пора перестать выбирать путь за модель и позволить ей самой находить лучший путь к задаче — там, где она на это способна, и ровно настолько, насколько способна. Где проходит это «настолько», как найти его для своей системы и чем за него платить — вся остальная книга. Начнём с того, чтобы научиться видеть суждения там, где мы привыкли видеть код, — с той самой строки, которую давно никто не открывал.

Часть I. Изъятие агентности

Две системы могут быть неотличимы снаружи. Один и тот же интерфейс, та же база знаний, та же модель под капотом. Две команды показывают демо своего ассистента поддержки; обе уверенно отвечают на два десятка типовых вопросов; обе выглядят готовыми к продакшену. Разница между ними не проявляется в демо. Она проявляется тремя неделями позже — на двадцать первом вопросе, том самом, который никто не прописал заранее.

Один ассистент на этом вопросе спотыкается ровно и предсказуемо: находит не тот раздел базы, отвечает уверенно и мимо, и не может сказать «мне принесли не то». Другой — переспрашивает базу иначе, замечает, что первый результат не отвечает на вопрос, и находит путь там, где его не прокладывали. Компоненты у них одинаковые. Различаются они тем, кто внутри выносит решения.

Именно это различие — предмет книги, и прежде всего его нужно научиться видеть. Возьмём типичную поддержку на базе знаний, docs-бот, знакомый почти каждому, кто строил что-то на языковых моделях. В самой распространённой его версии почти каждое содержательное решение принято не моделью. Что искать по запросу пользователя — решает механизм извлечения, отдающий модели готовую выборку. Какого рода это запрос и в какую ветку его направить — решает роутер намерения на входе. Стоит ли попробовать ещё раз, если найденное не отвечает на вопрос, — не решает никто: конвейер одноходовый, второй попытки в нём просто нет. Как оформить ответ — задаёт шаблон. Модель в этой системе делает ровно одно: формулирует текст поверх того, что ей уже выбрали, направили и разрешили. Она — последнее звено, а не то, где живёт интеллект.

Каждое из этих решений когда-то было принято — один раз, на этапе проектирования, — и с тех пор не пересматривалось. Не потому, что оно всегда верное, а потому, что его вынесли заранее и заморозили в коде. Замороженное суждение* — это решение, которое кто-то вынес однажды при проектировании системы и застыло в её логике: оно больше не зависит ни от запроса, ни от того, что вернул поиск, ни от того, справляется модель или нет. Замороженное суждение не ошибается в том смысле, в каком ошибается живое. Оно просто не пересматривается.*

Когда суждение, которое модель могла бы вынести сама, выносит за неё внешняя логика и замораживает, — агентность изъята. Это и есть паттерн: изъятие агентности. *Инъекционный RAG, роутер на входе, одноходовый конвейер, шаблонный ответ — не четыре независимых технических решения, а четыре точки одного паттерна. И этот паттерн — не экзотика и не признак плохой инженерии. Он повсюду: в поиске, в маршрутизации, в памяти, в том, разрешено ли системе действовать. Спешить осуждать его не стоит, и лекарство подождёт — тем более что не всякое изъятие в нём нуждается. Нужна оптика, в которой знакомый продакшен вдруг читается как список чужих замороженных решений. Дискомфорт этого узнавания — рабочий: пока не увидишь паттерн целиком, обсуждать, что с ним делать, преждевременно.*

Глава 1. Что такое агентность в ИИ-системе

Мы легко говорим, что один ассистент «умнее» другого. Но если выложить оба на стол и разобрать по частям, окажется, что части одинаковы: тот же класс модели, та же база, тот же способ доступа к ней. Разница, которую все чувствуют и которая через месяц решит судьбу обоих продуктов, в этой описи компонентов не значится. У неё пока нет имени.

Слово «агентность» напрашивается, но оно перегружено. Для одних это автономные системы, действующие без человека; для других — почти сознание; для третьих — модный ярлык на всём, что вызывает инструменты. Ни одно из этих значений не годится как рабочее: с ними нельзя спроектировать систему, потому что они описывают ауру, а не устройство. Прежде чем строить на понятии агентности, его нужно очистить до чего-то, что можно положить в основу решения.

Расчистка начинается с одной оси и одного определения. Ось: интеллект решения в системе живёт либо в статичной логике приложения, либо в модели — и почти никогда посередине для каждого отдельного решения. Определение: агентность — это мера, в которой суждения выносит модель, а не то, что застыло вокруг неё. Но и то и другое становится инструментом лишь после того, как различие увидено на конкретной системе: определение, пришедшее раньше разницы, запоминается как формулировка, а не как инструмент.

1.1. Интеллект системы: в приложении или в модели

Вернёмся к двум ассистентам поддержки и дадим им конкретную задачу. Пользователь пишет: «Обновил тариф вчера вечером, а лимиты на API остались старые — это нормально или что-то сломалось?». В базе знаний нет статьи с таким заголовком. Есть статья про то, как меняются тарифы, есть отдельная — про то, что изменения лимитов применяются в начале следующего расчётного периода, и есть третья — про типичные задержки синхронизации. Ответ живёт на стыке трёх статей, и ни одна из них по отдельности его не содержит.

Первый ассистент устроен так: запрос пользователя целиком уходит в поиск, поиск возвращает три самых похожих фрагмента, эти фрагменты подставляются в подсказку модели, модель пишет ответ. Запрос был про «лимиты остались старые», и поиск честно принёс самое похожее — статью про изменение лимитов. Модель получает этот фрагмент как данность и на его основании уверенно сообщает: лимиты меняются в начале следующего периода, всё в порядке, ждите. Ответ звучит гладко и почти наверняка неполон: он не различает штатную задержку и настоящий сбой, потому что в принесённой выборке этого различения нет, а спросить, поискать иначе или усомниться в выборке модель не может. Ей нечем: решение о том, что искать и что считать релевантным, уже принято — механизмом извлечения, до неё.

Стоит задержаться на том, чего именно лишена первая система. Не знаний — статья про сбои лежит в той же базе, до неё просто не дотянулись. Не сообразительности — модель одна и та же в обеих системах. Она лишена способности заметить, что отвечает не на тот вопрос. Ей отдали выборку, которую она не запрашивала, и она отвечает на вопрос, который эта выборка задаёт, а не тот, что задал пользователь. Проблема не в том, что система знает мало, а в том, что она не видит границ своего знания в этом конкретном случае, — и уверенность ответа от этого не страдает. Гладкость и правота здесь развязаны.

Второй ассистент устроен иначе. Поиск дан ему не как подкладка под ответ, а как инструмент, которым распоряжается модель. Она видит вопрос, замечает в нём две разные возможности — «штатная задержка» и «сбой», — и это её суждение, а не ветка, выбранная роутером. Она ищет по первой, получает статью про начало расчётного периода, ищет по второй, получает статью про задержки синхронизации, сопоставляет их с деталью «вчера вечером» и

отвечает так, как ответил бы человек, знающий базу: скорее всего, это штатная задержка до начала следующего периода, и вот как проверить, что это не сбой. Тот же интерфейс, та же база, тот же класс модели. Разными их сделало одно: во втором случае суждение о том, что здесь вообще происходит и где это искать, вынесла модель.

Важно сразу уточнить масштаб этой оси. Она приложена не к системе целиком, а к каждому её решению по отдельности. Неверно спрашивать «агентна ли эта система» так, как спрашивают, на каком она написана языке: система — не точка на оси, а связка решений, и каждое лежит на своей точке. В одном и том же ассистенте решение «как переформулировать запрос к поиску» может быть отдано модели, а решение «отправлять ли исходящее письмо клиенту без подтверждения человека» — намеренно заморожено в приложении. Это не противоречие и не половинчатость. Это норма: у зрелой системы карта решений пёстрая, и её пестрота — не признак недоделанности, а результат отдельных выборов, каждый со своей причиной. Тумблера «свобода/контроль», делящего систему надвое одним щелчком, не существует; есть десятки мелких решений, и каждое лежит там, куда его положили.

Ось эта не про поддержку и не про поиск — она проступает везде, где приложение обёрнуто вокруг модели. Возьмём систему, извлекающую из входящих документов структурированные данные: контрагент, сумма, срок оплаты. В одном варианте разработчик заранее задаёт жёсткую схему полей, а модель лишь заполняет клетки — что считать «суммой» и «сроком», решено до неё, схемой. В другом модель сама разбирается, что в этом документе есть значимого, и предлагает структуру под конкретный документ. Первый вариант надёжно провалится на документе непредусмотренного вида — там, где значимое поле в схему не заложено, его просто некуда записать; второй с таким документом справится, но заплатит меньшей предсказуемостью формы ответа. Те же две стороны оси, тот же размен — на совсем другом материале. Ось универсальна именно потому, что описывает не предметную область, а место, где принято решение.

Разложим, что именно различает эти системы. Не качество модели — она одна и та же. Не полнота базы — статьи те же самые. Не интерфейс, не язык, не оформление. Различает их место, где вынесено решение. В первой системе решение «что здесь релевантно» вынесено заранее и снаружи — логикой приложения, которая всегда делает одно и то же: берёт запрос, отдаёт в поиск, подставляет верхние результаты. Во второй то же решение вынесено моделью в момент задачи, с учётом её конкретики. Это и есть ось, на которую ложится любая ИИ-система: *интеллект системы* — это то, где принимаются её содержательные решения: в статичной логике приложения или в модели.

Слово «статичная» здесь несущее. Логика приложения не глупа — она бывает весьма изощрённой. Но она вынесена один раз и не зависит от конкретного запроса: она сделает с двадцать первым вопросом ровно то же, что с первым, даже если двадцать первый требует другого. Модель, напротив, выносит решение всякий раз заново, глядя на то, что перед ней. Оба режима легитимны. Есть решения, которые правильно застывают в приложении: их незачем принимать заново каждый раз, и цена ошибки живой модели тут выше выгоды. Решение о том, что данные одного пользователя нельзя показать другому, правильно заморожено в коде — доверять его живому суждению модели на каждом запросе — не гибкость, а риск, и его застылость здесь — не долг, а защита. И есть решения, застывание которых обходится дорого, — как «что здесь релевантно» у первого ассистента. Вопрос не в том, какой режим лучше вообще. Вопрос в том, какое конкретное решение в каком режиме должно жить.

Есть простой способ определить, по какую сторону оси лежит конкретное решение: посмотреть, меняется ли оно, когда меняется вход. Замороженное решение к входу глухо — оно применяет одну и ту же процедуру к типовому входу и к небывалому. Решение, оставленное модели, вход слышит: на непохожем запросе оно может выйти иначе, потому что выносится под него. Отсюда и слово «интеллект» в названии оси — не как похвала и не как метафора,

а почти буквально: интеллект решения тем выше, чем сильнее оно приспосабливается к конкретике задачи, а не воспроизводит заранее заданную форму. Приложение может быть сколь угодно изощрённым как программа и при этом нести нулевой интеллект решения — если это решение одно на все входы. Сложность кода и интеллект решения — разные величины, и их легко перепутать: система с тысячей правил маршрутизации выглядит умной, но каждое из правил заморожено, и на входе за пределами тысячи она беспомощна.

И вот здесь ось из наблюдения превращается в ось проектирования. Где живёт интеллект решения — не свойство, доставшееся системе от природы, и не следствие выбранной модели. Это сумма выборов её авторов, сделанных решение за решением, часто по инерции, часто не глядя. Когда команда первого ассистента писала «запрос → поиск → верхние результаты → модель», она не формулировала это как «отберём у модели суждение о релевантности». Она писала очевидный, стандартный, рекомендованный отовсюду пайплайн. Изъятие произошло не как злой умысел и даже не как решение — как форма по умолчанию. Стартовый шаблон, первый попавшийся пример из документации, готовый рецепт «как сделать RAG» — все они уже расставили решения по оси заранее, и расставили в одну сторону: суждение — приложению, текст — модели. Разработчик получает эту раскладку в наследство вместе с первой строкой кода и чаще всего не замечает, что она вообще была раскладкой, а не единственным способом. Но от того, что оно было незаметным, изъятие не перестало быть выбором. Кто-то расставил на этой оси каждое решение системы, и большинство систем стоят там, где стоят, не потому, что там их место, а потому, что туда их поставила привычка.

Цена этого выбора не видна на предусмотренном. На первых двадцати вопросах — тех, под которые пайплайн и настраивали, — замороженное решение о релевантности работает не хуже живого: выборка та, что нужно, ответ верный, демо гладкое. Разрыв открывается на двадцать первом — на запросе, которого проектировщик не предвидел, потому что предвидеть все запросы нельзя. Здесь замороженное решение делает то же самое, что делало всегда, — и именно постоянство оказывается ошибкой. Модель, которой оставили это суждение, на непредусмотренном запросе хотя бы пробует другой ход; модель, у которой его отобрали, уверенно отвечает на вопрос, который ей подменили выборкой. Место, где вынесено решение, ничего не стоит до тех пор, пока система остаётся внутри предусмотренного, — и начинает стоить сразу за его границей. А граница предусмотренного проходит не там, где её ждут: реальный поток запросов почти всегда шире тестового набора, на котором систему признали готовой.

Различие между двумя ассистентами теперь можно показать пальцем: оно в том, по какую сторону оси лежат их ключевые решения. Но «показать пальцем» — ещё не «назвать». У меры, в которой решения отданы модели, должно быть имя, а у самого понятия — определение, достаточно точное, чтобы на него можно было опереться.

1.2. Агентность как контроль над суждением

Различие между двумя ассистентами держится на одном: кто выносит содержательные решения внутри задачи. В первой системе их выносит приложение и подаёт модели готовыми; во второй значительную часть выносит сама модель. Эту меру и называют агентностью. *Агентность* — мера, в которой модель сама выносит суждения внутри решения задачи, а не получает их готовыми извне.

Определение стоит читать медленно, потому что каждое слово в нём отсекает частое недоразумение. «Мера» — потому что агентность не переключатель «есть/нет», а величина: у одной и той же системы одни суждения отданы модели, другие заморожены, и общий уровень складывается из этой раскладки. «Сама выносит» — потому что речь именно о вынесении решения, а не о его исполнении: модель, дописывающая текст поверх чужого выбора, ничего не выносит, хотя работает. «Внутри решения задачи» — потому что нас интересуют суждения,

из которых складывается путь к ответу, а не поведение системы во внешнем мире. И «суждения» во множественном числе — потому что их в любой задаче много, и каждое можно рассматривать отдельно.

При этом суждения не равны по весу, и мера агентности складывается не из их числа, а из их значимости. В любой задаче есть решения, от которых зависит исход, и решения проходные. Отдать модели выбор формулировки и заморозить за приложением выбор того, что вообще искать, — это низкая агентность, сколько бы мелких решений модель ни принимала попутно: несущее суждение вынесено не ею. Обратное тоже верно: система может оставлять модели немного решений, но именно те, что определяют путь, — и быть при этом высокоагентной. Поэтому «сколько суждений отдано модели» — вопрос не арифметический. Считать нужно не штуки, а вес: какие из решений, определяющих судьбу ответа, вынесены моделью, а какие застыли до неё. Одно существенное суждение, оставленное модели, меняет систему сильнее десятка косметических.

Какого рода эти суждения? Что здесь вообще спрашивают и в какую сторону это решать. Где искать ответ и по каким словам. Достаточно ли найденного или стоит зайти иначе. Нужно ли переспросить, уточнить, разбить задачу на части. Каким должен быть ответ, чтобы он отвечал именно на этот вопрос. Ни одно из этих решений не про действие во внешнем мире — все они про то, как система движется к ответу. И каждое из них может быть либо оставлено модели, либо вынесено за неё заранее. Агентность — это про то, сколько таких суждений и насколько существенных остаётся за моделью.

Здесь неизбежен вопрос, на котором книга могла бы поскользнуться: в каком смысле у модели вообще «есть суждение»? Модель не размышляет за столом и ничего не взвешивает в человеческом смысле; говорить, что она «решает» или «понимает», — соскользнуть в мистификацию, от которой эта книга держится подальше. Но и другая крайность — отказать модели в суждении вовсе — сделала бы разговор невозможным: тогда пришлось бы описывать вторую систему языком «активаций» и «распределений», в котором различие между двумя ассистентами просто не выразить. Выход — рабочая рамка. *Суждение модели* здесь — рабочая рамка, а не утверждение о внутреннем мире: способность на конкретном входе выдать один исход, а не другой, обоснованно с точки зрения задачи. Когда вторая система «замечает», что вопрос допускает две трактовки, и ищет по обеим, — это наблюдаемое поведение, которое проще и точнее всего описать как вынесенное суждение. Мы не заглядываем модели в голову; мы называем то, что видим на выходе, тем словом, которое позволяет об этом рассуждать и проектировать.

Проверить эту рамку можно тем же грубым способом, каким различали стороны оси: подменить вход — только смотреть теперь не на исход, а на путь к нему. Дайте первой системе запрос про тарифы, потом запрос про сбой — путь один и тот же: поиск по похожести, верхние результаты, ответ поверх них; менялся не путь, а лишь текст на его конце. Дайте те же два запроса второй — и путь разойдётся: разные переформулировки, разное число обращений к базе, разный порядок. Там, где вход меняет не только ответ, но и ход к нему, суждение вынесено на месте; там, где ход неизменен, — оно вынесено заранее и заморожено. Рамка не требует заглядывать внутрь модели: разницу видно снаружи, по следу, который система оставляет, решая задачу.

Эта рамка — не уступка удобству, а необходимое условие всего разговора об агентности. Без неё каждое утверждение вида «здесь суждение вынесла модель» пришлось бы либо переводить в мистику, либо разворачивать в абзац оговорок; с ней можно говорить прямо — и не приписывать модели ничего сверх наблюдаемого.

У контроля над суждением есть точный образ. Представьте дорогу и машину на ней. Приложение — тот, кто строил дорогу: заранее, для всех будущих поездок сразу, проложил ровно те повороты, которые предусмотрел. Модель с агентностью — тот, кто едет: выбирает

поворот здесь и сейчас, глядя на то, что перед ним. Пока маршрут совпадает с предусмотренным, разницы не видно — оба едут по одной трассе. Разница проступает там, где нужного поворота дорога не содержит: строитель уже ушёл и переложить полотно не может, а водитель ещё здесь и может свернуть. Контроль над суждением — это руль в руках того, кто едет, а не того, кто когда-то проектировал дорогу. Изъять агентность — значит забрать руль и оставить водителю только газ: скорость останется его, направление — уже нет.

Из этого образа сразу видно, чем агентность не является, — и два смешения стоит снять прямо в определении, пока они не приросли к термину. Первое: агентность — не автономия. Автономия — про надзор: действует ли система сама или под рукой человека, нужно ли подтверждение на каждый шаг. Это отдельная ось, и она пересекается с нашей как угодно. Можно построить систему с высокой агентностью и полным надзором: модель выносит все суждения о том, как решать задачу, но каждый её вывод проходит через человека перед применением. Это и есть human-in-the-loop — режим, в котором человек утверждает выводы системы перед их применением. Тот же docs-бот легко представить в этом режиме: модель сама разбирает обращение, ищет по базе, составляет полное решение — но не отправляет его клиенту, а кладёт на стол оператору как черновик. Все содержательные суждения здесь за моделью, ни одного действия наружу без человека. Агентность высокая, автономии — ноль. И можно построить систему автономную и почти безагентную: она крутится без человека сутками, но всё, что она делает, разложено по замороженным правилам, а модель лишь заполняет пустые места. «Без надзора» и «сама выносит суждения» — про разное, и путать их — значит спорить об одной оси, думая, что говоришь о другой.

Второе смешение: агентность — не действия. Соблазн измерять агентность тем, сколько система делает во внешнем мире — сколько вызывает инструментов, отправляет писем, меняет записей, — понятен, но обманчив. Действия — это то, что система совершает наружу; агентность — то, кто внутри решил их совершить и почему. Система, которая рассылает сотни писем по жёсткому сценарию, действует много и не решает почти ничего: все суждения о том, кому и что писать, вынесены до неё. И наоборот: система, которая не касается внешнего мира вовсе — только читает, ищет и рассуждает, — может нести очень высокую агентность, если каждое суждение о том, куда двигаться в задаче, выносит сама. Тот же обман прячется в счёте вызовов инструментов. Система, дёргающая десяток инструментов по заранее прописанной цепочке, выглядит деятельной, но каждый вызов в ней предрешён — модель лишь идёт по проложенному списку. Система, вызывающая один инструмент, но выбравшая его сама, под конкретную задачу, несёт больше агентности, чем первая при всей её суете. Мерить агентность объёмом действий — всё равно что мерить мышление количеством произнесённых слов.

Определение получилось короткое и, кажется, чистое: мера, в которой суждения внутри задачи остаются за моделью. Но слово «агентность» пришло в инженерный обиход не пустым — оно тащит за собой шлейф чужих значений, от философских до маркетинговых, и пока этот шлейф не отцеплен, определением нельзя пользоваться как инструментом: оно будет всякий раз стягиваться к тому, что читатель уже привык вкладывать в слово. Прежде чем строить, территорию вокруг термина нужно расчистить — по одному ложному толкованию за раз.

1.3. Чем агентность НЕ является

Слово пришло не пустым. «Агент», «агентность», «агентный» звучат в поле давно и успели обрасти значениями, которые к рабочему определению отношения не имеют, но липнут к нему при первом же разговоре. Пока эти значения не отцеплены, обсуждение любого проектного решения будет сворачивать в спор о терминах. Поэтому — расчистка: три ложных толкования и разметка границ, за которыми термин перестаёт значить то, что нужно книге.

У этих трёх — общее устройство. Каждое входит в разговор через свою дверь: одно из философии, где «агент» тянет за собой сознание; другое из идеологии, где агентность звучит как свобода-абсолют; третье из страха, где она слышится как отмена правил. Двери разные, а итог один — проектный вопрос подменяется чем-то посторонним, и обсуждение, кому отдать конкретное суждение, так и не начинается. Закрыть эти двери стоит до того, как термин пойдёт в работу.

Первое: агентность — не сознание. Рабочая рамка называет суждением наблюдаемую способность модели выдать на конкретном входе один обоснованный исход, а не другой; она ничего не говорит о внутреннем мире и намеренно в него не лезет. Смешение с сознанием опасно с обеих сторон. Кто принимает его всерьёз, начинает ждать от модели человеческого — устойчивых намерений, понимания последствий, ответственности — и либо переоценивает систему, доверяя ей то, чего она не тянет, либо, разочаровавшись, отказывает ей в суждении вовсе и возвращается к языку, на котором различие двух ассистентов не выразить. И то и другое уводит от инженерной задачи в спор, у которого здесь нет ни места, ни нужды: чтобы решить, кому отдать суждение о релевантности, не требуется знать, есть ли у модели внутренний мир. Требуется знать, выносит ли она это суждение лучше замороженного правила. Вопрос проектный, а не метафизический, и книга держит его таким.

Как это смешение выглядит вживую, видно по мелочам языка и решений. Команда начинает говорить «модель поняла, что клиент раздражён», «она решила, что так будет лучше», — и незаметно переносит на систему доверие, которое человек оказывает человеку. За словами приходят решения: раз «понимает» — можно убрать проверку, раз «хочет как лучше» — можно не перепроверять её выбор. Замените в тех же фразах «поняла» на «выдала исход, согласующийся с раздражением в тексте», и соблазн исчезает: становится видно, что доверять тут нужно не намерению, а надёжности исхода на этом классе входов — а это уже вопрос, на который есть инженерный ответ. Рамка суждения без сознания не обедняет разговор; она возвращает его на землю.

Стоит оговорить, чего это отсечение не делает. Оно не выносит приговора вопросу, способна ли модель к чему-то похожему на понимание в принципе, — этот вопрос книга оставляет тем, чья он тема, и не берётся ни утверждать, ни отрицать. Мы обходим этот спор не потому, что знаем ответ, а потому, что он не входит в условие задачи.

Второе: агентность — не автономия ради автономии. Тезис «отдавать суждения модели по умолчанию» легко прочитывается как призыв «отпустить всё» — снять ограничения, довериться модели во всём, отдать ей руль вместе с тормозами. Это не тезис книги. Дефолт — не абсолют: сказать «по умолчанию суждение остаётся за моделью» не значит «всегда и любое». Вся дисциплина подхода — про то, где этот дефолт сознательно нарушают: какое суждение осмысленно оставить приложению и почему. Манифест максимальной свободы обошёлся бы без второй половины; книга без неё не имеет смысла.

Полезно уточнить, что вообще значит здесь «по умолчанию». Дефолт — это не разрешение и не индульгенция, а место, откуда стартует рассуждение. Проектировщик, держащий агентность дефолтом, не освобождён от обоснований — он обязан обосновать каждое изъятие, тогда как при обратном дефолте молча обосновывать пришлось бы каждый возврат. Меняется не строгость, а сторона, на которой лежит бремя доказательства. Сказать «оставляем суждение модели, если нет причины забрать» — не то же самое, что «оставляем всё и не спрашиваем»: причины бывают, и веские. Дефолт задаёт, с чего начинать думать, а не позволяет думать не начинать.

Цена этого смешения двойная. Тот, кому оно нравится, слышит разрешение убрать всё и строит систему, которая на непредусмотренном срывается в непредсказуемое. Тот, кого оно пугает, слышит призыв к анархии и запирает всё обратно, теряя ровно ту способность находить

путь, ради которой агентность и возвращают. Оба спорят с манифестом, которого книга не писала.

Спор этот легко услышать на любом проектном обсуждении, где «агентность» произнесли без определения. Один инженер понимает её как «уберём лишние рамки, пусть модель решает» и уже прикидывает, что вырезать. Другой слышит в этом безрассудство и упирается: «в проде так нельзя, нужен контроль». Они спорят громко и мимо, потому что оба приняли толкование «агентность = свобода без границ» — только один за него, другой против. А вопрос, ради которого стоило собраться, — какое именно суждение эта система выносит лучше замороженного правила, а какое разумнее оставить снаружи, — не прозвучал ни разу. Определение с отцепленным «ради автономии» этот спор снимает: обсуждать становится нечего в идеологии и есть что в инженерии.

Третье: агентность — не «работа без правил». Здесь смешение самое цепкое, потому что кажется очевидным: раз суждение отдано модели, значит, рамки сняты. Но отдать суждение и снять рамку — разные операции. Правила, границы, ограничения никуда не исчезают; меняется их место — они встают вокруг суждения, а не вместо него. Разница практическая, не риторическая. Второй ассистент, тот, что сам решает, где искать, всё равно живёт в рамках: он не дотянется до данных другого клиента, не отправит наружу того, что требует подтверждения, не выйдет за пределы того, к чему ему открыт доступ. Эти границы очерчивают поле, внутри которого суждение остаётся за моделью, — но самого суждения не отменяют. Изъять суждение — значит решить за модель, что здесь релевантно; поставить границу — значит оставить ей решать, но очертить, где она может искать и что вправе сделать с найденным. Первое замораживает выбор, второе — оставляет его живым внутри очерченного поля.

Проще всего увидеть разницу, если заметить, что правила и агентность вообще лежат на разных осях. У первого ассистента, того самого замороженного, ровно те же внешние ограничения, что у второго: он тоже не дотянется до чужих данных, тоже не отправит наружу без подтверждения, тоже заперт в своей области доступа. Ограничений у него не меньше — а агентности нет вовсе. Значит, «есть правила» и «есть суждение» не связаны: можно иметь все мыслимые границы и нулевую агентность, а можно — высокую агентность внутри тех же границ. Толкование «агентность = отсутствие правил» сжимает эти две оси в одну и потому неверно в самой посылке. Как ставить такие границы, не изымая суждения, — отдельный разбор цены и защиты; здесь достаточно снять само уравнение «агентность = отсутствие правил».

Территория расчищена. Термин, определённый и очищенный, больше не тянет за собой ни сознания, ни анархии, ни отмены рамок: осталась мера, в которой суждения внутри задачи вынесены моделью. Но определение — ещё не метод. Знать, что такое агентность, и уметь увидеть, где она изъята в конкретной системе, — разные умения. Чтобы из чистого понятия получился рабочий инструмент, его нужно превратить в процедуру: не «что такое агентность вообще», а «чьё вот это решение — здесь, в этой строке архитектуры».

1.4. Единица анализа: одно решение — чьё суждение?

Понятие есть, оно очищено — но пользоваться им как есть неудобно. «Уровень агентности системы» — величина слишком крупная, чтобы за неё ухватиться: она усреднённая, размазанная по всей архитектуре, и на вопрос «а что здесь не так» отвечает в лучшем случае «в целом маловато». Инженеру нужно не «в целом», а «вот здесь». Чтобы понятие заработало, масштаб взгляда придётся сменить: перестать смотреть на систему как на набор компонентов и начать видеть её как набор решений.

Это смена оптики, и она непривычна. Архитектуру принято разглядывать по частям: вот механизм извлечения, вот роутер, вот модель, вот шаблон ответа, вот хранилище памяти. Части удобны — их видно на схеме, у них есть имена и границы. Но части ничего не говорят о

том, где живёт интеллект: механизм извлечения бывает и подкладкой под ответ, и инструментом в руках модели, и по коробке этого не отличить. Различие прячется не в компонентах, а в решениях, которые сквозь эти компоненты проходят. Поэтому единица анализа здесь — не компонент, а решение. *Суждение* здесь — единица анализа: одно решение внутри задачи, у которого есть исход и есть тот, кто этот исход выносит. Не «поиск» как модуль, а конкретное решение «что искать по этому запросу»; не «память» как хранилище, а решение «что из прошлого сюда относится».

Почему компонент для этого не годится, видно на любом из них. Возьмите тот же механизм извлечения. В первой системе он владеет решением о релевантности — берёт запрос и сам определяет, что подложить модели. Во второй тот же по названию механизм не владеет ничем: он лишь исполняет запросы, которые формулирует модель, а решение, что искать, осталось за ней. Компонент один и тот же, чертёж один и тот же, а владелец решения — противоположный. То же с памятью: «модуль памяти» может решать за модель, что из прошлого релевантно, а может просто хранить и отдавать по запросу модели — снаружи это один прямоугольник на схеме. Смотреть на компоненты — значит смотреть на коробки, внутри которых решение может лежать любой стороной. Смотреть на решения — значит сразу видеть ту сторону, которая всё определяет.

Как только систему разложили на отдельные решения, к каждому можно приложить один вопрос: чьё это суждение — приложения или модели? У каждого решения есть *владелец суждения* — то, что фактически выносит исход: приложение, заморозившее его на этапе проектирования, или модель, выносящая его на месте. Вопрос звучит просто, почти наивно, но именно он превращает разглядывание архитектуры в метод. Приложите его к первому ассистенту, решение за решением. Что искать по запросу? Решает механизм извлечения — владелец приложение. В какую ветку направить обращение? Решает роутер — владелец приложение. Повторить ли поиск, если найденное не отвечает на вопрос? Не решает никто, потому что конвейер одноходовый, — суждение заморожено в самой форме пайплайна, владелец приложение. Как оформить ответ? Задаёт шаблон — владелец приложение. Пройдитесь тем же вопросом по второму — и колонка владельцев меняется: что искать — модель, какого рода это вопрос — модель, повторить ли — модель, как ответить — модель. Один и тот же список решений, разные владельцы. Вот теперь различие двух систем не только видно, но и записано: не «этот умнее», а «у этих двух по-разному распределены владельцы одних и тех же суждений».

У этой записи есть немедленная практическая отдача: она показывает, где на самом деле лежит проблема. Когда первый ассистент отвечает мимо, привычный диагноз звучит как «плохой поиск» — и инженер идёт крутить механизм извлечения. Но карта владельцев говорит другое: решение о релевантности здесь вынесено приложением, и дело не в том, что поиск плохо ищет, а в том, что суждение о том, что искать, отобрано у модели. Диагноз по компонентам указывает на деталь; диагноз по решениям — на само решение и его владельца. Это разные адреса, и чинить, не различая их, — значит подолгу настраивать то, что не было причиной. Метод не обещает готового лекарства, он делает меньшее и более важное: точно называет, о чьём суждении идёт речь, — а без этого любое лечение бьёт наугад.

У этого вопроса есть свойство, делающее его несущим: он приложим к любому решению в любой системе и всегда даёт определённый ответ — исход либо застыл в коде, либо выносится моделью, третьего места ему нет. И потому «чьё это суждение?» работает не только на разборе чужих ошибок, но и как рабочий инструмент проектировщика над собственной системой: у каждого решения в системе есть владелец суждения, и этот вопрос всякий раз его называет.

Работает он в обе стороны времени. Назад — как диагноз: приложенный к готовой системе, он вскрывает, где суждение уже отобрано, и делает видимой раскладку, сложившуюся сама собой. Вперёд — как инструмент проектирования: заданный до того, как написана строка кода, тот же вопрос перестаёт быть вскрытием и становится развилкой. «Чьё будет вот это

суждение?» — спрошенное вовремя, оно возвращает автору выбор, который иначе сделался бы за него формой по умолчанию. Разница между двумя ассистентами возникла не потому, что кто-то из авторов ответил на этот вопрос неправильно, а потому, что один из них его не задал вовсе — и раскладку за него выбрал шаблон. Задать вопрос заранее — уже половина дела: изъятие, замеченное в момент проектирования, перестаёт быть случайным, даже если в итоге его оставляют.

Остаётся последнее, и оно важнее всего предыдущего. Владелец суждения — не свойство, приросшее к решению от природы. Это выбор. Кто-то — команда, автор пайплайна, стартовый шаблон, привычка поля — определил для каждого решения, застынет оно в приложении или останется за моделью. Чаще всего этот выбор не был сделан как выбор: его никто не проговаривал — просто написали стандартный пайплайн, в котором релевантность уже отобрана. Спросите инженера, унаследовавшего такую систему: «кто решил, что модель не должна повторять поиск, если найденное не подходит?» — и честный ответ будет «никто». Так решила форма: одноходовый конвейер не содержит места для второй попытки, и отсутствие этого места — тоже вынесенное суждение, только вынесла его не команда, а шаблон, из которого система выросла. Но незамеченный выбор — всё равно выбор, и у него всё равно есть автор и момент.

Метод на этом останавливается — и это намеренно. Назвать владельца суждения не значит осудить его. Часть замороженных владельцев стоит там по хорошей причине, и забирать у них решение было бы ошибкой; другие застыли по инерции и держатся только историей. Различить эти два случая — отдельная и более поздняя работа; вопрос «чьё это суждение?» её не делает, он лишь кладёт перед ней предмет. Он атом, а не вся процедура: полный разбор системы соберётся из множества таких вопросов и обрастёт критериями, но начинается всё с одного — приложенного к одному решению. Сначала увидеть владельца, потом судить о нём; в обратном порядке не выходит.

И как только на систему смотришь так — как на список решений, каждое со своим владельцем и каждый владелец назначен чьим-то выбором, — знакомый продакшен начинает читаться иначе. Не «вот моя архитектура», а «вот тридцать решений, и по каждому кто-то однажды выбрал, чьим оно будет; помню ли я, чтобы выбирал?».

Вопрос «чьё это суждение?» превращает разглядывание системы в разбор. Пока система — набор компонентов, о ней можно говорить только общими словами: удачная, неуклюжая, умная, тупая. Стоит разложить её на решения и над каждым спросить, чей исход, — и она перестаёт быть картинкой и становится списком: столько-то суждений, у каждого владелец, у каждого владельца — история назначения. Именно этот список, а не схема из прямоугольников, и есть настоящее устройство системы.

Два ассистента, с которых всё началось, теперь различимы до конца. Снаружи они по-прежнему одинаковы — тот же интерфейс, та же база, тот же класс модели. Но их различие больше не приходится списывать на неуловимое «один умнее». У них разные владельцы одних и тех же суждений: там, где первый заморозил решение в приложении, второй оставил его модели. Всё, что чувствовалось в открывающем контрасте и не имело имени, оказалось именно этим — картой владельцев, разложенной по-разному.

И остаётся вопрос, обращённый уже не к двум вымышленным ботам, а к системам читателя. Сколько суждений в них заморожено — и заметил ли он момент, когда это произошло? Большинство изъятий не обставлялись решением: они пришли формой по умолчанию и с тех пор не пересматривались. Значит, где-то в знакомой архитектуре стоят замороженные суждения, о которых никто не помнит, что когда-то выбрал их заморозить. Найти их — уже работа; но начинается она не с инструмента, а с одного вопроса, приложенного к каждому решению по очереди.

разбирай систему не на компоненты, а на суждения — у каждого решения в системе есть владелец, и это выбор, а не данность.

Глава 2. Флагманский случай: инъекционный RAG

Команда полгода улучшает извлечение. Меняет модель эмбеддингов на более точную, дробит документы на чанки поумнее, добавляет реранкер, который переупорядочивает найденное перед тем, как отдать его модели. На каждой итерации метрики сходства растут, а качество ответов бота упирается в стену, которую не видно на графиках. Стена не там, где её ищут. Она не в качестве компонентов — все они делают ровно то, чего от них ждали. Она в том, что модель, которая пишет ответ, ни разу не была спрошена, что именно ей искать.

Кто в системе владеет поиском — приложение или модель? Пока этот вопрос не задан, его ответ выглядит настолько само собой разумеющимся, что кажется не решением, а свойством мира: конечно, поиском занимается пайплайн, для того он и построен. Но это решение. Оно вынесено один раз, на этапе проектирования, и с тех пор не менялось. И у него есть цена — не в точности выборки, а в том, какие суждения оно отняло у модели и что стало невозможным вместе с ними.

Модель в такой системе отвечает по выборке, которой не заказывала, — так, как если бы выборка была исчерпывающей. Возразить против принесённого ей нечем: она не знает, что было доступно и чего не хватает. Это не глупость и не дефект обучения. Это слепота к собственному незнанию, встроенная в архитектуру: модель не видит границы того, что ей дали, потому что саму границу проводили без неё.

2.1. Инъекция против агентного извлечения

Возьмём привычную схему, по которой работает большинство ассистентов на базе знаний. Пользователь задаёт вопрос. Приложение превращает его в вектор, идёт в векторную базу — хранилище, где документы лежат как числовые представления смысла, — находит несколько наиболее близких фрагментов, при желании переупорядочивает их реранкером и вклеивает получившийся текст в промпт перед вопросом пользователя. Только теперь вызывается модель. Она видит вопрос и рядом — блок «вот релевантные документы», как будто он всегда там был. На этом блоке она и строит ответ.

Назовём эту механику по имени. *Инъекция (инъекционный RAG)* — схема, в которой приложение находит и вставляет контекст до вызова модели, а модель получает его как данность. Контекст именно впрыскивается — снаружи, готовым, без участия того, кто будет им пользоваться. RAG (retrieval-augmented generation) — генерация, дополненная извлечением; инъекционный RAG — та его разновидность, где извлечение целиком вынесено из модели в код вокруг неё.

Эта схема — не чья-то ошибка и не признак небрежности. Она стала стандартом по хорошей причине: когда модель нельзя было надёжно попросить самой сходить за нужным документом — сформулировать разумный запрос, не заблудиться, не выдумать источник, — единственным способом дать ей факты было положить факты рядом заранее. Приложение брало поиск на себя, потому что модель его не потянула бы. Инъекция — рациональный ответ на реальную задачу; узнать в ней свой продакшен не стыдно. Стыдно другое — не заметить, что решение, принятое под те условия, продолжает действовать, когда условия изменились. Но об этом уместно говорить, только предъявив сперва, что именно это решение замораживает.

У этой механики есть точный глагол. Инъекция *обходит* суждение модели о поиске: решение о том, что и где искать, принимается в обход того, кто будет отвечать. Не против модели — мимо неё. Приложение не спорит с моделью о выборке и не показывает ей альтернатив; оно просто ставит модель перед фактом. К моменту, когда модель включается в работу, поиск уже состоялся, его результат заморожен, и переиграть его нельзя — не потому что запре-

щено, а потому что механика не предусматривает такого хода. Модель здесь — последнее звено конвейера, а не его распорядитель.

Чтобы увидеть это не как схему, а как повседневность продакшена, проследим за одним вопросом к боту поддержки. Пользователь пишет: «После обновления перестал приходить вебхук на платёж — куда копать?» Приложение переводит эту фразу в вектор и уходит в базу знаний. В базе есть точная статья про вебхуки платежей, но её заголовок и текст построены вокруг слов «уведомление о транзакции», а пользователь написал «вебхук на платёж». Семантически это близко, но не настолько, чтобы нужный документ обошёл три статьи про вебхуки вообще — про их настройку, про повторную доставку, про подпись запроса, — которые по формулировке пользователя выглядят роднее. Приложение вклеивает эти три, статью про уведомления оставляет за порогом top-k — числа ближайших фрагментов, которые вообще попадают в выборку, — и вызывает модель. Модель получает три документа про вебхуки, ни один из которых не отвечает на вопрос про платежи после обновления, и добросовестно строит из них правдоподобный, уверенный и неверный ответ. Она не знает, что нужная статья существует и лежала рядом. Она не знает даже, что вопрос был про платежи, а не про вебхуки вообще, — потому что видела не вопрос, а его последствие: выборку. Это и есть обход в действии: суждение о том, где лежит ответ, вынесли за модель, а расплачиваться за него пришлось ей.

Вторая механика — пока только контраст, не рецепт. В *агентном извлечении* поиск зовёт сама модель. Он оформлен как инструмент, который модель может вызвать, когда сочтёт нужным: сформулировать запрос своими словами, посмотреть на результат, вызвать поиск ещё раз с другой формулировкой, обратиться к другому источнику. Приложение по-прежнему предоставляет механизм поиска — векторную базу, индекс, доступ к документам, — но не решает за модель, когда его применить и с каким запросом. Решение о поиске остаётся внутри работы над задачей, а не выносится перед ней.

Вернёмся к тому же вопросу про вебхук на платёж, но в этой второй механике. Модель, получив вопрос напрямую, видит именно вопрос, а не готовую выборку. Она может заметить, что «вебхук на платёж» — это, скорее всего, про уведомления о транзакции, и построить запрос под тот словарь, которым говорит база. Может, получив три статьи про вебхуки вообще, увидеть, что ни одна не про платежи после обновления, и сходить ещё раз. Может понять, что вопрос распадается на два — про платёж и про обновление — и поискать по каждому. Ничего из этого не гарантировано: модель тоже способна ошибиться формулировкой или остановиться рано. Но сама возможность так поступить — есть. В первой механике её нет: там ни один из этих ходов недоступен не потому, что модель не додумалась, а потому, что к моменту её включения поиск уже закончился. Разница не в том, что вторая механика умнее. Разница в том, что в ней суждение о поиске вообще существует как живое решение, а не как застывший в коде результат.

Различие это — не в качестве. Обе схемы могут стоять на одной и той же векторной базе, с одной и той же моделью эмбеддингов, с одинаково хорошим реранкером. Снаружи, в демонстрации на подготовленных вопросах, они могут быть неотличимы: обе находят релевантные документы, обе отвечают по делу. Разница обнажается там, где вопрос не предусмотрен, где первая выборка мимо, где ответ требует нескольких заходов, — и её источник не в компонентах, а в том, кто владеет поиском.

Это первый полномасштабный ответ на вопрос, который проходит через всю книгу: чьё это суждение? Суждение «что и где искать» — не техническая мелочь, а именно суждение: оно требует понять, что на самом деле спросил пользователь, какими словами это найдётся в базе, в каком из источников это вообще может лежать, и достаточно ли одного поиска. В инъекции всё это суждение целиком принадлежит приложению: оно сформулировано в коде — как строить запрос, куда идти, сколько фрагментов брать — и вынесено один раз, до и вместо

модели. В агентном извлечении то же суждение принадлежит модели: она выносит его каждый раз заново, под конкретный вопрос, глядя на то, что уже нашла.

Разобрать систему по решениям и спросить о каждом «чьё это суждение?» — это и есть рабочий метод. В приложении к поиску он даёт точный диагноз, а не общее ощущение. Мы не спрашиваем, хорош ли пайплайн; мы берём одно решение — «с каким запросом и куда идти за ответом» — и смотрим, где оно вынесено и кем. И тут важно слово «заморожено». В инъекции суждение о поиске принимается не в тот момент, когда приходит вопрос пользователя, а гораздо раньше — когда инженер писал код извлечения. Там, на этапе проектирования, было решено: строить запрос вот так, ходить вот сюда, брать вот столько фрагментов. Это решение застыло в коде и с тех пор применяется к каждому вопросу одинаково, каким бы вопрос ни был. Оно не пересматривается под конкретный запрос не потому, что кто-то запретил его пересматривать, а потому, что пересматривать его в этой механике некому: единственный участник, который видит конкретный вопрос целиком, — модель — включается уже после того, как решение применено.

Стоит задержаться на слове «владеет». Речь не о том, кто физически выполняет поиск, — в обеих схемах по индексу ходит один и тот же код. Речь о том, кто выносит решение: с каким запросом пойти, стоит ли идти ещё раз, откуда брать. В инъекции это решение вынесено в код и застыло там; модель его не касается. В агентном извлечении механизм поиска остаётся кодом, но решение о его применении возвращено модели. Владелец поиска — это владелец суждения о поиске, а не исполнитель операции. И именно владелец, а не качество компонентов, определяет, что система сможет, когда вопрос выйдет за пределы предусмотренного.

Различие владельца не привязано к docs-ботам — оно живёт в любой системе, где есть поиск. Ассистент юриста, который по запросу «прецеденты по расторжению из-за скрытого дефекта» либо сам решает, что стоит развести «скрытый дефект» и «существенное нарушение» на два поиска, либо получает единый список, собранный кодом по исходной формулировке. Помощник аналитика, который либо сам выбирает, идти ли в базу отчётов или в базу транзакций, либо всегда ходит туда, куда его направил маршрут в приложении. В каждом случае две системы можно поставить рядом, дать им одинаковый индекс и одинаковую модель — и снаружи, на подготовленной демонстрации, не различить. Различие проступит на первом же вопросе, который не лёг в предусмотренную колею: там, где одна система переспросит саму себя, а другая ответит по единственной выборке, какой бы та ни была.

Два ассистента на одной базе знаний, с одной моделью, за одинаковым интерфейсом могут оказаться разными системами — не потому что у одного лучше эмбеддинги, а потому что в одном суждением о поиске владеет приложение, а в другом — модель. Владелец назван. Осталось предъявить счёт: что именно теряет модель вместе с поиском, когда суждение о нём заморожено в коде. Потеря эта не абстрактна — она раскладывается на несколько совершенно конкретных способностей, каждую из которых легко узнать по тому, как система ведёт себя, когда первая выборка оказывается не той.

2.2. Четыре изъятые способности

Когда поиск вынесен из модели, отнимается не «немного качества». Отнимаются вполне конкретные способности — те самые, которыми живой специалист поддержки пользуется, не задумываясь, когда ищет ответ в документации. Их четыре: переформулировать запрос, сходить за ответом ещё раз, разбить сложный вопрос на части и выбрать, где искать. Все четыре исчезают не по отдельности и не из четырёх разных дефектов — их снимает один архитектурный жест: решение искать до вызова модели. Стоит сделать этот жест — и все четыре способности выпадают разом, потому что каждая из них требует, чтобы у модели было то, чего инъекция ей не оставляет, — право распоряжаться поиском.

У живого специалиста эти четыре хода настолько привычны, что не осознаются как отдельные умения: он переводит жалобу на язык базы, пробует ещё раз, если не нашёл, раскладывает запутанный вопрос и знает, в какой папке лежит нужное, — и всё это за секунды, не называя про себя ни одного из этих действий. Именно потому, что в человеке они невидимы, их отсутствие в системе так легко проглядеть: трудно хватиться способности, которую никогда не замечал. А между тем это и есть та самая абстрактная формула из предыдущего разбора — «поиском владеет приложение», — разменянная на осязаемое. Владеть поиском значит владеть вот этими четырьмя суждениями; отдать поиск приложению значит отнять у модели ровно их. Вопрос о владельце потому и не академический: за ним стоит конкретный список того, что система теперь не умеет.

Первая способность — переформулировать запрос. Пользователь почти никогда не спрашивает на языке базы знаний. Он пишет так, как чувствует проблему: «он тупит, когда жму сохранить». В базе про это есть точная статья, но называется она «Конфликт ручного и автоматического сохранения», и индексирована под словами, которых в жалобе пользователя нет вовсе. Специалист, прочитав жалобу, мысленно переведёт её: «тупит при сохранении» — это, скорее всего, про автосохранение, надо искать по нему. Модель сделала бы тот же перевод — если бы увидела жалобу. Но она её не видит: приложение превращает в вектор исходную фразу как есть и приносит модели статьи, ближайшие к словам «тупит» и «сохранить» — что-то про производительность интерфейса, мимо сути. Изъято здесь суждение о том, как назвать проблему на языке базы: право превратить кривой вопрос пользователя в точный поисковый запрос вынесено за модель — а точнее, не вынесено никуда, потому что в этой механике его просто некому вынести. Что при этом получит пользователь, предсказуемо: модель, послушно опираясь на принесённые ей статьи о производительности интерфейса, объяснит, как ускорить отрисовку, — исчерпывающий ответ на вопрос, которого никто не задавал.

Вторая способность — сходить за ответом ещё раз. Первая выборка не всегда попадает; это нормально, так и у человека. Разница в том, что человек, увидев мимо, ищет иначе, а инъекция даёт ровно один заход. Пользователь спрашивает: «как отменить подписку». Поиск по близости слов приносит «Как оформить подписку» и «Сравнение тарифов» — семантически рядом, по намерению мимо. Специалист бросил бы один взгляд на эти статьи, понял бы, что все они про заведение и выбор подписки, а не про её прекращение, и переспросил бы поиск словом «расторжение» или «отмена». Модель, получи она эту выборку с возможностью сходить снова, поступила бы так же. В инъекции возможности нет: что нашлось с первого раза, из того и придётся отвечать. Изъято суждение о том, годится ли выборка и стоит ли искать заново, — решение «повторить ли поиск» заморожено на «нет». Человек, спросивший, как отменить подписку, получит вежливую инструкцию, как её оформить, — и это худший сорт ошибки, тот, что выглядит как ответ.

Третья способность — разбить сложный вопрос на части. Некоторые вопросы составные: ответ на них собирается из нескольких мест, и один поиск их не обслуживает. Аналитик спрашивает ассистента: «сравни выручку по региону за прошлый квартал с планом и объясни, где расхождение больше нормы». Это не один запрос, а по меньшей мере три: факт по выручке, план, порог нормы — и лежат они, возможно, в разных таблицах и разных документах. Единый поиск по всей фразе принесёт что-то усреднённо-близкое ко всему сразу и точно ни к чему. Человек, взявшись за такой вопрос, разложил бы его на шаги и сделал бы несколько прицельных поисков. Модель способна на ту же декомпозицию — но только если поиск в её руках. В инъекции составной вопрос обслуживается одним заходом, потому что заход всегда один и планирует его не модель. Изъято суждение о том, что вопрос надо разложить и поискать по каждой части, — право спланировать извлечение под структуру задачи. Аналитик получит связный абзац про выручку и план и ни слова про порог нормы — не потому, что модель не

справилась с вопросом, а потому, что о пороге ей ничего не принесли, и узнать, что этой части не хватает, ей неоткуда.

Четвёртая способность — выбрать, где искать. У ассистента редко один источник. Есть основная документация, есть журнал изменений с описанием того, что поменялось в последних версиях, есть внутренние регламенты, есть, может быть, форум сообщества. Разные вопросы живут в разных источниках. Пользователь спрашивает: «почему после обновления изменилось поведение выгрузки». Ответ — в журнале изменений, где прямо записано, что в этой версии выгрузку переделали. Но приложение всегда ходит в основную документацию, потому что так настроен маршрут; журнал изменений в этот маршрут не входит. Модель, будь выбор источника за ней, сообразила бы, что вопрос про «после обновления» — это вопрос к журналу изменений, а не к общей документации. Она этого не сделает: куда идти, решено до неё и помимо неё. Изъято суждение о выборе источника — о том, из какого хранилища вообще может прийти ответ. Пользователь про «после обновления» получит уверенное описание старого поведения из основной документации, которую никто не удосужился обновить, — правильный ответ из неправильного источника.

Эти четыре потери к тому же не складываются, а перемножаются. Кривой запрос без права переформулировать — полбеды, если можно сходить ещё раз; но итерация тоже изъята, и первый неудачный заход оказывается единственным. Составной вопрос без декомпозиции ещё как-то вытянул бы верный источник — но и выбор источника заморожен, так что промах по формулировке и промах по хранилищу накладываются друг на друга. Отняв поиск, инъекция отняла не четыре независимые мелочи, а связку: каждая уцелевшая способность могла бы прикрыть отказ соседней, но уцелевших нет. Модель остаётся с одной-единственной выборкой, собранной чужим суждением по исходной формулировке, — и на этой выборке обязана построить ответ, каким бы промахом та ни оказалась.

Эти четыре — не список неудобств, а список конкретных суждений, которые в инъекции принадлежат приложению, а не модели: как назвать вопрос для поиска, стоит ли искать снова, надо ли разбить вопрос на части, где искать. Каждое из них модель могла бы вынести сама — и вынесла бы не хуже кода, а на нестандартном вопросе почти наверняка лучше, потому что видит конкретный вопрос целиком, а код видит только заранее заданную схему. Все четыре сняты одним и тем же жестом и по одной и той же причине: к моменту, когда модель начинает работать, поиск уже позади. Именно этот перечень позже придётся вернуть модели по одному — он и есть точная опись того, что заморожено.

Здесь стоит понять, почему эти потери так долго остаются незамеченными. На предусмотренных вопросах их не видно вовсе. Пользователь, который спрашивает ровно то и теми словами, под какие настраивали пайплайн, получает точную выборку с первого захода — переформулировать нечего, повторять незачем, вопрос простой, источник единственный. Ни одна из четырёх способностей на таком вопросе не нужна, а значит, их отсутствие ничем себя не выдаёт. Демонстрация на подготовленных примерах проходит блестяще; метрики сходства высоки; команда уверена, что система работает. Способности всплывают только там, куда демонстрация не заглядывает: на вопросе не теми словами, на выборке мимо, на составном запросе, на ответе из соседнего источника. Изъятие невидимо ровно до той секунды, пока оно не начинает стоить денег, — а к этой секунде оно уже вшито в архитектуру. И чем лучше система отвечает на предусмотренное, тем убедительнее выглядит, будто с ней всё в порядке, — тем дальше отодвигается та секунда и тем неожиданнее будет расплата.

И вот тут срабатывает инженерный рефлекс, знакомый каждому, кто такие системы строил: раз выборка бывает мимо — улучшим выборку. Возьмём модель эмбеддингов посильнее, нарежем документы аккуратнее, добавим реранкер получше. Рефлекс верный по инстинкту и бесполезный по адресу: он лечит не ту болезнь.

2.3. Почему тюнинг пайплайна этого не лечит

Рефлекс «улучшим выборку» заслуживает того, чтобы отнестись к нему всерьёз, а не отмахнуться. За ним стоит настоящая инженерная работа и настоящий результат. Более сильная модель эмбеддингов действительно располагает документы в смысловом пространстве точнее, так что близкое по смыслу оказывается близким и по вектору. Аккуратная нарезка на чанки действительно спасает от того, что ответ разорван между двумя фрагментами и ни один не попадает в выборку целиком. Хороший реранкер действительно поднимает наверх то, что релевантнее, отодвигая правдоподобный мусор. Всё это не самообман и не карго-культ: качество одноходового поиска этими средствами растёт, и растёт измеримо. Инженер, который этим занимается, не заблуждается — он честно решает реальную задачу.

Но задача эта — не та, что стоит на пути. Четыре потери из предыдущего разбора — не следствие плохой выборки; они следствие того, что выборка одна и собрана до модели. Это свойство механики, а не качества компонентов. Пассивность модели не берётся из слабых эмбеддингов и не лечится сильными: она берётся из места, которое модель занимает в конвейере, — ниже по течению от поиска. Можно довести единственный заход до идеала — он останется единственным заходом. Можно сделать выборку безупречно релевантной исходной формулировке — она останется выборкой под исходную формулировку, а не под то, что пользователь имел в виду. Компонент отвечает за то, насколько хороша выборка; он ничего не может сказать о том, кто решает, какой выборке быть, сколько их будет и откуда. Это разные вопросы, и живут они на разных осях.

Проще всего увидеть это через мысленный предел. Представим, что качество выборки доведено до совершенства: на любом предусмотренном вопросе пайплайн приносит идеально релевантный фрагмент с первого раза. Пассивность модели от этого никуда не делась — она просто перестала быть заметной, пока вопросы предусмотрены. Стоит прийти вопросу, которого схема не ждала, и совершенное качество выборки не поможет ничем: модель по-прежнему не может переформулировать, переспросить, разложить или сменить источник. Дефект, который тут вылезет, — не «выборка плоха», а «модель не властна над выборкой», и второе не выводится из первого никакой настройкой. Пассивность — это не низкое качество того, что принесли; это отсутствие у модели самого права распорядиться тем, что и как приносят. Оно вписано в позицию, а позицию тюнинг не трогает.

Разведём эти две оси прямо, потому что их постоянно путают. Первая ось — качество выборки: насколько хорошо найденное отвечает заданному запросу. По этой оси тюнинг двигает систему вперёд, и это ценно. Вторая ось — владение поиском: кто формулирует запрос, кто решает, что выборка не годится и надо искать снова, кто разбивает вопрос и выбирает источник. По этой оси тюнинг не двигает ничего. Реранкер, каким бы точным он ни был, переупорядочивает то, что уже найдено, — он не может поднять наверх документ, который не был извлечён, потому что запрос под него никто не переформулировал. Лучшая модель эмбеддингов сделает единственный заход точнее — но не превратит его в два захода, если первый промахнулся. Улучшение компонентов оптимизирует выстрел, не отдавая модели права выбрать, куда стрелять, стрелять ли ещё раз и из какого ствола. Оно поднимает потолок одноходового поиска и даже слегка расширяет коридор предусмотренных вопросов — но не делает поиск управляемым. Управляемость лежит на второй оси, а тюнинг всё это время работает на первой.

У этой второй оси есть имя. Владение поиском — это и есть вопрос «чьё это суждение?», приложенный к извлечению. Первая ось на него не отвечает и даже не задаёт его: она молча принимает, что суждение о поиске принадлежит коду, и оптимизирует то, что код принёс. Можно всю жизнь совершенствовать ответ на вопрос, не заметив, что сам вопрос вынесен

мимо модели. Тюнинг потому и не сдвигает вторую ось, что не касается её предмета: он делает выборку лучше, ни разу не спросив, кому она принадлежит.

Это видно на любой попытке пройти полный цикл улучшений. Команда меняет модель эмбеддингов, перенарезает документы, ставит новый реранкер, прогоняет тестовый набор — и на простых, предусмотренных вопросах метрики растут, выборка стала заметно чище. А потом приходит вопрос вроде «поддерживает ли тариф Pro единый вход и ведётся ли на нём журнал действий» — и система снова отвечает половину. Про единый вход — да, поддерживает, точно и уверенно; про журнал действий — молчание, потому что документ про журналы лежит отдельно, а поиск был один и вытянул то, что ближе к первой половине фразы. Никакой реранкер этого не исправит: нельзя переупорядочить документ, который не был извлечён, потому что второго запроса не было. Команда может тюнинговать этот пайплайн ещё полгода — конкретно этот класс вопросов останется там же, где был, потому что промах здесь не по качеству выборки, а по числу и адресности заходов.

Отсюда — точная граница, которую важно провести, чтобы не свалиться в ложную крайность. Тюнинг пайплайна не бесполезен и не «устарел»; он делает ровно то, для чего нужен, — повышает качество выборки, — и на предусмотренных вопросах этого хватает. Отказываться от него ради красивого лозунга было бы такой же ошибкой, как надеяться, что он вернёт модели поиск. Стоит быть к тюнингу справедливым и на конкретном: бывают провалы, которые лечатся именно им. Если ответ разорван между двумя чанками так, что ни один не попадает в выборку целиком, — это дефект нарезки, и аккуратное перечанкивание его чинит. Если синонимичные формулировки не сходятся в векторном пространстве и близкое по смыслу оказывается далёким по вектору — это дефект эмбеддингов, и модель посильнее его снимает. Это настоящие болезни выборки, и тюнинг — их настоящее лекарство. Разница в том, что все они лежат на первой оси: качество того, что приносит единственный заход. Ни одна из них не про то, кто решает, каким заходам быть. Возврат поиска — это движение по другой оси, другой архитектурный ход. Здесь достаточно понять, что улучшать выборку и возвращать суждение — не одно и то же и не соседние точки на общей шкале: это два перпендикулярных направления, и сколько ни двигайся по первому, к началу второго не приблизишься ни на шаг.

И вот теперь диагноз можно поставить целиком.

Инъекция не делает систему глупее — она делает её слепой к собственному незнанию. Модель получает выборку, которую не запрашивала, отвечает на вопрос, который не формулировала, и не может сказать: «мне принесли не то». Пайплайн можно тюнинговать бесконечно — суждение это не вернёт. Замороженное суждение не ошибается. Оно просто не пересматривается.

В этих двух последних фразах — вся разница между дефектом и конструкцией. Дефект можно исправить, доведя компонент до качества; конструкцию исправить нельзя, её можно только переустроить. Замороженное суждение не совершает ошибок в том смысле, в каком их совершает плохой реранкер: реранкер иногда ставит не тот документ первым, и это чинится настройкой. Замороженное суждение вообще не участвует в происходящем — оно вынесено один раз и с тех пор просто действует. У него нет режима, в котором оно могло бы посмотреть на конкретную выборку и сказать «этого мало» или «это не оттуда». Ошибку реранкера видно — она ловится глазом и метрикой. «Ошибку» замороженного суждения увидеть нельзя, потому что оно не выносит вердикта, в котором можно ошибиться: оно молчит и применяется. Слепота к собственному незнанию — это не то, что можно вылечить более качественной выборкой; более качественная выборка просто аккуратнее укладывается в ту же слепую зону. Сколько ни улучшай то, что модель видит, ты не даёшь ей увидеть то, чего она не видит, — а именно там, в невидимом, и лежит цена.

Так объясняется стена. Команда, месяцами двигавшая метрики сходства, упиралась не в предел компонентов — компоненты ещё можно было улучшать. Она упиралась в то, что всё это

улучшение шло по одной оси, а стена стояла на другой. Это и делает её стеной, а не склоном: по склону качества можно карабкаться сколь угодно долго, но он не выводит на ось владения — они не пересекаются. Диагноз этот, впрочем, поставлен пока снаружи: мы сказали, чего нельзя вылечить и почему. Осталась внутренняя сторона того же диагноза — что представляет собой модель, помещённая внутрь этой механики, и как называется положение, в котором она оказывается.

2.4. Модель как пассивный получатель контекста

Соберём главу в одну точку — в положение, которое модель занимает внутри инъекции. Оно описывается одним словом: получатель. Модель здесь не автор запроса. Вопрос, на который она отвечает, — не тот, что задал пользователь, и не тот, что сформулировала бы она сама, а тот, что молча составил код, превратив пользовательскую фразу в поисковый запрос и выборку. Модель получает результат этого чужого запроса как исходные данные и не имеет двух вещей, без которых нельзя владеть поиском: она не может переспросить — уточнить, переформулировать, сходить ещё раз, — и не может отклонить принесённое — сказать «эта выборка не отвечает на вопрос, дайте другую». Переспросить и отклонить — это и есть минимум субъектности в поиске; ни того, ни другого у получателя нет.

Видно это лучше всего там, где выборка приходит совсем мимо. Пользователь спрашивает о способности, которой в продукте попросту нет, — а значит, и в базе знаний про неё ничего. Поиск, обязанный что-то вернуть, возвращает ближайшее по словам: три статьи о смежных, похоже называющихся вещах. Живой специалист на этом месте остановился бы и сказал: «в документации этого нет, дайте уточню» или «покажите, где вы это видели». Модель-получатель так не делает — не потому, что не догадалась бы, а потому, что «остановиться и переспросить» не входит в её ход. Ей передали выборку как факт, и единственное действие, которое механика от неё ждёт, — ответить по этой выборке. И она отвечает: аккуратно сшивает три смежные статьи в складный ответ о способности, которой не существует. То, что модель не отвергла плохую выборку и не запросила другую, — не слабость, которую можно списать на модель; это ход, которого в позиции получателя просто нет. Получатель не отклоняет принесённое, потому что не ему решать, что приносить.

Четыре потери из разбора способностей — не четыре независимых увечья, а четыре грани этого одного положения. Нельзя переформулировать, потому что запрос составлен до тебя. Нельзя повторить, потому что заход не твой. Нельзя разложить вопрос, потому что план поиска не твой. Нельзя выбрать источник, потому что маршрут не твой. Всё это — один и тот же факт, увиденный с четырёх сторон: поиск принадлежит не модели. Отними у модели любую из четырёх способностей поодиночке — получишь частную неполноту; отними их разом, вынеся поиск целиком, — получишь пассивного получателя, и остальное следует само.

Стоит вглядеться в саму фигуру получателя, потому что она обманчива. Получателю можно приносить превосходные вещи. Идеально настроенный пайплайн — это очень хороший поставщик: он приносит получателю чистую, релевантную выборку. Но получатель и с превосходной выборкой остаётся получателем: он не выбирал, что ему принесут, не знает, что осталось непринесённым, и не может послать за другим. Качество подношения не меняет положения того, кому подносят. Именно поэтому пассивность нельзя перепутать с низким качеством: пассивен не тот, кому принесли мало, а тот, кому приносят, не спрашивая.

Слово «пассивный» здесь — не упрёк модели и не про её способности. Одна и та же модель, поставленная в другое положение, повела бы себя иначе; пассивность — свойство места, а не того, кто на нём стоит. Назвать модель пассивным получателем — значит описать не её характер, а её должность в этой конкретной механике. И должность эту назначила не она сама и не чей-то злой умысел: её назначил разумный в своё время выбор — искать до модели.

Пассивность модели — оборотная сторона активности приложения: ровно в той мере, в какой поиском распоряжается код, модель от поиска отстранена.

У того, что мы разглядывали, есть имя. Когда суждение, которое модель могла бы вынести сама, принимается и замораживается за неё внешней логикой, происходит *изъятие агентности*: у модели забирают контроль над суждением внутри решения задачи. Инъекционный RAG — первый и самый наглядный случай этого паттерна: суждение о поиске изъято у модели и заморожено в коде приложения. Название это шире самой механики: изъять можно не только поиск.

Название это опирается прямо на то, чем агентность была определена раньше, — контроль над суждением внутри решения задачи. Если агентность есть мера, в которой модель сама выносит суждения, то изъятие агентности — обратная операция: у модели забирают этот контроль и передают его коду. Пассивный получатель контекста — это модель, у которой изъяли агентность в поиске: не потому что она её не заслужила и не потому что не справилась бы, а потому что решение о поиске вынесли за неё. Паттерн, таким образом, — не новая сущность, а имя для того, что происходит с агентностью, когда суждение замораживают снаружи. Оно понадобится всякий раз, когда мы будем показывать это «снаружи» в новом месте.

Почему «изъятие агентности», а не просто «изъятое суждение»? Потому что здесь работают два уровня, и их полезно различать с самого начала. *Изъятие суждения* — операционный уровень: разбор одного конкретного решения, та самая единица анализа, к которой мы прикладываем вопрос «чьё это суждение?». «С каким запросом искать» — одно изъятое суждение; «повторить ли поиск» — другое; «где искать» — третье. Каждое можно взять по отдельности, указать пальцем, где оно вынесено и кем. *Изъятие агентности* — уровень системы: то, что складывается из этих отдельных изъятий, когда их достаточно, чтобы модель перестала быть распорядителем целой области решений. Агентность в поиске изъята не каким-то одним из четырёх изъятий, а всеми вместе: по одному отняли переформулировку, повтор, декомпозицию и выбор источника — и в сумме отняли у модели поиск как таковой.

Связь между уровнями простая и несущая: агентность изымается через изъятие конкретных суждений. Нет отдельного акта «изъятия агентности» помимо изъятия суждений — есть суждения, вынесенные наружу по одному, и есть их сумма, которую мы называем изъятой агентностью. Это различение — рабочий инструмент, а не терминологическая тонкость. Операционный уровень говорит, где именно искать заморозку: в конкретных решениях, каждое из которых можно назвать и проверить. Системный уровень говорит, что складывается в итоге: область, которой модель больше не распоряжается. Держа оба уровня в руках, читатель вооружён точнее, чем если бы у него было только общее ощущение «модели тут мало»: он может показать, из каких именно замороженных суждений собрана эта нехватка.

Приложить этот инструмент к инъекции можно почти построчно. Место в коде, где вопрос пользователя без изменений превращается в поисковый запрос, — вот здесь заморожено суждение «как назвать вопрос для поиска». Отсутствие в схеме второго обращения к поиску — здесь заморожено «повторить ли». Единственный вызов вместо цикла по частям вопроса — заморожено «разбивать ли». Жёстко заданный адрес одного индекса — заморожено «где искать». Четыре места, четыре изъятых суждения; их сумма и есть изъятая у модели агентность в извлечении. Инструмент не заставляет спорить, «хорошая» это архитектура или «плохая», — он лишь делает видимым, где вынесено каждое суждение и кем. Пока достаточно уметь показывать пальцем.

Именно эта пара уровней — то, что читатель уносит из главы. Названный на одном случае, паттерн на нём не заканчивается. Если изъятие агентности — это сумма замороженных суждений, то везде, где суждение заморожено снаружи, работает тот же паттерн, каким бы ни было само суждение. Поиск оказался удобной первой иллюстрацией — наглядной, знакомой,

с чётко считываемыми четырьмя изъятиями. Но ничто в определении не привязывает его к поиску.

Стоит на минуту приложить этот инструмент не к учебному примеру, а к системе, которую читатель знает лучше всех, — к своей. Где в ней вопрос пользователя уходит в поиск ровно так, как пришёл? Сколько в ней заходов — один или столько, сколько понадобится? Кто выбрал источник — модель на этом запросе или инженер полгода назад на всех сразу? Вопросы неудобные, потому что ответы чаще всего известны заранее: один заход, исходная формулировка, единственный индекс, выбранный однажды. И тогда всплывает то, что труднее всего заметить в собственной системе, — не то, что она делает плохо, а то, чего она не делает вовсе, потому что права на это у модели нет. Сколько таких замороженных суждений в системе читателя — и помнит ли он момент, когда каждое из них замерзало? Ответа здесь нет — но вопрос уже нельзя развидеть.

Слепота к собственному незнанию теперь читается иначе. Она никогда не была дефектом зрения модели — тем, что можно было бы поправить, обучив модель лучше или дав ей выборку почище. Она конструкция. Модель ослепили не в бою, а на чертеже: суждение о поиске вынесли из неё и заморозили в коде на этапе проектирования, и вместе с ним вынесли саму возможность увидеть, чего в выборке недостаёт. Пассивный получатель слеп к собственному незнанию не потому, что глуп, а потому, что так устроено место, на которое его поставили. Агентность здесь изъята — и изъята не абстрактно, а совершенно конкретно: через четыре замороженных суждения, каждое из которых мы назвали.

И если так устроен всего лишь поиск — почему тот же почерк узнаётся повсюду? Решение о том, что делать с запросом, вынесенное в роутер до модели. Решение, повторять ли попытку, зашитое в одноходовый конвейер. Решение, как ответить, закреплённое жёстким шаблоном. Решение, что запомнить, отданное внешнему управлению памятью. Всякий раз — суждение, которое модель могла бы вынести сама, вынесенное за неё и застывшее снаружи. Инъекция оказалась не единственной комнатой, а первой дверью в длинный коридор, и вопрос, который она задаёт, больше её самой: если суждение можно отнять у поиска, какое ещё суждение уже отнято — и замечено ли это.

Отсюда — формула. Не выборка была плоха и не модель туга: изъяли не качество ответа, а право его искать.

инъекция отнимает не точность, а суждение.

Глава 3. Паттерн целиком: таксономия изъятия

Почерк, однажды разобранный, проступает повсюду. В инъекционном RAG решение о том, что искать, вынесено за модель и застыло в пайплайне — но стоит перевести взгляд на остальную систему, и тот же жест обнаруживается там, где его не ждёшь. Роутер намерения решает за модель, чем ей заняться. Конвейер без петли запрещает ей переиграть неудачный ход. Шаблон отвечает вместо неё — ещё до того, как она сформулировала, что́ отвечать. Харнесс наводит порядок в её памяти, оставляя одно и вычёркивая другое. Пять разных подсистем, пять бригад инженеров, пять обоснований, написанных в пять разных кварталов, — и один и тот же жест в пяти одеждах: суждение, которое могла бы вынести модель, вынесено снаружи и заморожено.

Разложить этот жест по типам — значит получить не список технологий, а классификацию по одному основанию: какое именно суждение отнято. Что делать. Повторить ли. Как ответить. Что помнить. Действовать ли. У этого жеста есть глаголы, точные до буквальности: роутер решение перехватывает, инъекция его обходит, суммаризация подменяет. Разные глаголы описывают разную механику, но общий знаменатель один — чужое замороженное суждение в том месте, где могло бы жить живое.

Такая таксономия работает как проявитель. Приложенная к собственной системе, она превращает то, что на схеме называлось «архитектурой» и «обвязкой», в перечень изъятых суждений — по одному на каждый заботливо выстроенный контур контроля. И тем же движением она ставит вопрос, который до времени останется открытым: если изъятие обнаруживается всюду, то было ли оно всюду ошибкой — или где-то это трезвый выбор, а где-то просто привычка, пережившая свою причину.

3.1. «Что делать»: оркестраторы и роутеры намерения

Первое решение, которое система выносит о запросе, часто принимает не модель. Пользователь пишет ассистенту поддержки: «после обновления приложение не открывается на втором устройстве». Прежде чем эти слова дойдут до модели, их рассматривает роутер намерения — классификатор, относящий входящий запрос к одной из заранее заданных категорий и направляющий его по соответствующей ветке. «Техническая проблема, подкатегория „синхронизация“» — значит, ветка синхронизации: поднять свой набор документов, задать заготовленную пару уточняющих вопросов, при неуспехе завести тикет нужного типа. Модель включится позже, уже внутри ветки. Но развилка пройдена без неё. Чьё это суждение — какую способность применить к запросу?

Ответ вшит в саму конструкцию: суждение вынес роутер. Он посмотрел на запрос, решил, к какому классу задач тот относится, и тем самым определил, чем система займётся дальше, — какие документы поднимать, какие вопросы задавать, какой инструмент считать уместным. Всё это суждения о том, «что делать». И все они приняты до модели и за модель.

Роутер редко приходит один. Обычно он — часть оркестратора: управляющего слоя, задающего заранее прописанную последовательность шагов над запросом. Классическая обвязка выглядит так: сначала классифицировать намерение, затем по намерению выбрать источник, затем извлечь, затем сгенерировать ответ по шаблону ветки, затем прогнать его через фильтр. Каждая стрелка на этой схеме — заранее вынесенное решение. Последовательность неизменна, выбор ветки жёсток, переход между шагами не обсуждается. Инженер, рисовавший эту схему, отвечал на честный вопрос: как сделать поведение системы предсказуемым и покрыть известные случаи? Жёсткая цепочка шагов и классификатор на входе — прямой и разумный ответ на него.

Полезно задержаться на самой цепочке шагов, потому что её жёсткость обманчиво выглядит нейтральной. «Классифицировать → выбрать источник → извлечь → ответить по шаблону → отфильтровать» читается как техническое описание потока данных, а не как последовательность решений. Но каждая стрелка — замороженное суждение, и их легко перечислить поимённо. Стрелка от классификации к выбору источника решает, что для этой категории релевантна вот эта база и никакая другая. Стрелка к шаблону решает, что ответ такого типа выглядит именно так. Порядок стрелок решает, что извлечение всегда предшествует рассуждению, а не наоборот, — хотя встречаются запросы, где сначала стоило бы подумать, а потом искать. Ни одно из этих решений не является технической неизбежностью; каждое — выбор, сделанный однажды и вшитый в граф. То, что решения выстроены в ряд и подписаны стрелками, маскирует их природу: это не поток данных, это очередь вынесенных за модель суждений, каждое из которых могло бы приниматься заново под конкретный запрос.

Тот же контур обнаруживается далеко за пределами поддержки. В разборе входящей почты классификатор сортирует письма по темам и раздаёт их предопределённым обработчикам. В системе, читающей юридические документы, диспетчер по типу договора решает, какой из узкоспециализированных конвейеров запустить. В голосовом помощнике распознаватель намерения сопоставляет фразу с одним из поддерживаемых сценариев и отбрасывает всё, что не легло в список. Механика везде одна: перед моделью стоит слой, который смотрит на задачу и назначает ей маршрут. Модель получает уже не задачу «разберись, что здесь нужно», а узкую подзадачу «выполни шаг ветки N», выбранную за неё.

Внутри ветки модель работает добросовестно — и именно поэтому изъятие незаметно. Ей выдали подзадачу «задай эти три вопроса и, если ответы не совпали с ожидаемыми, заведи тикет», и она выполнит её безупречно. Она не откажется, не усомнится, не спросит, почему её вообще позвали сюда: у неё нет доступа к развилке, на которой решили, что это задача про синхронизацию. Возьмём тот же запрос про второе устройство. Настоящая причина — просроченный токен авторизации, и способная модель распознала бы это по одному уточнению. Но ветка синхронизации про токены не спрашивает: её сценарий писали под другую гипотезу. Модель прилежно ведёт пользователя по вопросам о сети и версии приложения, пользователь честно отвечает, тикет заводится, круг замыкается — и ни в одной точке этого круга не нашлось места, где чей-то интеллект мог бы сказать: развилка выбрана неверно. Интеллект в системе есть, но он заперт ниже той развилки, где его суждение и было нужно.

Редко когда перехват случается однажды. Зрелые системы наслаивают роутеры: верхний классификатор делит запросы на крупные домены, внутри домена другой выбирает подсценарий, внутри подсценария селектор инструментов назначает конкретное действие. Каждый слой — ещё один перехват, ещё одно суждение «что делать», вынесенное до модели. К тому моменту, когда модель наконец получает управление, большинство решений о задаче уже приняты за неё каскадом классификаторов, а её роль сжата до исполнения листа в чужом дереве. Дерево это росло из лучших побуждений — каждая новая ветка добавлялась в ответ на реальный непокрытый случай, — но сумма побуждений даёт архитектуру, в которой суждение о задаче раздроблено между десятком замороженных развилок и нигде не принадлежит тому, кто способен увидеть запрос целиком.

У этого типа изъятия есть точное имя. Роутер не запрещает модели рассуждать и не портит её ответ — он перехватывает решение. Запрос летит к модели, которая сама способна сообразить, что с ним делать, но на подлёте его встречает классификатор и разводит по веткам раньше, чем модель успевает взглянуть. Перехват — это изъятие суждения «что делать», выполненное на входе: решение о задаче принимается до того, как задача дошла до того, кто мог бы вынести его лучше. Перехватывают не результат работы модели, а само право эту работу выбрать.

Образ подлёта буквален. Суждение «что делать» существует ровно один короткий миг — между тем, как запрос сформулирован, и тем, как за него взялись. В этот миг решение ещё открыто: его можно вынести под конкретные слова пользователя. Роутер вклинивается именно сюда, в зазор перед моделью, и закрывает решение прежде, чем оно дошло до того, кто прочитал бы запрос целиком. Оттого перехват так трудно заметить постфактум: к тому времени, когда система выдала ответ, развилки уже нет на схеме исполнения — она отработала на входе и растворилась в маршруте. Чтобы её увидеть, приходится смотреть не на то, что́ система ответила, а на то, кто решил, каким будет вопрос.

Разница между «модель выбрала способ» и «способ выбран за модель» кажется тонкой, пока система движется по размеченным рельсам. Она становится решающей на первом же запросе, который не лёг ни в одну категорию. Пользователь спрашивает разом про списание денег и про синхронизацию; или формулирует так, что классификатор уверенно относит запрос не к той ветке; или описывает случай, которого при проектировании веток просто не предвидели. Классификатор не сигнализирует о промахе — у него нет представления о том, что распознавание могло не удаться. Он выносит суждение с той же уверенностью, что и на идеальном примере из обучающей выборки, и передаёт модель в ветку, где та честно и качественно решает не ту задачу. Решение, застывшее в классификаторе, не пересматривается — потому что в этой архитектуре пересматривать его некому.

Особенно ясно перехват виден на том, что системы делают с запросом-сиротой — тем, что не опознан ни одной категорией. Здесь у архитектуры два обычных выхода, и оба подтверждают диагноз. Либо запрос уходит в ветку «прочее» — универсальный тупик, где пользователю уходит общий шаблон вежливого отказа или совет обратиться к человеку; либо порог классификатора настраивают так низко, что он всё равно приписывает сироту к ближайшей категории, лишь бы не сознаваться в незнании. В первом случае способную модель, которая могла бы разобраться в незнакомом запросе, до него попросту не допускают. Во втором — её запускают в заведомо неверную ветку. Ни то ни другое — не дефект реализации: и тупик «прочее», и заниженный порог — добросовестные ответы на проблему неизвестного входа. Просто оба отвечают на неё изъятием — тем, что суждение о незнакомом запросе выносит правило о пороге, а не интеллект, способный этот запрос прочитать.

У перехвата есть своя убедительность, и её стоит признать прямо. Классификатор намерения измерим: его точность можно посчитать, ошибки — увидеть на размеченной выборке, распределение намерений — вывести на панель и показать на ревью. Он даёт то, чего инженер справедливо хочет от продакшена: наблюдаемость, воспроизводимость, возможность сказать заказчику, что система делает ровно то и только то, что описано в спецификации веток. Роутер — не небрежность и не леность мысли; это инструмент, который делает поведение системы читаемым. Хорош он или плох — вердикт подождёт. Важно другое: что именно он замораживает. И цену стоит назвать без обиняков — читаемость покупается тем, что суждение о задаче переходит от того, кто видит конкретный запрос, к тому, кто заранее расчертил категории, ещё не зная будущих запросов.

В демонстрации этот слой невидим. На заготовленных вопросах роутер попадает в цель, ветки отрабатывают гладко, система выглядит понятливой — потому что демо и составляют из запросов, под которые ветки писались. Изъятие проступает только на живом трафике, на запросах, которых при разметке категорий никто не держал в голове; и проступает не как ошибка модели, а как молчаливое несовпадение между тем, что́ спросили, и тем, к какой полке это отнесли. Систему потом чинят, добавляя ветки, — но каждая новая ветка лишь передвигает границу неизвестного, а суждение о том, что за этой границей, обратно модели не отдаёт.

То, что на архитектурной схеме подписано «intent router» или «orchestration layer», в таксономии изъятия читается одной строкой: здесь заморожено суждение «какую способность применить». Обвязка, выглядевшая нейтральной инфраструктурой — просто «как система

устроена», — оказывается первым и, как правило, самым ранним местом, где у модели забрали право решать. Проявитель сработал на первом же слое: то, что казалось водопроводом, оказалось вынесенным суждением.

Перехватить, впрочем, можно не только выбор задачи. Даже когда модель допустили до самого решения, за ней остаётся ещё одно суждение — стоит ли, увидев результат, зайти на второй круг. Его тоже изымают, и тоже одной строкой конфигурации.

3.2. «Повторить ли»: одноходовые конвейеры

Модель ответила. Ответ, положим, слаб: извлечение подняло не тот документ, и на его основе получилась гладкая, уверенная и неверная реплика. Возникает решение — принять этот ответ или зайти ещё раз: иначе поискать, иначе сформулировать, перепроверить себя. Чьё это суждение — повторить ли?

В одноходовом конвейере его нет ни у кого. Точнее, оно вынесено заранее и вморожено в форму управляющего кода: проходов ровно один. Запрос входит, извлечение отрабатывает, модель генерирует ответ, ответ уходит пользователю — линейная труба без единого места, где кто-нибудь взглянул бы на промежуточный результат и решил, что стоит переиграть.

На схеме это выглядит обезоруживающе просто:

обработать (запрос):
контекст = извлечь (запрос)
ответ = модель (запрос, контекст) # ровно один проход
вернуть ответ

Ни ветвления, ни возврата назад, ни условия «если результат неудовлетворителен — повторить». Та же бедность выражается ещё короче, когда систему собирают на агентном каркасе и задают ему предел шагов: max_steps=1. max_steps — предел числа шагов, которые исполнителю позволено сделать в рамках одной задачи; значение 1 означает, что шаг будет ровно один. Одна строка конфигурации — и суждение «повторить ли» заморожено на этапе проектирования, задолго до того, как появится конкретный запрос, на котором второй заход оказался бы спасением.

Одноходовость не всегда носит имя. Явный max_steps=1 — лишь самый честный её вид, тот, что признаётся в конфигурации. Гораздо чаще второй заход отсутствует молча: конвейер собран как прямая последовательность вызовов, и место для возврата в нём просто не предусмотрели — не потому, что решили его запретить, а потому, что о нём не задумались. Такая система одноходова оттого, как она написана, а не оттого, что кто-то выбрал предел. И то и другое замораживает одно и то же суждение «повторить ли» — с той разницей, что явный предел хотя бы виден на ревью и может быть оспорен, а неявная линейность выглядит не решением, а просто «так устроен код».

Стоит заметить, когда именно принято это решение. Не тогда, когда пришёл трудный запрос, а задолго до него — на этапе проектирования, когда трубу рисовали под воображаемый средний случай. Проектировщик решал «одного прохода хватит» вслепую, не имея перед глазами тех запросов, на которых одного прохода не хватит; и это решение застыло, чтобы применяться ко всем будущим запросам одинаково — и к тем, для которых оно верно, и к тем, для которых губительно. Замороженное суждение не подстраивается под вход: в этом вся его природа. Оно вынесено один раз и для всех, кто ещё даже не обратился к системе.

В том, что здесь изъято, легко ошибиться — оно похоже на безобидную механику. Модель и так выдаёт один ответ на один вызов — в этом смысле любой отдельный вызов «одноходовый». Изъятие не в этом. Оно в том, что система не даёт модели посмотреть на плод собственной работы — на поднятый контекст, на черновик ответа, на результат вызванного инструмента — и, оценив его, распорядиться сделать иначе. Одноходовый конвейер отнимает

не второй проход как таковой, а суждение о том, нужен ли он: право сказать «этого мало, зайду ещё раз». Модель, способная заметить, что извлечение пустое или ответ противоречит вопросу, вынесла бы это суждение сама. Труба его не предусматривает.

Дефект виден там, где задача честно требует более одного захода. Помощник, сгенерировавший фрагмент кода, мог бы прогнать его через проверку, увидеть падение и исправить — но без петли он отдаёт первый вариант, падение и всё. Модель в принципе способна оценить промежуточный результат и решить, что стоит переиграть, — и именно этого решения её лишили. Промежуточный результат в одноходовой системе — не промежуточный: он окончателен по построению, потому что «между» в ней нет. Первый выход и есть последний.

Разыграем это на живом примере. Пользователь спрашивает docs-бота, как перенести данные при смене тарифа. Первый поиск поднимает страницу про сравнение тарифов — близко по словам, мимо по сути. Одноходовая система передаёт эту страницу модели, и та, добросовестно опираясь на единственный данный ей контекст, объясняет различия тарифов: гладко, по делу и не о том, о чём спросили. Модель тут не ошиблась в рассуждении — она безупречно ответила на вопрос, которого ей не задавали, потому что суждение «этот контекст не тот, надо искать иначе» ей выносить не позволили. Довольно было бы одного взгляда на несоответствие между вопросом и поднятым текстом — того самого взгляда, который отнимает единственный проход.

Тот же провал в другой области. Конвейер, извлекающий поля из счетов-фактур: распознаватель вернул сумму с потерянным разрядом, дата ушла в неверном формате. Модель, дай ей второй взгляд, сверила бы итог с построчной суммой и увидела расхождение — арифметика здесь элементарна. Но одноходовый контур отдаёт извлечённое как есть: проверить себя ему нечем и негде. Ошибка уходит дальше по системе с полной уверенностью, потому что суждение «сходится ли это, стоит ли перечитать» изъято на уровне формы конвейера, а не упирается в способность модели.

У одноходовости, как и у роутера, была своя причина, и причина основательная. Один проход — это предсказуемая стоимость, предсказуемая задержка и предсказуемое поведение: система, которой отмерен ровно один шаг, не уйдёт в долгий цикл, не удивит счётом и не зависнет. max_steps=1 — это ещё и страховка от модели, которая на слабом поколении могла закружиться, повторяя одно и то же бесполезное действие. Ограничение писалось как ответ на реальный страх неуправляемого цикла. Вопрос снова не в том, оправдан ли этот страх, — а в том, что вместе со страховкой от дурного цикла заморозили и суждение о хорошем: о том единственном втором заходе, который отделил бы верный ответ от уверенно неверного.

И, как всякое замороженное суждение, это не подаёт признаков жизни. Одноходовая система не жалуется, что ей не хватило проходов; она не знает, что их могло быть больше. Она возвращает первый ответ с той же ровной уверенностью, выверен он или случаен, потому что понятия «недостаточно, надо ещё раз» в её устройстве попросту нет. Инженер увидит на выходе связный текст и сочтёт систему работающей — а то, что она ни разу не переспросила саму себя, не оставит следа ни в логе, ни в ответе.

Из всех видов таксономии этот, пожалуй, самый незаметный — и оттого самый распространённый. Роутер хотя бы стоит на схеме отдельным блоком; одноходовость — это отсутствие, а отсутствие на диаграмме не рисуют: никто не проводит стрелку, чтобы показать, что её нет. Систему без петли не отличить с первого взгляда от системы с петлёй, которой второй заход просто ни разу не понадобился на демонстрации, — разница вскрывается лишь на запросе, где заход был нужен и не состоялся. Потому изъятие «повторить ли» так часто оказывается не выбранным, а доставшимся форме кода: одноходовость — состояние по умолчанию для всего, что собрано прямой трубой, и, чтобы дать модели второй заход, его надо предусмотреть, тогда как чтобы отнять — довольно ничего не делать.

Одноходовость — родовая черта самой распространённой конструкции прошлого поколения: «извлечь и прочитать». Достать контекст, дать модели, получить ответ — эта схема одноходова по замыслу, второй заход в ней не предусмотрен архитектурно. Флагманский случай инъекции живёт как раз внутри неё: суждение «что искать» изъято на входе, а суждение «повторить ли поиск» изъято тем, что вход ровно один. Два вида таксономии сходятся в одной трубе — и потому «извлечь и прочитать» так уверенно промахивается там, где ответ лежал в базе, но не в том её месте, куда система сходила свой единственный раз. Право повторить — это, в сущности, право признать «пока недостаточно»; отнять его значит объявить любой первый результат достаточным по определению.

Так одна строка — max_steps=1 или её неявный эквивалент в виде линейной трубы без ветвления — оборачивается вторым видом таксономии: изъятием суждения «повторить ли». На архитектурной схеме это даже не компонент, а свойство её формы: отсутствие стрелки, ведущей назад.

Перехвачен вход, заморожен второй заход — но остаётся ещё сам выход. Даже когда модели позволили выбрать задачу и, положим, повторить попытку, за ней остаётся последнее суждение: в какой форме высказать то, к чему она пришла. Изымают и его.

3.3. «Как ответить»: форс-вызовы, жёсткие схемы, шаблоны

Модель пришла к содержанию: она разобрала запрос, нашла нужное, у неё есть что сказать. Остаётся облечь это в форму — выбрать, ответить ли текстом или вызвать инструмент, какой развёрнутости требует случай, признать ли, что однозначного ответа нет. Чьё это суждение — как ответить?

Всё чаще — не модели. Форма ответа задаётся раньше, чем возникает содержание, которое в неё предстоит уложить. Ответ отливают в форму до того, как есть что отливать, — и делают это тремя привычными способами.

Странность этого порядка стоит проговорить. Обычно форма следует за содержанием: сначала есть что сказать, потом решают, как. Здесь наоборот — сосуд отлит прежде, чем налито, клетки готовы прежде, чем есть чему в них лечь. Модель приходит к выводу и обнаруживает, что способ его выразить уже выбран за неё — под ответ вообще, а не под этот. Иногда сосуд впору, и тогда изъятия не заметить. Но отлит он не под конкретный ответ, и на любом, что не лёг в заготовленную форму, зазор проступает: модель либо обрезает не поместившееся, либо натягивает своё суждение на чужую мерку.

Первый — форс-вызов. Форс-вызов — конфигурация, обязывающая модель на этом шаге непременно вызвать инструмент (или конкретный инструмент), не оставляя ей выбора, нужен ли вызов вообще. docs-бот, настроенный всегда вызывать поиск по базе, даже когда вопрос — простое «спасибо, помогло», или когда модель уже держит готовый ответ: инструмент срабатывает вхолостую, потому что решение «сейчас вызов уместен» вынесено за модель и приколочено к шагу. Или бот, обязанный на каждый запрос завести тикет: пользователь ещё уточняет вопрос, а тикет уже создан, потому что так велит форс-вызов. Модель могла бы рассудить, нужен ли инструмент здесь и сейчас; за неё решили, что нужен всегда.

Тонкость форс-вызова в том, что он изымает не столько сам вызов, сколько суждение о его уместности — а с ним и право не действовать. Способность вовремя не вызвать инструмент, промолчать, обойтись рассуждением — такое же суждение, как и способность вызвать, и оно исчезает первым. Модель, обязанная на каждом ходу что-то предпринимать, теряет возможность сказать «здесь делать ничего не надо» — а ведь именно этот ответ нередко и есть правильный.

Второй — переспецифицированная схема вывода. Схема вывода (structured output) — заданная заранее структура, в которую модель обязана уложить ответ: поля, их типы, допусти-

мые значения. Сама по себе структурность ответа — вещь полезная, и не о ней сейчас речь. Речь о схеме, затянутой так туго, что она диктует не только форму, но и содержание суждения. Схема, требующая уложить оценку тональности в три клетки — «позитивно», «негативно», «нейтрально», — вынуждает модель выбрать одну даже там, где честный ответ «смешанно» или «данных мало». Схема, где под причину отказа отведён перечисляемый список из пяти пунктов, заставляет подвести под один из пяти любой случай, включая шестой, которого в списке нет. Модель не лжёт — она подчиняется форме: заполняет поле ближайшим из разрешённых значений, потому что промолчать или ответить иначе схема ей не даёт. Суждение о том, какой формы требует этот конкретный ответ, вынесено проектировщиком схемы заранее и для всех ответов разом.

Опаснее всего, что искажение не остаётся внутри модели. Схема сортировки обращений, требующая проставить уровень срочности от одного до пяти, вынудит модель выставить число даже там, где по сообщению срочность не определить, — и это выдуманное число пойдёт дальше как факт. Ниже по системе никто уже не отличит цифру, взятую из существа дела, от цифры, поставленной лишь потому, что поле обязательно к заполнению. Изъятие формы порождает ложную определённость: там, где модель, будь ей позволено, честно сказала бы «не могу судить», схема заставляет её выбрать — и стереть собственное сомнение.

Третий — канонизированный ответ мимо модели. Здесь модель не формулирует вовсе: по совпадению условия система достаёт готовый текст и отдаёт его пользователю. docs-бот, распознав частый вопрос, возвращает заготовленный абзац из FAQ; модель в этот момент либо не вызвана, либо вызвана лишь затем, чтобы подтвердить совпадение. Ответ выражен — но не ею. Суждение «как сказать это данному пользователю в данном контексте» изъято целиком: сказано типовое, отобранное правилом.

И шаблон подводит именно там, где случай лишь притворяется типовым. Вопрос по ключевым словам совпал с частым, система выдала заготовленный абзац — а у пользователя ровно тот редкий поворот, из-за которого стандартный совет неверен. Модель, дай ей сформулировать, уловила бы оговорку в вопросе; шаблон её не слышит, потому что он вообще не слушает — он срабатывает по совпадению, а не по смыслу.

За тремя приёмами — один жест. Во всех случаях форма ответа зафиксирована снаружи и до содержания: вызов назначен прежде, чем стало ясно, нужен ли он; клетки схемы расчерчены прежде, чем известно, что в них ляжет; текст написан прежде, чем задан вопрос. Суждение о выражении заморожено — и, как прежде, заморожено вслепую, под воображаемый ответ, а не под тот, что возникнет. Различаются механики, но тип изъятия один: у модели забрали право решать, в какой форме высказать то, к чему она пришла.

В этом и особая цена изъятия выражения. Форма — то место, где обычно проступает неуверенность модели: в свободном ответе она может оговориться, развести случаи, сказать «скорее всего» или «точно не знаю». Тесная схема без клетки «неизвестно», форс-вызов без права воздержаться, шаблон, не знающий полутонов, — все они стирают именно эту способность сигналить о собственных пределах. Замороженная форма не просто сужает ответ; она заставляет модель звучать увереннее, чем та есть, и этим вводит в заблуждение всех, кто примет гладкость выхода за надёжность. Ответ, отлитый в форму до содержания, всегда выходит ровным — потому что форма не оставила места для шва, по которому было бы видно, что где-то модель сомневалась.

Как и другие изъятия, это редко выглядит поломкой. Ответ по схеме валиден, ответ по шаблону грамотен, форсированный вызов исправно отрабатывает — на выходе всё корректно по форме. Корректность и усыпляет: систему признают исправной, потому что она отдаёт ожидаемое по формату, и никто не спрашивает, во что обошлась эта опрятность. А обошлась она ровно в суждение о форме — в тысячах ответов, где модель сказала бы иначе, точнее или чест-

нее, если бы её об этом спросили. Опрятность выхода и изъятость выражения часто оказываются одним и тем же, увиденным с двух сторон.

У жёсткой формы, как и у прочих изъятий, причина честная. Структурированный вывод нужен, чтобы ответ модели можно было передать дальше по программе — в базу, в интерфейс, в смежный сервис, которые ждут ровно определённых полей и споткнутся о свободный текст. Форс-вызов гарантирует, что нужный шаг случится, а не будет пропущен на живом трафике. Шаблон обеспечивает единый выверенный тон и юридически проверенные формулировки там, где импровизация недопустима. Это не капризы: за каждым приёмом стоит реальное требование системы к предсказуемости выхода, и требование законное.

И потому важно: изъятие здесь только названо, приговор ему не вынесен. Из того, что схема или форс-вызов замораживают суждение о форме, не следует, что их надо изгнать: и то и другое способно служить иначе — оформлять суждение модели, а не занимать его место. Схема может задавать контракт на структуру, не диктуя содержания; вызов может быть гарантирован так, что модель всё равно остаётся автором того, что этот вызов несёт. Но это уже разговор о возврате — о том, как удержать пользу формы, не изымая решения. Здесь, в таксономии, фиксируется только сам факт: в этих трёх приёмах суждение о выражении вынесено за модель. Где проходит граница между формой, что служит суждению, и формой, что встаёт на его место, — вопрос отдельный, и до него дойдёт черёд.

Так три знакомых приёма контроля вывода — принудительный вызов, тесная схема, готовый шаблон — сходятся в третьем виде таксономии: изъятии суждения «как ответить». Проявитель показывает под словами «валидация вывода» и «единый формат» одно и то же замороженное решение о форме. Три вида таксономии уже собраны, и вместе они очерчивают всю траекторию работы системы: перехват берёт вход, одноходовость запирает петлю, жёсткая форма правит выходом — изъять суждение можно в любой её точке.

До сих пор речь шла о том, что система делает с выходом модели — с выбором задачи, с попытками, с формой ответа. Но изъять суждение можно и на другом конце, ещё до того, как модель начнёт думать: решив за неё, что ей держать перед глазами. Кто определяет, что модели помнить и что считать сейчас важным, — следующее место, где чужая рука наводит порядок.

3.4. «Что помнить»: внешнее управление памятью и суммаризация

Разговор с системой длится, задача обрастает историей, и рано или поздно всего накопленного становится больше, чем модель может держать перед собой разом. Что-то придётся оставить, что-то убрать, что-то сжать. Кто-то должен решить, что из прошлого всё ещё важно. Чьё это суждение — что помнить?

В большинстве систем — не модели. Решает харнесс — управляющая обвязка вокруг модели, распоряжающаяся её вызовами, инструментами и памятью. Когда история перестаёт помещаться в контекстное окно — объём текста, который модель способна держать перед собой в одном обращении, — харнесс наводит порядок: недавнее сохраняет дословно, давнее прогоняет через суммаризацию (сжатие накопленного в короткую сводку, заменяющую оригинал), а часть отбрасывает вовсе. Модель на следующем шаге получает уже прибранную память и работает с тем, что ей оставили, не зная, что осталось за кадром.

Чужая рука наводит порядок в чужой памяти. Хозяин памяти — модель, которой предстоит на неё опереться; распоряжается ею внешний код, который сам этой памятью пользоваться не будет. Рука решает, какие записи оставить на виду, какие свернуть в одну строку, какие смахнуть со стола, — и делает это по своим правилам, а не по нуждам того, кто потом будет вспоминать.

Внешнее распоряжение памятью принимает больше обличий, чем одна лишь подчистка диалога. Скользящее окно держит последние несколько сообщений и роняет всё, что вышло за край, — граница релевантности проведена по длине, а не по смыслу. Отдельный модуль памяти выуживает из архива прошлого «подходящие» записи по сходству эмбеддингов и подкладывает их модели — но что считать подходящим, решает механизм сходства, а не модель, которой этими записями предстоит воспользоваться. Обрезка результата инструмента оставляет первые строки ответа и отсекает хвост — на случай, если важное было в хвосте. Операции выглядят по-разному, но выносят одно и то же суждение: что из доступного заслуживает попасть модели перед глаза. Модуль памяти, отбирающий прошлое по сходству, — это, по сути, тот же перехват, только над памятью: решение о релевантности принято до модели и за неё.

Вот как это выглядит в поддержке. Пользователь в начале длинного диалога упомянул, что у него корпоративный тариф и особая настройка интеграции. Десяток реплик спустя история подросла, и харнесс свернул ранние сообщения в сводку: «пользователь спрашивает про интеграцию, испытывает трудности». Деталь про корпоративный тариф в сводку не попала — на вид несущественная частность. Ещё через несколько ходов пользователь задаёт вопрос, ответ на который зависит именно от типа тарифа. Модель отвечает общим случаем, не подозревая, что случай особый, — потому что того факта в её памяти уже нет. Она не забыла: у неё отняли. Суждение «эта деталь ещё пригодится» вынес за модель харнесс, свернувший историю по общему правилу, — и вынес неудачно, но модель об этом даже не знает.

Та же пропажа случается и вне поддержки. Агент, помогающий с кодом на большой базе, к середине задачи упирается в границу окна; харнесс сворачивает ранее прочитанные файлы в сводку, и в сводку не попадает сигнатура функции, которую агент видел двадцать шагов назад. Дальше он пишет вызов по памяти — и ошибается в аргументах, потому что точной сигнатуры у него уже нет. Он не поленился и не перепутал: решение, что эта деталь не стоит места в окне, принял за него харнесс.

Соль в том, что релевантность — это суждение, и суждение не механическое. Что из прошлого важно сейчас, зависит от того, какова текущая задача, а её видит именно модель. Харнесс же решает по признакам, к смыслу безразличным: по давности, по числу токенов, по месту в диалоге. Он свернёт то, что давно не упоминалось, даже если оно ключевое, и сохранит недавнее, даже если это шум. Модель, дай ей самой распоряжаться памятью, держалась бы не за свежее, а за важное. Внешнее управление памятью замещает суждение о важности расписанием уборки.

Уязвимость правила видна в одном частом узоре: сказанное однажды и в начале нередко правит всем, что идёт потом. Ограничение, названное пользователем в первой реплике; цель, поставленная в задании; условие, оговорённое на старте, — они произносятся один раз и больше не всплывают, потому что о них не переспрашивают, их держат в уме. Ровно их правило давности и роняет первыми: давно не звучало — свернуть. Механическая память путает «давно не упоминалось» с «неважно», хотя чаще верно обратное — важное потому и не повторяют, что считают установленным. Суждение по смыслу удержало бы такую опору; расписание уборки стирает её именно как старую.

И заметить пропажу модель не может. Ей не с чем сравнить: она видит ту память, что ей оставили, и принимает её за всю. Нет ни следа, ни зарубки «здесь было свёрнуто», по которой можно бы спросить, что именно ушло. Модель наследует отредактированное прошлое как полное и опирается на него с той же уверенностью, с какой опиралась бы на подлинное. Пробел в памяти не ощущается изнутри как пробел — он ощущается как отсутствие вопроса, который модель не задаёт, потому что не знает, что его стоило бы задать.

Это не то же, что забыть. Забывая, человек обычно чувствует границу забвения — знает, что деталь где-то была, и признаёт, что не помнит. Модель, чью историю свернули снаружи, лишена и этого чувства края: ей вручили опрятную, правдоподобную версию прошлого, в кото-

рой нет прорех, за которые можно зацепиться. Свёрнутое оставляет не пустоту, а гладкость, а гладкость подозрений не вызывает. Сама суммаризация в этом смысле коварнее простого отбрасывания: выброшенное хотя бы отсутствует честно, а сводка присутствует — она выглядит как содержание, оставаясь при этом внешним решением о том, что в этом содержании было главным. Модель принимает сводку за то, что было, и строит следующий шаг на чужом конспекте как на собственной памяти.

Причина, по которой памятью распоряжаются снаружи, снова уважительная. Контекстное окно не бесконечно; на длинной истории что-то сжимать приходится просто чтобы всё поместилось. Прогонять давнее через сводку, а недавнее держать целиком — разумная политика по умолчанию, и во времена тесных окон она была почти безальтернативной. Речь опять не о том, плоха эта политика или хороша, а о том, чьё суждение она собой замещает: механическое правило встало на место решения о важности, которое могло бы приниматься по смыслу.

Из всех изъятий это, пожалуй, легче всего пропустить, потому что оно рядится в чистую техничность. «Управление контекстом», «стратегия памяти» звучат как санитарная процедура, как забота о ресурсе, а не как решение о содержании мысли. Оттого его редко ставят в один ряд с роутером или схемой — а напрасно: свернуть воспоминание не менее весомо, чем перехватить задачу или сковать форму. И у этого вида есть более скрытая, более коварная разновидность — когда сжатие происходит не в служебной памяти между ходами, а на самом входе, до модели, и та получает уже чужую сводку вместо исходного материала, не подозревая, что видит пересказ. Эта разновидность заслуживает отдельного, подробного разбора; здесь довольно зафиксировать сам тип: суждение о релевантности вынесено за модель.

Стоит отметить и то, как этот вид складывается с прочими. Система, уже перехватившая задачу и стеснившая форму ответа, тем же движением распоряжается и памятью модели — одна и та же внешняя логика правит входом, петлёй, выходом и тем, что между ними хранится.

Так незаметная служебная операция — подчистка контекста, рутинная сводка — оказывается четвёртым видом таксономии: изъятием суждения «что помнить» — ещё одним замороженным решением о том, что из прошлого имеет вес.

Остаётся последний тип, и самый откровенный из всех. До сих пор изымали то, что скрыто внутри работы модели, — выбор задачи, второй заход, форму, память. Но есть суждение, вынесенное наружу демонстративно, у всех на виду: право самого действия. И на нём впервые обнаружится, что изъятие бывает не только по недосмотру.

3.5. «Действовать ли»: гейты автономии

Модель разобралась в задаче, выбрала, что сделать, и готова это сделать — отправить письмо, оформить возврат средств, удалить запись, изменить настройку. Между решением и его исполнением встаёт последний вопрос: пускать ли модель к самому действию. Чьё это суждение — действовать ли?

Всё чаще его держит у себя человек или жёсткое правило. Между намерением модели и его воплощением ставят гейт автономии — обязательную остановку, на которой действие должно быть подтверждено извне, прежде чем оно совершится. Самая частая его форма — human-in-the-loop: человек в контуре, чьё одобрение требуется, чтобы шаг состоялся. docs-бот может составить ответ и предложить оформить возврат, но само оформление уйдёт в очередь на подтверждение оператору; агент может подготовить письмо, но не отправит его без нажатия «отправить»; система может предложить удаление, но исполнит его лишь после явного «да».

Гейт принимает разные формы, но все они — вариации одной остановки. Иногда это очередь на подтверждение, где каждое предложенное действие ждёт живого одобрения. Иногда — режим «только предложить»: модель формирует список того, что сделала бы, а исполняет его человек. Иногда — песочница, где модели позволено что угодно, но лишь понарошку, без

выхода наружу. Иногда — белый список: этих действий не касайся без санкции, а вот эти совершай свободно. От самого глухого гейта, где подтверждают всё, до самого узкого, где живой рукой держат одну-единственную необратимую операцию, тянется целый разброс — и место, в которое систему поставили на этой линии, и есть то решение о доверии, что здесь вынесено за модель.

Перед каждым действием опущен шлагбаум. Модель доезжает до него со всем решённым — куда ехать и зачем, — и останавливается, ожидая, пока поднимут. Поднимает не она.

Разыграем оба края на поддержке. docs-бот разбирает жалобу, признаёт её обоснованной, готовит возврат средств и кладёт его в очередь оператору; оператор, глянув, одобряет — гейт сработал ровно там, где деньги покидают счёт, и никого не стеснил. А теперь тот же бот с гейтом на всём подряд: чтобы заглянуть в документацию, он спрашивает разрешения; чтобы уточнить у пользователя деталь — спрашивает разрешения; чтобы предложить черновик, которого ещё никто не видел, — снова спрашивает. Первый гейт бережёт; второй превращает способную систему в того, кто ежеминутно дёргает человека за рукав по пустякам. Механика подтверждения одна и та же; смысл — противоположный.

Тот же расклад повторяется всюду, где система дотягивается до мира. Агент, управляющий инфраструктурой, волен сколько угодно рассуждать о том, какой сервер перезапустить, — но сам перезапуск в бою держат за гейтом, и это разумно. Тот же агент, вынужденный спрашивать разрешения, чтобы всего лишь прочитать лог, упирается в изъятие без повода. Финансовый помощник может собрать и обосновать сделку, но исполнение уводят за подтверждение — из-за той же необратимости, из-за которой оформление возврата не отдают боту насовсем.

Этот вид стоит в таксономии особняком, и различие важно. Прежние четыре изымали суждение внутри мышления модели — что делать, повторять ли, как выразить, что помнить. Гейт мышления не трогает: модель по-прежнему сама решает, что нужно сделать, и решает целиком. Изъято ровно одно — право перейти от решения к делу. Граница проведена не внутри рассуждения, а на его выходе, там, где мысль становится поступком с последствиями в мире.

И даже здесь важно, что именно перекрывает шлагбаум. Хороший гейт держит спусковой крючок, оставляя модели всё суждение: реши сама, что и почему, — а совершит поступок пусть человек. Гейт похуже перекрывает не поступок, а само решение: план не исполнить, пока его не утвердят, — и тогда к изъятию права действовать тихо примешивается изъятие права судить, то самое, что мы разбирали в первых четырёх видах. Санкционировать поступок и санкционировать мысль — разные вещи, и смешивать их значит под видом одного гейта ставить два.

И вот здесь впервые за главу стоит остановиться и качнуть маятник в другую сторону. До сих пор всякое изъятие мы разбирали как то, что модель могла бы вынести лучше, — как заморозку, доставшуюся системе по инерции. С гейтом всё иначе. Действие в мире необратимо там, где мысль обратима: неверный вывод можно переписать, а переведённые деньги, отправленное письмо, удалённую запись — уже нет. Остановка перед необратимым — не обязательно небрежность и не обязательно долг. Очень часто это трезвый, обдуманный выбор: отдать модели всё суждение о том, что делать, но оставить за человеком право санкционировать сам поступок. Изъятие права действовать сплошь и рядом делается нарочно — и по хорошей причине.

У этого есть причина глубже, и она объясняет, почему именно здесь маятник качнулся. За первыми четырьмя изъятиями чаще всего стояла ставка против умения модели: её не пускали выбирать, повторять, формулировать, помнить из страха, что не справится, — а такая ставка тает по мере того, как модель крепнет. Гейт стоит на другом основании. Он поставлен не против умения, а против последствий, а последствия действия не уменьшаются оттого, что модель поумнела: перевод остаётся необратимым, письмо — отправленным, запись — удалённой, как бы хорошо система ни рассуждала. Оттого повод остановиться перед необратимым не

выветривается со сменой поколений так, как выветриваются поводы для прочих изъятий, — и оттого право действовать возвращают не так, как остальные.

Что не отменяет вопроса, как именно гейт поставлен. Одно дело — шлагбаум перед необратимым и дорогим: перед платежом, перед рассылкой тысяче адресатов, перед удалением данных. Другое — шлагбаум перед всем подряд, включая чтение справки и черновик, который, кроме модели, ещё никто не видел. Первый — граница, поставленная с умыслом, вокруг того единственного места, где цена ошибки высока. Второй — тот же старый рефлекс тотального контроля, просто переехавший на действия: подтверждать всё, на всякий случай, не различая необратимого и пустякового. Тогда гейт из осмысленной остановки вырождается обратно в изъятие по привычке — модель, способная сама отличить рядовое от опасного, снова просит разрешения на каждый шаг. Разница между этими двумя гейтами не в механике, механика одна; она в том, вынесено ли решение «здесь нужна санкция» осознанно или проштамповано на всём скопом.

У тотального гейта есть цена тоньше, чем раздражение. Когда подтверждать приходится всё, подтверждение становится рефлексом. Человек, которому за день подсовывают три сотни одобрений, из которых почти все — рутина, перестаёт вчитываться и штампует «да» не глядя, — и тогда то единственное опасное действие, ради которого гейт и ставили, проскакивает вместе со всеми. Остановка, размазанная по всему подряд, не ловит редкий рискованный шаг, а топит его в потоке безобидных. Гейт, поставленный везде, охраняет хуже гейта, поставленного в одном верном месте; изъятие, сделанное на всякий случай, подрывает ту самую защиту, ради которой его вводили.

Потому право действовать и занимает в карте суждений особое место. Прочим четырём видам предстоит прямой возврат: выбор задачи, второй заход, форму, память модели можно отдать. С этим возврат условен. Его не отдают безоглядно; его разбирают как то самое изъятие, которое бывает и правильным, — и вопрос не в том, вернуть или нет, а в том, где санкция человека оправдана последствиями, а где она лишь тень былой недоверчивости к слабым моделям. Этот разбор впереди — там, где речь пойдёт о риске и о том, каким сценариям можно доверять, а каким нет. Здесь довольно поставить сам тип на его место в перечне и отметить его особость: это первое суждение, об изъятии которого нельзя сказать «верните — и дело с концом».

Так пятый и самый заметный вид таксономии — гейт автономии, шлагбаум перед действием — приносит с собой то, чего не было в первых четырёх: законный повод для изъятия. Проявитель показывает под «подтверждением действий» замороженное суждение «действовать ли» — но впервые оговаривается, что замороженным оно бывает и по уму, а не только по инерции.

Пять видов собраны. У четырёх изъятие выглядело долгом, у пятого — порой осознанным выбором, и это расхождение требует, чтобы его назвали прямо. Все пять, при всей несхожести механик, держатся на чём-то одном; и различие между изъятием по уму и изъятием по привычке, только что мелькнувшее на гейте, просит своей формулы.

3.6. Общий знаменатель: замороженное чужое суждение

Пять видов, пять механик, пять разных мест в системе — и всё же за ними одно. Роутер, перехвативший задачу; конвейер, запретивший второй заход; схема, отлившая форму; харнесс, свернувший память; гейт, придержавший действие, — сколь ни различны их устройства, все они делают ровно одно и то же. Каждый берёт суждение, которое модель могла бы вынести сама, выносит его вместо неё и замораживает результат снаружи. Это и есть формула, к которой сводится вся таксономия: изъятие агентности — это вынесенное за модель и застывшее чужое суждение.

Стоит назвать механику точным словом. Замороженное суждение — решение, вынесенное один раз, на этапе проектирования, и застывшее в коде: оно принято до того, как появился конкретный запрос, и применяется ко всем запросам одинаково, не пересматриваясь. Заморозка — не про то, что решение плохое; про то, что оно неподвижное. Классификатор выбирает ветку сегодня по правилу, записанному вчера; одноходовость отвергает второй заход, которого ещё не было; схема диктует форму ответу, который ещё не возник. Во всех случаях живое суждение, чуткое к конкретному входу, заменено мёртвым — вынесенным заранее и навсегда.

Суть заморозки — в том, когда принято решение. У живого суждения есть свойство, отличающее его от замороженного: оно выносится в тот момент, когда виден конкретный вход. Модель решает на инференсе — глядя на этот запрос, этот контекст, этот промежуточный результат; её суждение рождается вместе с задачей и подстроено под неё. Замороженное решение вынесено раньше — на этапе проектирования, когда конкретной задачи ещё не было, а был лишь воображаемый средний случай. Изъятие, если смотреть в корень, и есть этот сдвиг момента: суждение переносят с исполнения на проектирование, от того, кто видит вход, к тому, кто входа не видел. Всё прочее — механика: роутер, схема, гейт — лишь разные способы закрепить сдвиг в коде. Оттого замороженное суждение и не ошибается в привычном смысле — ему нечем ошибиться, оно вынесено до того, как появилось на чём ошибаться; оно просто не пересматривается, когда вход оказывается не тем, под который его выносили.

В этом свете флагманский случай книги встаёт на своё место — как один из. Инъекция, при которой модель получает подложенный контекст и отвечает по нему, не выбирая, что искать, — это заморозка суждения «что и где искать», ровно того же рода, что и все прочие. Она заметнее остальных, потому что стоит на входе и бьёт по самому очевидному — по фактам в ответе; оттого с неё удобно начинать. Но особого статуса у неё нет. RAG — не болезнь и не исключение; это первый узнанный симптом паттерна, который, узнав, видишь повсюду. Инъекционное извлечение — частный случай изъятия, а не отдельная беда.

Конец ознакомительного фрагмента.