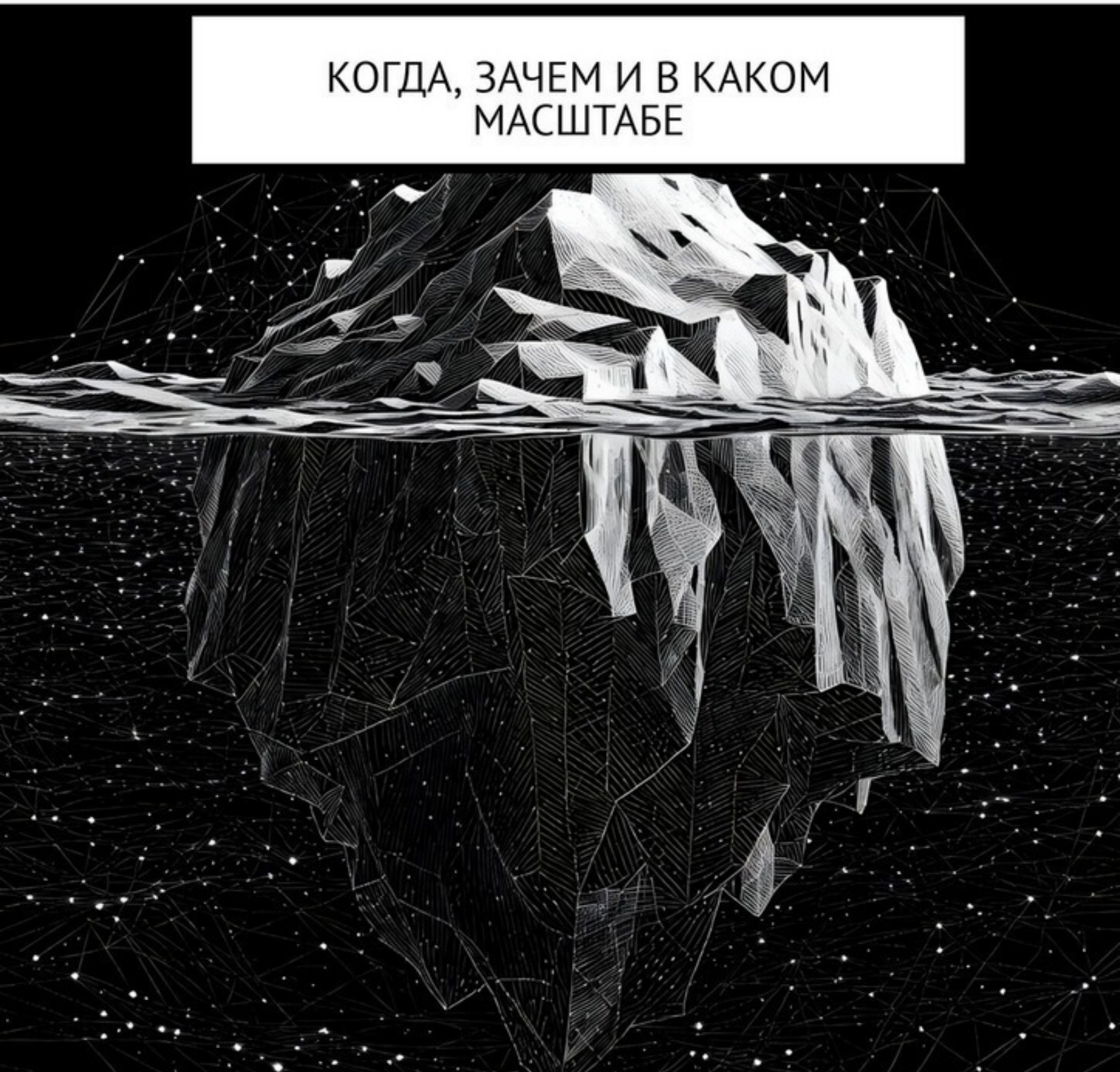
СЕРГЕЙ КИРНИЦКИЙ

Собственная языковая модель

КОГДА, ЗАЧЕМ И В КАКОМ МАСШТАБЕ



Сергей Кирницкий

Собственная языковая модель. Когда, зачем и в каком масштабе

«Издательские решения»

Кирницкий С.

Собственная языковая модель. Когда, зачем и в каком масштабе / С. Кирницкий — «Издательские решения»,

Большие языковые модели за несколько лет прошли путь от лабораторного любопытства до рабочего инструмента в руках миллионов. Вслед за этим почти перед каждой крупной компанией встал вопрос: нужна ли ей собственная такая модель — и если да, то в каком масштабе. Книга разбирает этот вопрос от начала: что значит «своя модель» на самом деле, из чего она собрана, сколько лет и людей требует, почему большинству компаний она не нужна, а некоторым всё же нужна и в каком виде.

Содержание

Собственная языковая модель
Когда, зачем и в каком масштабе

Сергей Кирницкий

Оформление обложки Created with Grok

ISBN 978-5-0069-9184-2
Создано в интеллектуальной издательской системе Ridero

Часть I. Собственная LLM как стратегическое решение

Глава 1. Актуальность темы и постановка вопроса

1.1. Почему вопрос «своя LLM» встал сейчас

Последние несколько лет в повестке советов директоров появился пункт, которого там прежде не было. Он звучит по-разному — «ИИ-стратегия», «собственная модель», «цифровой суверенитет компании», — но за всеми этими формулировками стоит один и тот же вопрос: нужна ли компании собственная большая языковая модель (**LLM** — large language model, модель, способная работать с текстом на уровне универсального инструмента). Вопрос новый, и многие советы обнаруживают, что у них нет готового языка, чтобы на него ответить — как нет ещё и внутреннего консенсуса о том, кто именно за него отвечает.

Волна, которая вынесла этот вопрос в залы заседаний, собиралась несколько лет, но видимой стала быстро. Сначала ChatGPT — продукт, выведший большие языковые модели из исследовательских лабораторий в публичное пространство. За ним — практическое внедрение подобных инструментов в операционные процессы: обработка обращений, разметка документов, извлечение фактов из писем, черновики отчётов и служебных записок. То, что ещё недавно описывалось как эксперимент, превратилось в рабочий инструмент, про который у каждого крупного подразделения появился либо свой ответ, либо честное «пока нет, но думаем». В этот момент технология перестала быть темой для научной конференции и стала темой для оперативного совещания. Возникла новая базовая грамотность руководителя — умение оценивать, где и как модели этого класса могут встроиться в работу компании.

Параллельно с публичной волной шла другая, менее заметная. Открытые модели, которые пять лет назад существовали почти исключительно в виде исследовательских прототипов, достигли уровня, на котором их всерьёз можно использовать в продукте. Публикация Llama и технический отчёт DeepSeek V3 — два из множества сигналов, что экосистема созрела: базовая модель больше не является секретом крупнейших лабораторий, её можно взять как отправную точку, использовать напрямую или дообучать на собственных данных. Тогда мысль «построить свою» звучала фантастически; сейчас она для узкого круга компаний звучит обоснованно. Зрелость экосистемы означает, что у вопроса появилась техническая осуществимость — и именно отсюда у советов директоров возникла задача проверить, приложима ли она к ним.

К этим двум волнам добавляется третья — внешнее давление. Его редко называют так прямо, но оно формирует значительную часть повестки. Конкуренты объявляют собственные ИИ-стратегии; и даже если эти стратегии пока ничего не значат по существу, их само наличие превращается в аргумент на совете: у них есть — у нас нет. Деловые СМИ задают прямой вопрос об ИИ-стратегии, и отсутствие внятного ответа становится репутационным риском. Государственные регуляторы постепенно формируют собственные требования: ЦБ выпускает методические указания по применению моделей в финансовом секторе, Минздрав начинает обсуждать границы использования моделей в клинической практике, тема персональных данных по 152-ФЗ получает новое измерение из-за того, что модели обучаются на больших корпусах и плохо забывают попавшее в них. Для объектов критической информационной инфраструктуры (**КИИ** — категория объектов, отказ которых создаёт системные риски для страны) сам факт отправки данных во внешний API оказывается вопросом не удобства, а режима. В

этой внешней рамке вопрос о собственной модели встаёт даже перед теми компаниями, которые изнутри его не задают; он приходит снаружи и требует ответа — независимо от того, готова компания его сформулировать или нет.

Почему именно большие языковые модели оказались в центре «ИИ-стратегии» в текущем цикле — вопрос, заслуживающий отдельного замечания. Предыдущие волны искусственного интеллекта — классические алгоритмы машинного обучения, компьютерное зрение, системы рекомендаций — встраивались в конкретные процессы, решали точечные задачи и оставались техническим выбором внутри соответствующего подразделения. Рекомендательная система работала в фоне, модель кредитного скоринга — тем более; пользователь о них не знал и не должен был знать. Большие языковые модели ведут себя иначе. Одна и та же модель способна обрабатывать множество разных задач: отвечать на вопросы сотрудников, готовить проекты документов, анализировать обращения клиентов, помогать в разработке программного кода. Она встраивается в ядро продуктов — и у пользователя появляется прямой, видимый контакт с ней. То, что прежде было внутренней деталью архитектуры, получило внешнее лицо и голос. Актив, у которого такое лицо, неизбежно становится предметом разговора на уровне совета: владеть или арендовать — это вопрос стратегической собственности, а не технической реализации.

Отсюда и новизна самой постановки. Раньше, в предыдущие волны, решение о конкретной модели почти всегда делегировалось ML-команде или отделу аналитики: специалисты сами выбирали алгоритм, закупали лицензию, договаривались с вендором. Доходить до совета директоров такие решения практически не имели поводов. Современные LLM сдвинули эту границу. Сама модель стала точкой контроля над целым классом продуктов компании, её поведением в коммуникации, её доступом к внутренним данным. Решение о том, кто этой точкой владеет — сама компания или внешний поставщик, — перестало быть техническим. Оно стало стратегическим, и делегировать его команде ML больше нельзя, как нельзя делегировать решение о выходе на новый рынок или о смене ключевого поставщика. Именно поэтому вопрос оказался там, где раньше его не было, — в повестке высшего руководства. И именно поэтому от него нельзя отмолчаться: отказ от ответа здесь — тоже ответ, только принятый по умолчанию.

1.2. Ловушка «хочу свою модель»

У нового вопроса на повестке обнаруживается и своя типовая ошибка. Её не всегда называют вслух, но она регулярно видна в том, как разговор строится: желание иметь собственную LLM проявляется раньше, чем проведён серьёзный анализ того, зачем она компании и что для неё потребуется. Решение оказывается эмоциональным — статусным, — а не аналитическим. Его можно узнать по характерным признакам: формулировка «нам нужна своя модель» появляется в обсуждении до того, как кто-либо спросил, какую задачу она должна решать; в качестве аргумента звучит не потребность, а ссылка на действия крупного игрока — «у них есть, значит, и у нас должна быть»; сроки и бюджеты называются раньше, чем описана предметная область, в которой модель будет работать. Эта ошибка почти неизбежна в атмосфере внешнего давления: если вопрос о собственной LLM приходит снаружи, самым быстрым способом закрыть его кажется ответ «да, занимаемся», а дальше — уже не до анализа.

В индустрии это называют по-разному — «статусный ИИ», «витринная модель», «LLM для годового отчёта», — но суть одна: собственная языковая модель воспринимается не как решение задачи, а как символ технологической зрелости компании. Проблема не в том, что символ плох сам по себе, а в том, что символическая мотивация не проходит через ту же проверку, что и функциональная. Проект, в основании которого лежит символический мотив, особенно уязвим к первым же трудностям, потому что у команды нет чёткого внутреннего критерия, ради чего их терпеть.

Удобно посмотреть на ситуацию через аналогию с собственным заводом. В большинстве отраслей вопрос «строить свой завод или покупать у поставщика» имеет давний и ясный ответ: покупать. Завод строят там, где у компании есть уникальное преимущество в производстве, где рынок не даёт нужного качества, объёма или скорости поставки, где вертикальная интеграция — часть осознанной стратегии, а не способ заполнить графу. В остальных случаях собственное производство — это дорогое, долгое и рискованное занятие, уводящее компанию от её настоящего дела. История знает множество случаев, когда успешные потребительские бренды попадали в капкан собственного производства и выходили из него с потерями. С большими языковыми моделями логика та же, только масштаб риска выше. «Дорого» здесь означает не просто крупную сумму, а сумму, удваивающуюся или утраивающуюся по мере того, как выясняются неочевидные при старте детали: сбор и очистка данных, подготовка инфраструктуры, цикл выравнивания поведения модели под требования компании, сопровождение после запуска. «Долго» — значит не «дольше, чем хотелось бы», а значительно дольше, чем закладывается в первоначальный план, и с обязательным продолжением после запуска: модель живёт недолго, и поддерживать её свежесть придётся непрерывно. В подавляющем большинстве случаев правильный ответ — пользоваться внешней моделью через интерфейс доступа, настраивать её поведение через вспомогательные механизмы, дообучать лёгкими методами на собственных данных. «Своё производство» в применении к LLM имеет смысл там, где у компании есть уникальное преимущество в данных, особый продукт, нетипичные требования к режиму — и где всё это перевешивает гигантские накладные расходы на самостоятельную разработку. Такая конфигурация встречается гораздо реже, чем кажется изнутри.

Из аналогии следует и то, что означает «оценить, что действительно требуется». Если импульс звучит как «мы хотим свою модель», то зрелая реакция — разложить этот импульс на проверяемые элементы. Какие данные у компании есть, и что в них такого, чего нет в открытом интернете? Есть ли у неё команда, способная не просто обучить модель, а поддерживать её жизненный цикл годами? Какой временной горизонт компания готова выдерживать — полгода, два года, пять? Какая у неё оценка риска, что через год результат окажется хуже, чем внешняя модель, доступ к которой можно купить по подписке? Готов ли совет директоров к тому, что проект не закончится запуском, а перейдёт в режим постоянного обновления? Каждый из этих вопросов имеет свою предметную плоть. Собранные вместе, они заменяют общее «хочется» на конкретную картину, в которой «строить своё» либо обретает основание, либо распадается на составляющие и исчезает.

Именно в этом задача книги — не отговорить руководителя и не подтолкнуть его, а помочь отличить обоснованное стратегическое решение от импульсивного. Ни один из этих двух путей не универсален: для какой-то компании «своя LLM» — действительно верный ответ, а для другой — дорогостоящая демонстрация, которая закончится тихим сворачиванием проекта через полтора года. Книга не берётся назначить, кто в какой группе, а даёт читателю собственный инструмент распознавания. В итоге решение остаётся за читателем; меняется лишь то, на чём это решение основывается. Разница между импульсом и стратегией — не в смелости, а в наличии картины поля перед глазами.

1.3. На какой вопрос отвечает книга

В этой книге нет универсального ответа «делайте X». Это не уклонение автора от ответственности и не уступка разнообразию мнений. Просто универсального ответа на вопрос «нужна ли нам своя LLM» объективно не существует: ситуации компаний настолько различны, что любой единый совет обязательно кого-то обманет — либо читателя, которому он не подходит, либо самого себя, если будет признан верным независимо от контекста. Книга, кото-

рая утверждает обратное, полезна разве что в качестве мотивационного текста — но не как инструмент управленческого решения.

Пространство возможных ответов, впрочем, не бесконечно. Оно раскладывается на пять устойчивых рекомендаций, и именно между ними проходит реальный выбор. Первая рекомендация — «ничего не делайте, используйте RAG или внешние API»: работать с готовыми моделями через интерфейсы доступа, настраивать промпты, выстраивать внешнюю память для точного извлечения знаний, применять лёгкие методы адаптации. Вторая — **Малая** (сценарий S) — простая специализированная модель, собственная, но небольшая, обученная под одну узкую задачу. Третья — **Адаптированная** (сценарий A) — продолженное предобучение на открытой базе, когда за точку отсчёта берётся модель класса Llama или DeepSeek и дообучается на собственных данных компании. Четвёртая — **Собственная** (сценарий B) — предобучение средней модели с нуля, проект уровня серьёзной исследовательской лаборатории. Пятая — **Фронтир** (сценарий C) — лидирующий класс моделей, сопоставимых с GPT-4/5, Claude, Gemini, — уровень, на котором в мире работает лишь очень узкий круг организаций. Эти имена и короткие пояснения — всё, что нужно знать о модели пяти рекомендаций сейчас. Подробное содержание каждой, различия в масштабе, требованиях и рисках раскрываются в Главе 6; способ определить, какая из пяти — ваша, — в Главе 10.

Такая архитектура — не случайная. Методика выбора, которую книга доводит до практического инструмента в Главе 10, состоит из набора диагностических вопросов и оценочной карты. Ответить на эти вопросы формально, за пятнадцать минут, можно и без остальной книги — получится анкета, которую можно сдать начальнику. Проблема в том, что за каждым из вопросов стоит предметная реальность, которую необходимо увидеть, чтобы ответ имел смысл. Нельзя всерьёз оценить, достаточно ли у компании данных, не понимая, что именно считается «данными» для обучения модели и чем они отличаются от того, что компания привыкла называть своим информационным активом. Нельзя оценить, готова ли команда, не представляя, какие роли вообще в такой команде должны быть, какой у них рынок и насколько опасен уход одного ключевого специалиста. Нельзя оценить ресурсы, не зная, что уровни масштаба принципиально разные: между Малой моделью и Фронтиром — не разница в цене подписки, а смена индустриальной категории. Нельзя оценить риски, не видя, как именно устроены типовые неудачи подобных проектов, и чем в них оборачивается каждая сэкономленная статья бюджета. Поэтому книга построена последовательно: в Главе 2 очерчивается предмет — что именно мы называем «своей LLM» и чем она отличается от соседних вариантов; в Главах 3 и 4 раскрывается её устройство и процесс создания; в Главах 5 и 6 — основания для разработки и четыре сценария масштаба; в Главах 7 и 8 — данные и команда; в Главе 9 — риски, на которые стоит смотреть трезво. К Главе 10 читатель приходит с готовой картиной поля — и только тогда оценочная карта перестаёт быть анкетой и становится инструментом.

Отсюда и позиция автора. Он не принимает решение за читателя и не ведёт его к заранее заготовленному выводу. Роль, которую автор на себя берёт, — показать устройство поля: где проходит экономика, где риски, где границы возможного. Всё, что читатель должен вынести из книги, — ясное видение того, из каких материалов и при каких условиях складывается каждый из пяти ответов, и чем они отличаются друг от друга. Окончательное решение — всегда за читателем, и оно будет лучшим, если принимается с полным пониманием предмета. Книга рассчитана не на инструкцию, которую можно выполнять по пунктам, а на карту, по которой человек сам прокладывает маршрут под собственные обстоятельства. Пять возможных ответов — это пять островов на этой карте; и каждый остров ещё предстоит сделать узнаваемым, чтобы ошибиться, определяя свой, стало существенно труднее.

Остаётся сказать ещё об одном свойстве этого разговора. На вопрос «нужна ли нам своя LLM» серьёзно отвечает не тот, кто отвечает быстро, а тот, кто отвечает после того, как увидел предмет целиком. Темп чтения этой книги — не темп сводки для совещания; скорее темп

разговора с опытным коллегой, которому не нужно продавать свою позицию и который не спешит. Если в конце такого разговора руководитель приходит к ответу «нам это не нужно», это не провал книги, а её успех. Если приходит к ответу «нам нужна Малая, и вот почему» — тоже успех. Книга считает себя полезной в обоих случаях, при условии, что ответ получен не из импульса, а из картины. Всё остальное — частные варианты одной и той же задачи: увидеть поле и принять решение, глядя на него, а не от него отвернувшись.

Глава 2. Предмет, границы и альтернативы

Слова «своя LLM» звучат понятно ровно до того момента, когда за них приходит счёт. На совещании фраза объединяет людей: все кивают, будто речь идёт об одном и том же. В договоре подряда, в техническом задании, в спецификации на закупку видимая общность рассыпается. Одни имеют в виду закрытый контур с чужой моделью внутри, другие — дообучение на своих данных, третьи — модель, созданную с нуля. Эта глава делает одно: отделяет «свою LLM» от всего, что на неё похоже, и показывает четыре готовых пути, каждый из которых в большинстве случаев правильнее собственного проекта.

2.1. Что мы называем «своей LLM»

Есть короткая формула, которая снимает большую часть путаницы: своя LLM — это владение весами. Не контроль над инфраструктурой, не возможность поднять модель в своём контуре, не монопольный доступ к API — именно веса, те самые числовые параметры, которые и делают модель моделью. Владение здесь — в юридическом смысле: компания может эти веса модифицировать как угодно, дообучать, переобучать, сжимать, склеивать с другими, лицензировать третьим сторонам, встраивать в свои продукты без разрешения вендора. Из этого следует и обратное: все обязанности по модели тоже остаются на компании. Ответственность за её поведение, правовые риски по данным, расходы на поддержание — всё это часть права собственности, а не приложение к нему. Владение — одновременно свобода и обуза. Именно это сочетание отличает свой актив от аренды, даже самой комфортной.

Хороший способ думать об этом — дом. Можно построить его с нуля, можно перестроить из купленного вчерне, но в обоих случаях документы оформлены на тебя. Арендованная квартира — пусть с отдельным входом, своим замком и разрешением переклеить обои — домом в этом смысле не становится. Разница незаметна день в день и становится очевидной в тот момент, когда арендодатель меняет правила.

Владение весами достигается одним из двух путей. Первый — **предобучение с нуля** (pretraining): модель начинается с набора случайных чисел и проходит через гигантский корпус текстов, постепенно превращаясь в то, что мы называем языковой моделью. Это самый тяжёлый режим. Компания начинает с пустоты и создаёт модель целиком. Характерен этот путь для сценариев B (Собственная) и C (Фронтир), подробно разворачиваемых в Главе 6. Порядки величин, которые здесь появляются, — триллионы токенов обучающих данных, сотни или тысячи GPU, многомесячные прогоны обучения; детализация этих цифр — предмет Глав 6—7. Для нас пока важно одно: предобучение с нуля — это режим создания модели из ничего, и он даёт полное, ничем не обременённое право собственности на результат.

Второй путь — **продолженное предобучение** (continued pretraining): берётся уже существующая модель — открытая, как Llama или DeepSeek, или собственная предыдущая версия — и дообучается на новых данных. Результат этого процесса — новые веса, отличающиеся от исходных, и именно они становятся собственностью компании. Ресурсов требуется кратно меньше, чем для предобучения с нуля — в разы меньше данных, в разы меньше вычислений, — но статус актива получается тот же: новые веса полностью принадлежат компании. Этот режим характерен для сценария A (Адаптированная) и отчасти для S (Малая). И именно поэтому A — полноценная «своя LLM» наравне с B и C: их отличает масштаб и дороговизна, а не статус актива на выходе. Разница между арендой и собственностью здесь важнее, чем разница между большим домом и малым.

Осталось отделить своё от похожего. Частый случай терминологической путаницы — приватное развёртывание чужой модели. Компания поднимает Llama или DeepSeek внутри

своего контура, получает полный контроль над инфраструктурой и данными, может работать с моделью в закрытом режиме и применять к ней лёгкие методы настройки. Всё это серьёзная инженерная работа, и для многих задач её более чем достаточно. Но веса остаются чужими. Лицензия исходной модели продолжает действовать, право на глубокую модификацию ограничено её условиями, лицензировать третьим сторонам эти веса нельзя. Это не своя LLM — это приватная эксплуатация чужой.

Различение практически важное, потому что на внутренних совещаниях оба проекта часто называют одним словом — «наша LLM». За одним стоит актив с собственными правами и долгой жизнью внутри компании; за другим — аккуратно развёрнутая чужая модель, у которой есть срок годности, лицензия и вендор. Проекты разные, и путать их в смете опасно.

2.2. Четыре альтернативы — и когда они лучше

С противоположной стороны от своей LLM стоят не «менее амбициозные варианты», а четыре зрелых рабочих пути, у каждого — своя область, где он просто лучше. Полка с готовым здесь — не синоним дёшево-и-сердито. Это признание, что в подавляющем большинстве случаев готовое решение даёт нужное качество быстрее, дешевле и с меньшим риском, чем самодельное. Развёрнутый разбор каждого из них — отдельный жанр; здесь — ровно столько, чтобы увидеть границу. Объединяет их одна черта: веса в них остаются не у компании.

Первый путь — **RAG** (Retrieval-Augmented Generation): подход, при котором ответы модели дополняются поиском по своим документам, а сама модель остаётся нетронутой. Вокруг неё выстраивается инфраструктура индексации и поиска по корпоративным данным, и при каждом запросе нужные фрагменты подставляются в контекст. RAG часто оказывается правильным ответом в двух типичных ситуациях. Первая — данные быстро меняются, и переобучать под каждое обновление бессмысленно: каталог продуктов, техническая документация, внутренняя база знаний, живущая своей жизнью. Вторая — данные чувствительны, и «вшивать» их в веса модели просто нельзя по правовым причинам или из соображений безопасности. Когда руководитель формулирует задачу словами «нам нужна модель, знающая наши документы», правильный ответ в большинстве случаев — RAG.

Второй путь — **prompt engineering**: искусство составления входных запросов, которое даёт результат быстро и без инфраструктурных затрат. Ни обучения, ни дообучения, ни отдельного контура — только хорошо продуманные инструкции и примеры, подаваемые готовой модели. Этот путь в крупных компаниях недооценён по любопытной причине: он «слишком простой». Он не создаёт ощущения серьёзной работы, не вписывается в крупный бюджет, не требует команды. Между тем грамотно настроенный промпт часто решает задачу, под которую казалось необходимым дообучение. Классификация входящих обращений, первичная разметка документов, маршрутизация запросов — задачи, под которые в компании уже успели обсуждать свою LLM, нередко снимаются точной постановкой и десятком хороших примеров. Правило здесь простое: прежде чем переходить к более тяжёлым методам, стоит убедиться, что лёгкий исчерпан.

Третий путь — **LoRA** (Low-Rank Adaptation): метод лёгкой и дешёвой адаптации открытой модели под узкую задачу. Технически это надстройка — к исходной модели добавляются небольшие «дельты», которые подстраиваются под конкретные данные, а сами основные веса остаются нетронутыми. Результат получается быстро и обходится в разы дешевле полноценного дообучения. Масштаб тут характерный: надстройка весит десятки мегабайт против сотен гигабайт исходной модели — это и делает её технически лёгкой и экономически соблазнительной. Формально это не своя LLM: исходные веса компании не принадлежат, права определяются лицензией базовой модели, без неё LoRA-надстройка не работает. Но для множества

задач этого уровня адаптации достаточно, чтобы модель начала вести себя ближе к домену компании — и ближе, чем покрывает голый prompt engineering.

Четвёртый путь — **внешние API**: использование моделей через интернет без владения какими бы то ни было весами. Это, как правило, самый быстрый способ получить качество. OpenAI, Anthropic, Google и другие поставщики дают доступ к моделям лидирующего класса без капитальных затрат на инфраструктуру и команду. Для очень многих компаний именно это и есть правильный ответ — отсюда и берётся первая из пяти возможных рекомендаций книги: «ничего не делайте, используйте RAG или внешние API».

У этих четырёх есть общий принцип. Если главное — быстро получить результат, альтернативы почти всегда правильнее. Своя LLM имеет смысл только там, где эти пути объективно не подходят, — а таких ситуаций меньше, чем обычно кажется. Практический критерий коротко звучит так: если задача формулируется как «быстрее и точнее отвечать клиентам с учётом наших документов», ответ почти наверняка в этом списке; если как «корпоративный ассистент, похожий на ChatGPT, но на наших данных», стоит проверить все четыре пути, прежде чем браться за свою LLM. Если к концу Главы 5 компания не прошла фильтр оснований и готовности — ответ, скорее всего, именно из этого короткого списка.

2.3. Миф «свой GPT за год»

Последнее распространённое заблуждение, которое стоит назвать прямо, — фраза «сделаем свой GPT за год». Она регулярно звучит на совещаниях, и произносящий её редко имеет в виду буквально GPT-4 или GPT-5. В массовом восприятии слово «GPT» давно оторвалось от конкретной линейки моделей и стало общим именем «сильной современной LLM» — примерно так же, как когда-то «ксерокс» стал именем для любой копировальной машины. За фразой «свой GPT» стоит амбиция вполне прозрачная: собственная модель фронтирного или около-фронтирного класса, с качеством, сопоставимым с тем, что продают лидеры рынка.

Проблема этой амбиции не в смелости, а в масштабе несоответствия. Фронтирные модели требуют сотен миллиардов параметров, многих триллионов токенов обучающих данных, тысяч современных GPU и сотен человек в команде; вся эта арифметика подробно развёрнута в Главе 6 в сценарии C. Реалистичный срок до первой осмысленной версии — минимум 24—36 месяцев, и это только начало: дальше модель требует непрерывной работы, без которой быстро теряет актуальность. Год в этой арифметике — не граница возможного, а граница разговоров, обычно заканчивающихся ничем. Разрыв между ожиданием «за год сделаем» и реальностью фронтирного проекта измеряется не процентами, а порядками. Чтобы увидеть это сопоставление наглядно: «свой GPT за год» означает одновременно ускорить мировой темп таких проектов в несколько раз и уменьшить нужные ресурсы на порядок. Каждое из двух требований по отдельности — героическое; вместе они означают, что фраза не описывает реальный проект, она его замещает.

Хорошая параллель — двое спорят, можно ли за год построить Boeing 747 в гараже. Правильный ответ — нет. Правильный вопрос — другой: какой самолёт вообще можно построить в своих условиях и за какой срок. Ответ будет скромнее, но он тоже будет самолётом.

Парадокс спора про Boeing в гараже в том, что сам вопрос «можно ли за год» тихо замещает настоящий — «с какими ресурсами и основаниями вы заходите в проект». Замена незаметна, и к моменту, когда её замечают, полгода уже ушло на разговоры.

Это и есть настоящий разговор о достижимом. Для компаний, у которых есть основания заходить в собственный проект, пространство возможного очерчено довольно ясно. Малая модель под узкую задачу — 2—4 месяца от решения до полезного результата. Адаптированная модель на основе открытой базы — 4—9 месяцев. Собственная модель среднего класса — 18—24 месяца. Каждая цифра здесь имеет свою природу. Малая быстра потому, что задача наме-

ренно сужена: узкий домен, понятный критерий успеха, небольшая команда. Адаптированная опирается на то, что открытая база уже прошла многомесячное предобучение, и добавляет к нему смещение в сторону собственного корпуса. Собственная модель — это полное предобучение, но без амбиции на фронтирный размер: меньше параметров, меньше данных, меньше вычислений, и поэтому сроки укладываются в горизонт полутора-двух лет, а не растягиваются на три года и больше. Это настоящие варианты своей LLM для большинства компаний, принявших решение идти в проект; подробная развёртка каждого — в Главе 6. Такие варианты не дают результата «как у OpenAI». Но дают другое: владение собственным активом, заточенным под конкретную задачу. Это тоже своя LLM — просто не свой GPT.

Различие между «нашим GPT» и «нашей LLM» выглядит словесным, а на деле — стратегическим. Если у руководителя в голове продолжает жить «свой GPT за год», остаток книги читается против себя самого: каждая следующая глава будет звучать неубедительно, потому что читатель подсознательно будет искать короткий путь, недостающее средство, секретный приём, которого не существует. Этот несуществующий короткий путь обычно принимает вид «минимальной конфигурации», которая «на самом деле» даёт модель уровня лидеров; поиск такой конфигурации заменяет работу над реальным выбором и съедает первые месяцы проекта без видимого результата. Когда же масштаб откалиброван правильно, разговор о сценариях начинает попадать на свои места, а разговор о рисках перестаёт восприниматься как избыточный пессимизм — он становится естественной частью размера проекта.

У этой калибровки есть и обратный эффект. Когда со стола переговоров уходит фраза «свой GPT за год», вместе с ней уходит и главный источник управленческой тревоги в этой теме — ощущение, что компания опаздывает относительно чего-то невозможного. На месте тревоги остаётся задача: сопоставить собственные основания, готовность и масштаб с картиной поля. Задача тяжёлая, но решаемая. Именно в этом состоит работа книги и работа руководителя, который её читает.

Часть II. Устройство и процесс создания LLM

Не зная, из чего собрана модель и как её делают, сравнивать сценарии S/A/B/C можно только на слух. Дальнейший разговор о масштабе, ресурсах, командах и рисках держится именно на этом фундаменте.

Глава 3. Анатомия LLM

Описать LLM изнутри можно только учебником, к которому мы здесь не стремимся; задача скромнее и важнее — передать ощущение устройства. Модель — большое, тонко устроенное сооружение; в ней нет случайных деталей, и нет деталей, которые можно «просто увеличить», не потревожив всё остальное. Каждая «ручка», которую можно крутить, стоит денег — и не в переносном смысле, а в виде вполне конкретных счетов за вычисления, память и время.

К Главе 6 те же самые «ручки» соберутся в четыре целостных сценария: без чувства устройства они читаются как таблица, а должны читаться как разные масштабы одного реального сооружения.

3.1. Как модель видит текст

Прежде чем говорить о том, что происходит у модели внутри, стоит признать одно упрямое обстоятельство: с текстом она не работает вовсе. Строка букв, фраза, абзац — всё это живёт только на входе и на выходе, как упаковка. Внутри модель работает с токенами — минимальными кусками, на которые разрезан текст, и последовательностью этих кусков она и оперирует.

Токен — не слово и не буква, а что-то между ними. Длинное слово вроде «предложение» распадается на несколько коротких фрагментов; короткое и частое слово может остаться одним токеном целиком. Модель не видит ни букв в человеческом смысле, ни слов в школьном; она видит, в каком порядке друг за другом идут знакомые ей куски. То, что для нас — беглое чтение, для неё — перебор элементов заранее известного набора.

Сам этот набор — словарь модели. Одна из её фиксированных характеристик, такая же технически жёсткая, как размер алфавита в обычной письменности; только тут это «алфавит осмысленных фрагментов». Чем словарь больше, тем точнее модель умеет «говорить», тем тоньше нарезка, тем выше её гибкость, — но тем тяжелее внутренняя машинерия, дороже хранение, медленнее каждый шаг. Как и всё в этой истории, размер словаря — компромисс, а не оптимизация: выбор делается однажды, на самом старте, и потом живёт с моделью всю её жизнь.

На практике нарезка выглядит так: короткое частое слово оказывается одним куском целиком; слово средней длины распадается на два-три фрагмента; редкое, длинное или специфическое — на большее число мелких частей, иногда вплоть до отдельных букв. Нарезка не похожа на разбор по морфемам, как в школьной грамматике; это статистическая нарезка, подобранная так, чтобы наиболее частые сочетания в корпусе получили короткую запись, а редкие — удлинённую. Внутренняя логика тут инженерная, не лингвистическая: словарь не знает, где корень, а где приставка, он знает, что часто встречается в текстах, на которых его учили.

А теперь первое место, где абстракция встречается с экономикой. Стандартные словари, которыми пользуются открытые модели, собирались в основном на англоязычных корпусах. Английский в них лежит экономно: частые слова попадают в словарь целиком, редкие — дробятся аккуратно. Русский в тех же словарях лежит с перекосом. Кириллица расходует заметно больше токенов, чем латиница: тот же самый смысл на русском занимает в полтора-два раза больше кусков, чем на английском. Это не стилистическая особенность и не досадное неудобство — это множитель. Прямой, грубый множитель ко всем затратам: к стоимости обучения, к стоимости каждого ответа, к расходованию контекстного окна, в которое укладывается разговор.

У этого множителя есть осязаемые проявления, о которых не всегда вспоминают. Компания, работающая преимущественно с русскоязычным текстом через открытую модель, платит надбавку ежедневно и незаметно — в виде чуть более высоких счетов за каждый запрос. Длинный разговор с пользователем достигнет потолка контекстного окна раньше, чем тот же разговор на английском; при прочих равных русскоязычный ассистент «устаёт» и теряет нить диалога быстрее — просто потому, что каждый его шаг обошёлся дороже в токенах. На масштабе массового сервиса эта разница в полтора раза превращается в миллионы лишних токенов в день и в заметное сокращение того, что модель способна удержать в контексте за одно обращение.

Отсюда — первое осмысленное решение, которое приходится принимать всякому, кто собирается делать модель для русского языка. Имеет смысл собрать свой словарь — свой токенизатор, обученный на русских корпусах. Это короткий отдельный этап, предшествующий основному обучению, и детально он разбирается в разделе 4.2. Экономия, которую он даёт, окупается всей последующей жизнью модели: дешевле предобучение, дешевле инференс (работа модели после обучения, на каждом запросе пользователя), длиннее доступный контекст в тех же инженерных рамках. Поэтому собственный токенизатор — не пропускаемая деталь и не технический нюанс, который можно оставить команде на потом. Это один из первых выборов, определяющих экономику всего проекта; словарь, собранный на коленке, аккуратно тянется шлейфом через все последующие счета.

Теперь, когда понятно, в каком виде текст попадает внутрь, можно посмотреть, что с ним там происходит — куда попадают эти последовательности токенов и в каком пространстве начинает жить их смысл.

3.2. Как модель хранит смысл

Снаружи токен — номер в словаре. Внутри модели он живёт в другой форме: у каждого токена есть эмбеддинг (embedding) — длинный вектор, набор из сотен или тысяч чисел, в котором упакована информация о том, как этот токен связан с другими. Не «номер слова» и не «код буквы»; это координаты точки в многомерном пространстве, и именно с такими точками модель и работает. Токен на входе — как почтовый индекс; эмбеддинг — как сама точка на карте, со всей её географией.

Образ, который работает ближе к физике, чем к математике. Огромное пространство — не три измерения, как в нашем физическом мире, а тысячи. В этом пространстве каждый токен — точка, и расположены они не случайно: близкие по значению оказываются близкими геометрически, далёкие — удалёнными. «Король» и «королева» располагаются рядом; «тигр» и «лев» — тоже; «стол» и «ложка» оказываются ближе друг к другу, чем «стол» и «галактика». Это — в укороченном виде — действительно то, что происходит внутри модели. Главное здесь — идея: смысл в LLM устроен как геометрия. Близость слов есть расстояние, аналогия есть параллельный сдвиг, противоположность — направление.

Ближе всего это пространство по устройству напоминает звёздную карту. Есть плотные скопления — области, в которых слова «гравитационно» связаны множеством устойчивых смысловых нитей: бытовая речь, политика, международные новости, поп-культура. Есть разреженные окраины — области, в которых точек мало и расстояния между ними большие: редкие языки, узкие профессиональные жаргоны, специализированные технические темы. Есть и пустоты — области, куда обучающие данные почти не заглядывали, и где модель, вынужденная туда перемещаться, теряет уверенность шага. Эта карта — не абстрактная схема, а во многом портрет того, на чём модель училась: по плотности скоплений можно прочитать, какие темы были в её корпусе обильны, а какие — проходили поодаль.

Это пространство не возникает само по себе и не прописывается инженерами вручную; оно складывается в процессе обучения. Модель видит триллионы последовательностей токенов и постепенно размещает их в смысловом пространстве так, чтобы слова, встречающиеся в похожих контекстах, оказывались рядом. Геометрия смысла вытекает из статистики употребления: если два слова ведут себя в текстах одинаково, модель начинает считать их близкими. Это не тот способ хранить смысл, каким пользуется человек, и любые аналогии с человеческой памятью быстро подводят, — но это способ, эффективно работающий для текста. И ещё одно замечание, связанное с разговором из Главы 2: именно эта обученная геометрия — координаты всех точек и связи между ними — и есть значительная часть того, чем компания «владеет», когда речь идёт о владении весами. Веса — это, в том числе, и конкретное расположение «короля», «королевы» и всего остального в тысячах измерений; чужую модель можно развернуть приватно, но геометрию её внутреннего мира нельзя изменить — она у неё одна на всех.

Теперь — тонкий момент, который часто пропускают. Сам по себе механизм, работающий с эмбеддингами, последовательность токенов не видит. Казалось бы, странно: модель ведь читает текст слева направо, как мы, — разве порядок не заложен в устройство? На деле нет. Каждый токен модель рассматривает как точку в пространстве, и сам факт «этот токен идёт после того» для неё не существует, пока его отдельно не закодировать. Порядок слов — не побочный факт чтения, а инженерный сюжет, решаемый специально.

Способ, которым модели сообщают порядок токенов, — отдельная техническая задача, и от её решения зависит одна очень практическая вещь: на какую максимальную длину текста модель способна распространить свою работу. Одни способы кодирования хорошо переносятся на последовательности длиннее тех, на которых модель училась; другие ломаются уже на выходе за привычное окно. Существуют разные подходы; для руководителя их перечень неважен. Важно, что способность модели уверенно обрабатывать длинные документы — не «просто выкрутить ещё одну ручку», а следствие вполне конкретного инженерного решения, принятого на ранней стадии её конструирования.

Когда в рекламе новой модели звучит «контекстное окно в сотни тысяч или в миллион токенов», за этой цифрой стоит не общая магия масштаба, а в том числе выбор способа кодирования позиций — наравне с готовностью платить квадратичную стоимость центрального механизма. Два разных инженерных решения, принятых на ранних этапах, формируют то, что потом будет продаваться как свойство продукта.

И ещё одна линия, которую стоит дочертить, пока мы ещё в смысловом пространстве. Геометрия смысла — не только удобный образ для объяснения; это причина, по которой LLM ведут себя так, как ведут. Модель, отвечая на запрос, по сути перемещается в этом пространстве: выбирает направление, движется туда, шаг за шагом подбирая токены, которые ближе всего к текущему смысловому положению. Это объясняет и её сильные стороны, и её болезни. Модель хорошо попадает туда, где её смысловое пространство плотно заполнено, — где в обучающих корпусах было много близких примеров. И плохо — там, где в пространстве пустота: редкие темы, специализированные домены, узкие профессиональные контексты. Для компаний с собственной предметной областью эта геометрия важна в практическом смысле: там, где в смысловом пространстве открытой модели зияет пустота, нередко лежит самое ценное содержание бизнеса, — и именно эту пустоту имеет смысл заполнять специализированным обучением или собственной моделью. Этот сюжет вернётся в Главе 5, когда речь пойдёт об основаниях браться за проект, и в Главе 7 — о данных.

Пространство, в котором модель живёт, описано; порядок токенов в этом пространстве обозначен как отдельный инженерный выбор. Остаётся главный вопрос — как модель связывает эти точки между собой, какой механизм заставляет все они работать вместе. Здесь и находится центральный приём всех современных LLM и, одновременно, главная статья их расходов.

3.3. Сердце модели — механизм внимания

У всех современных LLM одно архитектурное сердце — трансформер (transformer). Модели отличаются масштабом, особенностями обучения, дополнительными инженерными хитростями, но основа у них общая. И в центре этого сердца — один приём, давший архитектуре её имя и всей области — её нынешний облик. По-английски он называется attention; по-русски — внимание.

Идея внимания описывается обманчиво просто. Каждый токен в последовательности «оглядывается» на все предыдущие и решает, кто из них важен для того, чтобы его самого понять. Местоимение в середине предложения ищет среди предыдущих токенов то существительное, к которому оно относится. Слово в конце длинного абзаца дотягивается до начала сюжетной нити. Глагол подбирает подходящее согласование. Всё, что мы воспринимаем как «модель что-то понимает в тексте» — удержание связей, улавливание смысла, чувствительность к стилистике, — работает именно через этот механизм. Без внимания связный текст распадается на бессвязные фрагменты; с вниманием — складывается в единое целое.

Это именно сердце конструкции, а не один из равноправных элементов. Остальные части трансформера существуют, по сути, чтобы обслуживать работу внимания — готовить для него представления токенов на входе, преобразовывать его выходы, передавать их следующему слою, где всё повторится снова. В любой современной LLM — десятки таких слоёв, и в каждом внимание работает заново. Биение сердца — не единичный акт: сотни последовательных биений на каждый сгенерированный токен, и каждое имеет свою цену.

Описывается просто; стоит — дорого. Словесное описание укладывается в один абзац; реализация в рабочей модели требует огромного количества вычислений и памяти. Именно здесь, в самом центре конструкции, модель оказывается наиболее прожорливой. Сердце бьётся — и на каждый его удар уходит непропорционально много топлива.

Причина — способ, которым стоимость внимания связана с длиной текста. Она растёт не линейно, а квадратично. Удвоить размер окна, в котором модель видит контекст, — значит увеличить вычисления примерно вчетверо. Утроить — увеличить вдевятеро. Это не технический артефакт, от которого можно избавиться простыми ухищрениями; это коренное свойство механизма, прямое следствие того, что каждый токен должен «посмотреть» на каждый другой. Существуют приближённые варианты, смягчающие квадратичность, — часть современных моделей их использует, — но в базовой форме эта квадратичность остаётся константой индустрии. Её нельзя отменить; можно только платить за неё.

Квадратичность касается не только счёта. Она касается и памяти: чтобы внимание работало, модели нужно хранить промежуточные таблицы «каждый с каждым», и размер этих таблиц тоже растёт как квадрат длины окна. На коротком контексте это — шум; на длинных окнах именно память, а не вычисления, становится узким горлом. Инженерные приёмы, позволяющие работать с большими окнами на ограниченном железе, существуют и активно развиваются, но все они выстроены вокруг одной задачи — как-то примириться с этим квадратичным аппетитом, не отменяя его.

Отсюда — одна из главных экономических констант всей области, и, пожалуй, центральная формула этой главы: длинный контекст — роскошь. И роскошь заметная. Поддерживать окна в сотни страниц текста технически можно; но каждая страница, добавленная к окну, умножается на стоимость — и в обучении, и в каждом последующем ответе модели. Когда в рекламе новой модели звучит эффектная цифра «окно в миллион токенов», за ней стоит не изящное улучшение алгоритма, а готовность платить квадратичную стоимость в большом объёме — на инфраструктуре, способной это выдержать. Длинный контекст — не очередная опция, добавленная в список возможностей; это отдельный бюджет и отдельная инженерная дисциплина.

Для руководителя, выбирающего направление для собственной модели, это соотношение имеет практический смысл. Если главная ценность, которую модель должна давать, требует работы с длинными документами целиком — анализом договоров на сотни страниц, сведением больших баз внутренней документации, обработкой долгих разговоров, — длинный контекст становится одной из определяющих характеристик проекта. А раз так, то и кластер под обучение, и размер команды, и сроки, и инференсная инфраструктура после запуска будут считаться уже с этим множителем внутри. Компании, для которых длинный контекст не критичен и задачу можно решать нарезкой документа на фрагменты с последующей сборкой, чаще всего на этом множителе экономят, — и поступают разумно. Это одно из тех редких мест в проекте LLM, где дорогую опцию не стыдно отложить: она не только стоит много денег, но и добавляет инженерной хрупкости.

Стоит закрыть разговор о сердце модели одним наблюдением — тем, как эта конструкция порождает текст. Сама генерация устроена честно и прямолинейно: модель предсказывает следующий токен, приписывает его к уже сказанному и делает следующий шаг — снова предсказывает один токен, снова приписывает. Шаг за шагом, слово за словом, до конца ответа. Никакого «планирования ответа целиком» не происходит; никакого наброска с последующей правкой; модель идёт вперёд по токенам, каждый раз опираясь только на то, что уже написано. Это и есть та самая непоседливая природа LLM, которую иногда списывают на «творческий характер»: небольшой сдвиг в первом же токене — другой выбор на развилке, другое направление в смысловом пространстве — способен увести весь ответ в другую сторону.

У этого свойства есть прямое продуктовое следствие. Любой контроль над тем, что модель скажет, выстраивается либо на входе — аккуратной формулировкой запроса, предварительной оснасткой, вспомогательными подсказками, — либо на выходе — проверкой уже сказанного и, при необходимости, новой попыткой. Промежуточного вмешательства нет; редактировать ответ «на ходу» не получится в принципе. Это свойство — фундаментальное, а не настроечное, и оно многое объясняет в том, как LLM ведут себя после запуска; в Главе 9 при разборе продуктовых рисков к этому ещё придётся вернуться.

Центральный механизм разобран. Остаётся посмотреть, какими ручками измеряют масштаб получившегося сооружения — что на самом деле стоит за цифрами, привычно звучащими в разговорах о моделях: «семь миллиардов параметров», «семьдесят слоёв».

3.4. Ручки масштаба

В любом разговоре о размере модели регулярно звучат две цифры — число параметров и количество слоёв. «Семь миллиардов параметров». «Семьдесят слоёв». Эти две величины и стоят за словами «большая модель» и «маленькая модель»; всё остальное — подробности, которые обычно настраиваются автоматически. Поэтому полезно усвоить их как две физические ручки, каждую из которых можно крутить, — и каждая крутится дорого.

Первая ручка — число параметров. Параметр — внутреннее число модели, одно из тех, которые настраиваются в процессе обучения, чтобы на выходе получался осмысленный ответ. Для управленческого разговора самая удобная интуиция такая: параметры — условно ёмкость памяти модели, сколько знаний в ней в принципе может уместиться. Большая модель запомнит больше фактов, стилей, языков, доменов; маленькая будет работать в узкой области, но всё остальное ей придётся оставить за скобками. Это не строгая метафора — параметры не хранят факты кусочками, как карточки в картотеке, — но для разговора о масштабе её достаточно.

Величины, о которых принято говорить, измеряются большими порядками. Открытые модели среднего размера — от семи до тринадцати миллиардов параметров. Крупные открытые модели — десятки миллиардов. Фронтирные модели — сотни миллиардов, а при использовании особой архитектурной схемы, о которой пойдёт речь чуть позже, — и триллионов.

Когда в публикациях встречается сокращение «7B» или «70B», речь идёт именно об этом — о миллиардах параметров, с приписанной английской буквой B от billion. Два устойчивых якоря, которые удобно держать в голове: открытая линейка Llama — это разноразмерные модели, от небольших до крупных, с подробно опубликованными характеристиками; DeepSeek V3 — одна из крупнейших открытых моделей последнего времени, тоже с развёрнутым техническим отчётом. По этим двум маркерам легко сверять любые другие упоминания, избегая фактологических ошибок.

Вторая ручка — количество слоёв. Слой — одна последовательная ступень обработки, на которой модель с помощью механизма внимания и нескольких других приёмов пересчитывает представления всех токенов. Каждый токен проходит через все слои по очереди; на каждом слое он получает чуть более точную картину того, чем окружён, и того, чем в итоге должен стать на выходе. Слои — условно глубина рассуждения, сколько последовательных шагов обработки проходит каждый токен, прежде чем модель решит, что сказать дальше. Простая реакция требует меньшей глубины; сложный вывод, длинная логическая цепочка, тонкий стилистический ход — большей.

Эти две ручки связаны, но не эквивалентны. Можно собрать широкую и плоскую модель — много параметров, мало слоёв; можно — узкую и глубокую; можно — сбалансированную. Практика показывает, что крайние пропорции работают хуже, и в обычных моделях соотношение подбирается в определённых инженерных рамках. Но это не значит, что выбор между «шире» и «глубже» — второстепенный: от того, как две ручки настроены относительно друг друга, зависит и скорость ответа, и способность модели к сложным задачам, и то, как хорошо она принимает последующее обучение.

Поворот любой из ручек вверх — не просто «сделать модель больше». Это разом изменить несколько вещей. Больше параметров — больше памяти под модель, больше вычислений на каждом шаге, большее количество данных, необходимых для того, чтобы модель научилась не на песке. Существует полезный инженерный якорь, известный как закон Chinchilla: в первом приближении на каждый параметр модели приходится примерно двадцать токенов обучающих данных. Для модели в тридцать миллиардов параметров этот ориентир означает порядка шестисот миллиардов токенов данных; для модели в триста миллиардов — уже шесть триллионов. Поворот ручки параметров на один порядок тянет за собой поворот ручки данных на тот же порядок. Этот же якорь связывает Главу 3 с Главой 7, где речь пойдёт о данных — где их брать, сколько они стоят, и почему к определённому масштабу модели «просто добрать данных» становится самостоятельной инженерной проблемой.

То же со второй ручкой. Добавить слоёв — не только сделать модель «умнее»; это умножить количество последовательных операций на каждый токен и, значит, напрямую замедлить и обучение, и каждый последующий ответ. Глубокая модель обычно умнее плоской, но дороже во всех смыслах — и в обучении, и в эксплуатации после запуска. Поэтому соотношение параметров и слоёв — не только техническая, но и экономическая настройка: две модели одного и того же суммарного размера могут иметь заметно разную стоимость инференса, в зависимости от того, как распределены их параметры между шириной и глубиной.

Всё остальное в модели — производное от этих двух ручек. Ширина внутренних представлений, количество «голов» внимания, размеры промежуточных преобразований между слоями — всё это связано с основными параметрами и обычно подбирается по отработанным правилам пропорций. Когда разработчики говорят «мы увеличили размер модели», в подавляющем большинстве случаев речь идёт именно о двух ручках, которые только что были описаны; остальные настраиваются автоматически вокруг них. У руководителя, читающего отчёт команды, нет нужды помнить десяток параметров — достаточно помнить две и понимать, что одна отвечает за ёмкость, другая за глубину, и обе дорого стоят в обе стороны.

Итог прост. Размер модели — не одна цифра и не цельная величина; это комбинация двух ручек, каждая со своей ценой. «Большая модель» сама по себе не означает «хорошая модель». Модель с большим числом параметров, обученная на недостаточном объёме данных, работает хуже, чем модель поменьше, но обученная как следует; модель с большой глубиной, но плохо сбалансированной архитектурой, теряет в качестве и одновременно дорожает в эксплуатации. В индустрии это хорошо известно, и именно поэтому серьёзные проекты тратят значительную часть подготовки на выбор конкретных значений двух основных ручек и пропорций вокруг них, а не просто «крутят побольше, пока позволяет бюджет».

Но две ручки задают только общий масштаб. Поверх них — ещё один выбор, уже не количественный, а архитектурный. Выбор, способный радикально изменить экономику того, что эти ручки построили.

3.5. Один архитектурный выбор, меняющий экономику

До сих пор мы говорили о модели как о цельной машине: вход, последовательность слоёв, механизм внимания, выход. Каждый токен идёт через всю конструкцию целиком, каждый параметр участвует в каждом шаге. Эта классическая, общепринятая схема называется dense — плотная. Большинство моделей предыдущего поколения устроены именно так: если модель насчитывает сто миллиардов параметров, то в каждом маленьком шаге генерации каждого токена участвуют все сто миллиардов. Честный, предсказуемый, архитектурно простой путь — и, что важно для нашего разговора, дорогой прямо пропорционально своему размеру.

А теперь — альтернатива. В архитектуре со «смесью экспертов» — MoE, Mixture of Experts — каждый токен проходит не через всю модель, а только через небольшую её часть. Модель устроена как коллегия специалистов: маленький внутренний «маршрутизатор» на каждом шаге решает, какие эксперты должны «включиться» для данного токена, и остальные остаются в покое. На один токен «включается» не вся модель, а, скажем, восьмая или шестнадцатая её часть. Это не уловка и не способ сэкономить на качестве; это архитектурный приём со своей инженерной и экономической логикой.

Картина становится понятнее через образ двух способов консультации. В dense на каждый вопрос отвечает вся команда целиком — все сотрудники вовлечены, независимо от того, разбираются они в теме или нет; получается равномерно и предсказуемо, но трудоёмко до неприличия. В MoE вопрос адресуется коллегии, устроенной как поликлиника: маршрутизатор на входе смотрит на запрос и отправляет его нужному специалисту. Те, кому вопрос понятен, отвечают; остальные молчат и не тратят времени. Клиника с сотней врачей способна принимать сложные случаи всех мастей, но каждый конкретный пациент занимает время лишь одного-двух специалистов, а не всей сотни разом.

Экономический смысл этого приёма прост и важен. Он позволяет иметь очень большую суммарную модель с умеренной стоимостью запуска. Суммарное число параметров исчисляется десятками или сотнями миллиардов; реально используемая на каждом шаге — только небольшая часть. Модель остаётся большой по ёмкости — коллегия специалистов в целом обладает огромным совокупным знанием, — но на каждый отдельный запрос она отвечает экономно. Для крупных моделей это принципиально: MoE делает возможной эксплуатацию того, что в плотном варианте стало бы неподъёмно дорогим в инференсе.

Именно этим путём пошли несколько заметных открытых моделей. DeepSeek V3 построена как MoE-архитектура: её суммарное число параметров исчисляется сотнями миллиардов, а реально активируется на каждом шаге заметно меньшая часть; этим удаётся удерживать стоимость инференса на уровне значительно меньшей плотной модели. Mixtral — другая открытая линейка того же семейства архитектурных решений, меньшего суммарного размера, но с той же идеей избирательной активации. Это не единственные примеры — MoE-архитектуры есть

и у части фронтирных закрытых моделей, только без публичных технических отчётов, — но DeepSeek и Mixtral остаются двумя заметными точками, на которые удобно ориентироваться, когда заходит речь об этом архитектурном выборе.

Было бы слишком красиво, если бы у MoE не было обратной стороны. Она есть, и значительная. MoE сложнее в обучении: инженерам приходится отдельно учить маршрутизатор распределять токены между экспертами равномерно, не сваливая всё в одного «дежурного»; это обучение сопряжено со своей особой нестабильностью, с которой у индустрии постепенно копится опыт, но которая остаётся существенным фактором. MoE дороже в отладке: когда модель начинает вести себя странно, вопрос «что именно пошло не так» усложняется, потому что разные токены теперь идут разными маршрутами и разные эксперты специализируются на разных подзадачах. MoE менее предсказуема в качестве: при недостаточной осторожности одни эксперты остаются недозагруженными, а другие — перегружаются, и это плохо сказывается на итоговом поведении модели. И наконец, MoE требует больше памяти под веса: хранить нужно всех экспертов, даже если в конкретном шаге работает только часть. Экономия на вычислениях не означает автоматической экономии на инфраструктуре.

Отсюда — управленческий смысл этого выбора. Dense проще, надёжнее, с понятной экономикой, но дороже на большом масштабе. MoE экономичнее в инференсе на большом масштабе, но сложнее и требовательнее к команде. Какая сторона этой развилки правильнее, зависит от задачи и от команды: достаточно ли компетенций, чтобы справиться с тонкостями MoE; достаточно ли масштаб модели велик, чтобы экономия на инференсе оправдывала усложнение; достаточно ли предсказуемое качество важнее, чем итоговая стоимость. Это один из тех вопросов, на которые нельзя ответить за пределами конкретного проекта. В Главе 6 он неизбежно возникнет при разговоре о сценариях B (Собственная) и C (Фронтир); там же он будет обсуждаться предметно.

Одна сторона этой развилки заслуживает отдельного замечания. Для пользователя модели разница между dense и MoE практически незаметна: и та, и другая принимают на вход текст и выдают на выходе текст, и внешнее поведение может быть очень близким. Внутри же — это два разных мира разработки, инфраструктуры и эксплуатации: разные инженерные дисциплины, разные диагностические приборы, разные способы обучать, разные траектории ошибок. Это важно именно для руководителя: выбор, невидимый снаружи, формирует существенную часть того, как устроен проект изнутри, — кого и сколько нужно нанимать, какой кластер собирать, сколько времени закладывать на отладку. Архитектурная развилка транслируется в организационную прежде, чем в продуктовую.

Важно одно: выбор между dense и MoE — не техническая деталь, которую можно оставить команде на самом низу. Это одно из ключевых решений, определяющих экономику проекта и то, что именно будет построено в итоге. Для небольших специализированных моделей — тех, что в Главе 6 собраны в сценарий S (Малая), — этот вопрос обычно не стоит: dense достаточно, MoE избыточна. Для крупных проектов, где модель измеряется сотнями миллиардов суммарных параметров, — стоит всерьёз и заслуживает внимания на уровне, на котором принимаются стратегические архитектурные решения. И для команды, в которой опыта с MoE нет, честный ответ нередко такой: dense — не потому что он лучше в принципе, а потому что мы сейчас умеем с ним работать, и это само по себе половина качества будущей модели.

Отдельные ручки и один архитектурный выбор разобраны. Остаётся свести всё вместе — увидеть, где именно в получившейся конструкции находятся дорогие места, и как они связаны со сценариями, которые составят Главу 6.

3.6. Где в этой конструкции дорогие места

Две ручки — параметры и слои — задают общий масштаб. Но дорогие места конструкции ими не ограничиваются. К двум основным ручкам добавляются ещё две, и у каждой своя цена. Первая — длина контекста: та самая квадратичная история. Вторая — мультимодальность: способность модели работать не только с текстом, но и с изображениями, аудио, видео. Каждая из этих двух — не «приятное дополнение», а отдельная большая статья бюджета.

Не вариатор, где любая комбинация настраивается плавно и незаметно; а четыре больших переключателя, каждый со своим прайс-листом. Поворот любого из них вверх — прыжок, а не плавное нарастание. Параметры, слои, длина контекста, число модальностей — четыре рычага, и именно их положение определяет, во что обойдётся проект.

У длинного контекста своя экономика. Одна строчка в продуктовом описании модели — «окно в несколько сотен тысяч токенов» — стоит за собой отдельной инженерной программы, которую команда ведёт наравне с основной разработкой.

Мультимодальность устроена иначе, но стоит не меньше. Текстовая модель — это одна конструкция, работающая с одной последовательностью токенов. Мультимодальная модель — по сути, несколько конструкций, связанных общим рамочным приёмом: отдельные компоненты-энкодеры, которые переводят изображения, аудио или видео в ту же форму, в которой модель уже умеет работать; отдельные обучающие данные — парные корпуса «картинка — описание», «звук — транскрипт», «видео — разметка»; отдельные бюджеты на разметку; отдельные процедуры оценки; отдельные инженеры, разбирающиеся в соответствующей модальности. Мультимодальная модель — это, грубо говоря, несколько проектов, собранных в один. Стоимость растёт не сложением, а интегрально: команда больше, данные сложнее, оценка запутаннее, отладка дольше, и зависимости между модальностями создают собственный класс ошибок, которых в чисто текстовой модели не было.

К этому добавляется ещё одно обстоятельство, которое часто недооценивают. Каждая новая модальность — это не только больше данных и больше инженеров; это отдельные правовые и этические контуры. Изображения и видео несут в себе сложную историю согласий и прав на использование, заметно более запутанную, чем право на публикацию текста; звук приносит с собой вопросы голосовой идентификации; медицинские изображения — особый режим регулирования. Там, где текстовая модель работает с письменным корпусом, мультимодальная модель берёт на себя сразу несколько юридических миров, каждый со своими правилами. Для команды это — дополнительный пласт подготовки, которого в текстовом проекте просто не было.

Конец ознакомительного фрагмента.