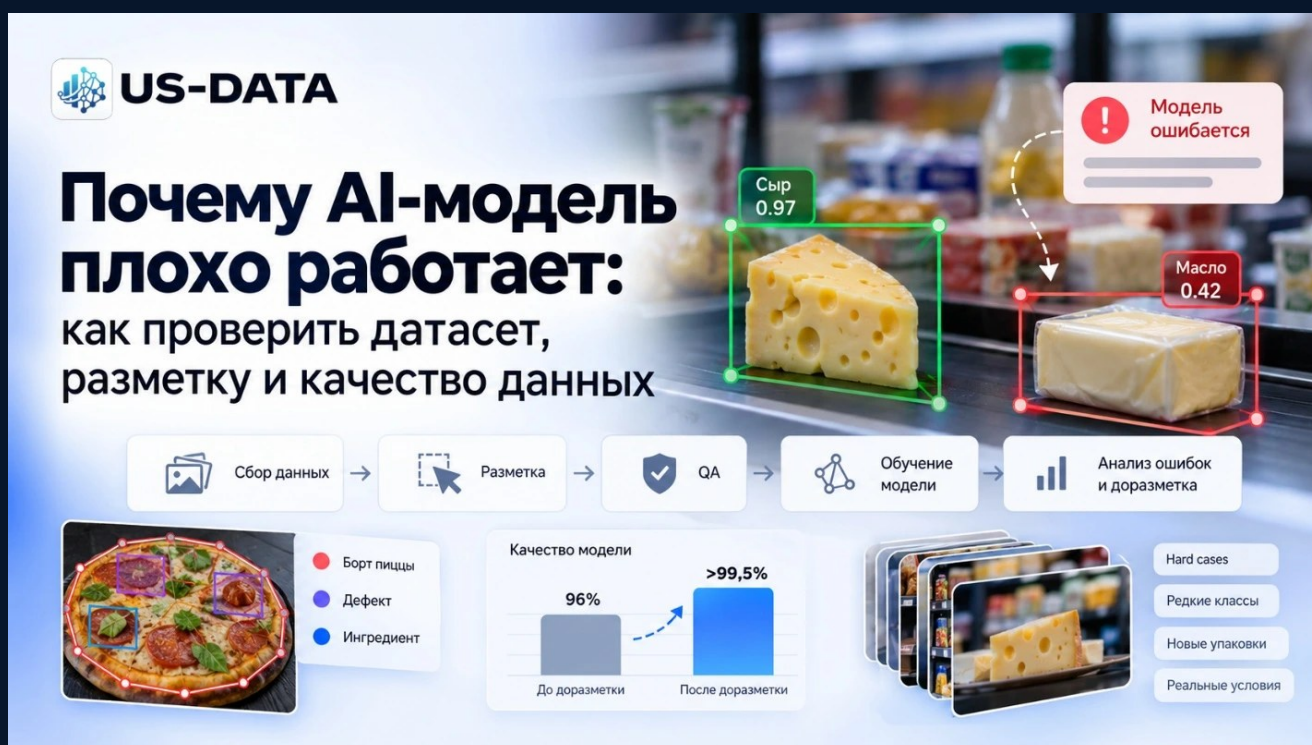


Почему AI-модель плохо работает:

как проверить датасет, разметку и качество данных

Практическое руководство по аудиту датасета, QA, hard cases, data drift и continuous annotation на основе реальных проектов US-DATA.



НАВИГАЦИЯ ПО РУКОВОДСТВУ

Содержание

От диагностики production-ошибок до управляемого цикла обновления датасета.

Модель ошибается — сначала поймите, где именно	4
1000 изображений — это ещё не датасет: как проектировать покрытие данных	10
Если два разметчика видят объект по-разному: проблема может быть не в людях, а в ТЗ	19
Контроль качества разметки: что проверять кроме «правильно / неправильно»	26
Как исправить уже размеченный датасет, не начиная проект с нуля	34
Data drift: почему модель работала вчера, а сегодня начала ошибаться	41
Hard cases: какие примеры сильнее всего улучшают модель	49
Кто отвечает за качество данных: ML, QA, разметчики и domain expert	57
Правовые и управленческие риски датасета: provenance, персональные данные и governance	66
Датасеты 2030: как изменится разметка данных в ближайшие 3–5 лет	75

Показатели качества в кейсах относятся к внутренним метрикам конкретных проектов. Они не обозначают accuracy, precision, recall или F1, если это прямо не указано в тексте.

ПРАКТИКА US-DATA

Реальные данные из проектов

96% → >99,5% Внутренняя метрика проекта после целевой доразметки	1000+ Минимум изображений на SKU в одном из проектов
100 000+ Изображений в проекте сети пиццерий	3-25 Обновлений дизайна упаковки в отдельные месяцы
60 / 40 «В заказ / под списание» в SKU-проекте	

Цифры относятся к разным проектам и условиям. Требования к SKU приведены для конкретного проекта, а не как универсальная норма.

ГЛАВА 01 / 10

Модель ошибается — сначала поймите, где именно

ПРИНЦИП ДИАГНОСТИКИ

**Сначала локализовать ошибку.
Затем менять модель.**

01Расхождение с бизнес-данными	02Ручная проверка	03Отбор hard cases
04Целевая доразметка	05Дообучение	06Повторная оценка

Кейс US-DATA · Сыр и масло

Отрасль	Продуктовый ритейл
Задача	Распознавание товаров в видеопотоке
Проблема	Ошибки между визуально похожими категориями: сыр и сливочное масло
Решение	Сопоставление с кассовыми продажами, ручная проверка, targeted annotation и дообучение
Результат	Примерно 96% → более 99,5% по внутренней метрике проекта

Когда качество ML-модели перестаёт устраивать команду, первое желание — искать проблему в самой модели: менять архитектуру, подбирать другие гиперпараметры, увеличивать число эпох обучения или пробовать более сложный backbone. Но прежде чем переделывать модель, полезно ответить на более простой вопрос: где именно она ошибается и что объединяет эти ошибки?

Среднее значение метрики само по себе даёт мало информации. Модель может показывать высокий результат на большей части данных и при этом систематически ошибаться в нескольких сценариях, которые критичны для бизнеса. Например, хорошо распознавать большинство товаров, но путать две визуально похожие категории. Или успешно работать на исходном тестовом наборе, но терять качество после изменения упаковки товара.

Поэтому диагностику стоит начинать не с вопроса «как улучшить модель?», а с вопроса: Какие именно данные модель не умеет обрабатывать правильно? И только после этого решать, требуется ли изменение самой модели или достаточно изменить данные, на которых

она обучается.

Высокая точность не означает, что задача решена

Хороший пример — один из проектов US-DATA для продуктового ритейла.

У заказчика уже работала ML-модель, которая анализировала видеопоток и определяла продукты. В целом система показывала высокий результат — около 96% по внутренней метрике проекта. Но оставшиеся несколько процентов ошибок приходились не случайным образом на все товары.

Модель ошибалась прежде всего там, где категории были визуально похожи.

Один из таких случаев — сыр и сливочное масло.



Сыр и масло: семантическая ошибка класса до и после QA · исходный материал US-DATA

Пример пограничной ошибки в разметке. Слева объект геометрически выделен корректно, но ему присвоен неверный класс «масло сливочное». После QA класс исправлен на «сыр». Такие ошибки особенно опасны для похожих категорий: сама форма annotation может быть правильной, поэтому формальная проверка геометрии дефект не обнаружит. Источник: US-DATA.

Здесь важна одна деталь. Ошибка находится не в контуре объекта. Polygon построен достаточно точно. Ошибка семантическая: изображению присвоен неправильный класс.

Это показывает важное ограничение обычного QA. Можно идеально проверить координаты bounding boxes, masks или polygons и всё равно передать модели плохие данные, если сами значения labels определяются нестабильно.

Ищите не отдельные ошибки, а их закономерность

В этом проекте качество работы модели контролировалось уже в production. Результат распознавания товаров сопоставлялся с фактическими продажами через кассы.

Получалась своеобразная дополнительная контрольная система:

касса показывает, что товар был продан → AI должен был зафиксировать соответствующее событие.

Если статистика продаж и данные модели начинали расходиться, команда клиента сначала локализовала подозрительные участки, а затем часть видео проверялась вручную.

Так из общего показателя вроде «модель работает на 96%» появлялась значительно более полезная информация: какой товар → в каких ситуациях → с каким другим товаром → модель путает чаще всего. После этого проблемные примеры передавались US-DATA для дополнительной разметки. Фактически работа строилась как цикл:

Процесс был итеративным: расхождение между бизнес-данными и результатом AI служило сигналом для ручной проверки; подтверждённые сложные случаи возвращались в датасет, после чего модель дообучалась и проходила повторную оценку.

Именно такой подход позволяет отличить случайный шум от системной проблемы в датасете.

Что дала точечная доразметка

Вместо того чтобы просто добавлять в датасет ещё больше случайных изображений продуктов, команда концентрировалась на тех категориях и ситуациях, где модель уже показала слабые места.

После серии таких итераций показатель качества по внутренней метрике проекта вырос примерно с 96% до более чем 99,5%.

- Результат после работы с пограничными случаями

Показатель качества по внутренней метрике проекта продуктовой сети.



Результат по внутренней метрике проекта · исходный материал US-DATA

При этом первоначальных данных в целом было достаточно для работы модели. Резерв качества находился не столько в количестве изображений, сколько в нехватке именно тех примеров, на которых модель ошибалась.

Принципиально важно различать общий объём данных и их информационную ценность для конкретной ошибки модели.

Если модель плохо распознаёт товар в одном конкретном сценарии, дополнительные десять тысяч простых изображений этого товара могут почти ничего не изменить. Гораздо полезнее добавить несколько сотен действительно сложных случаев, которые закрывают найденный пробел.

Такой подход часто называют *targeted annotation* — целевой доразметкой данных.

Production постоянно создаёт новые сложные случаи

Есть ещё одна причина, почему даже хороший первоначальный датасет со временем перестаёт быть достаточным: реальный мир меняется.

В этом же проекте US-DATA работала примерно с десятью категориями продукции. При этом дизайн упаковок регулярно обновлялся. В отдельные месяцы происходило от 3 до 25 изменений.

Для человека новый дизайн пачки масла остаётся тем же маслом. Для компьютерного зрения ситуация сложнее. Измениться могут:

- цвет упаковки;
- логотип;
- расположение текста;
- фотографии продукта;
- форма отдельных элементов;

- контрастность;
- размер надписей;

визуальное сходство с соседними категориями.

В результате хорошо работавшая вчера модель может получить объект, визуальное представление которого недостаточно похоже на данные, использованные во время обучения.

Поэтому цикл работы с ML-системой не заканчивается после первого запуска модели.

На практике он выглядит скорее так:

После запуска данные продолжают меняться, поэтому эксплуатационный цикл включает мониторинг новых ошибок, обновление датасета, дообучение модели и повторную проверку в production.

Поэтому первый этап аудита — сегментация ошибок

Когда модель показывает неудовлетворительный результат, недостаточно посмотреть на одну итоговую метрику. Нужно разложить ошибки на группы.

Например, выяснить:

Какие классы модель путает между собой? Если ошибки концентрируются между похожими классами, проблема может находиться в taxonometry, labels или недостатке пограничных примеров.

Есть ли условия, в которых качество резко падает? Темнота, блики, другая камера, необычный ракурс, новая упаковка, частично закрытый объект — всё это отдельные срезы данных.

Ошибается ли модель одинаково на всех товарах? Средняя метрика может скрывать ситуацию, когда девять категорий работают почти идеально, а десятая существенно хуже.

Возникают ли ошибки только после запуска? Если test set выглядит хорошо, а production — значительно хуже, нужно проверять соответствие обучающей выборки реальным данным.

Изменился ли сам мир после создания датасета? Ассортимент, упаковка, камеры, освещение, оборудование и бизнес-процессы со временем меняются.

Главная задача этого этапа — перейти от формулировки: «Модель работает недостаточно хорошо» к значительно более конкретной: «Модель путает классы А и В при таких-то условиях, и именно этих примеров недостаточно в обучающем датасете». Только после этого становится понятно, что делать дальше.

Когда всё-таки стоит смотреть на архитектуру

Разумеется, данные объясняют не любую проблему ML-модели. Причина может находиться в архитектуре, preprocessing, loss function, гиперпараметрах, вычислительных ограничениях или самой постановке ML-задачи.

Но изменение модели и изменение датасета требуют совершенно разных ресурсов.

Поэтому практический порядок действий разумно выстроить так:

Практический порядок диагностики следующий: сначала локализовать ошибки, затем проверить данные и метки, оценить покрытие сложных сценариев и корректность train/validation/test. Только после этого имеет смысл решать, требуется ли изменение модельного подхода.

Так команда не тратит недели на перестройку архитектуры, пытаясь компенсировать проблему, которая фактически находится в нескольких отсутствующих или неправильно размеченных сегментах датасета.

ГЛАВА 02 / 10

1000 изображений — это ещё не датасет: как проектировать покрытие данных

≥1000 изображений на SKU в проекте	60-90% площади кадра занимает объект
≥2 MP разрешение ближней RGB-камеры	≥5 ракурсов для дефектных объектов
2700-6500 K диапазон температуры света	80 / 10 / 10 train / validation / test
60 / 40 «в заказ / под списание»	

Один из самых распространённых способов оценить будущий датасет — посчитать количество изображений. «Нам нужно 10 000 фотографий». «На каждый класс должно быть не меньше 1000 примеров». «Добавим ещё несколько тысяч кадров — модель станет лучше». Но само по себе количество изображений почти ничего не говорит о том, какую часть реального мира они покрывают.

Можно собрать 10 000 очень похожих фотографий одного товара при одинаковом освещении и с одного ракурса. Формально датасет будет большим. Но стоит изменить свет, повернуть объект или показать экземпляр с необычным дефектом — и модель столкнётся с ситуацией, которой в обучающих данных фактически не было.

Поэтому при подготовке датасета важен не только его объём, но и покрытие вариативности будущей среды эксплуатации модели.

Качество данных начинается ещё до разметки

В одном из проектов US-DATA требовалось подготовить данные для моделей компьютерного зрения, определяющих степень зрелости и визуальные аномалии отдельных единиц продукции.

В данном случае нельзя было просто получить архив изображений и начать разметку.

Датасет необходимо было фактически спроектировать и собрать.

Команда US-DATA участвовала в работе непосредственно на складе: помогала организовать съёмочную зону, подготавливать продукцию, контролировать условия съёмки и проверять получаемые изображения ещё до того, как они попадали на этап разметки.



Физический сбор изображений SKU на площадке · исходный материал US-DATA

Сбор датасета для задачи оценки качества продукции. Команда работает с реальными SKU заказчика непосредственно на площадке: контролируются положение объекта, ракурс, фон, освещение и техническое качество изображения. Источник: US-DATA.

Этот пример хорошо показывает важную вещь: в некоторых ML-проектах работа с данными начинается не за компьютером.

Если будущая модель должна различать зрелость, повреждения или дефекты поверхности, ошибка на этапе съёмки может оказаться не менее критичной, чем ошибка разметчика.

Почему нельзя просто собрать фотографии из открытых источников

Для проекта использовалась фактическая товарная номенклатура заказчика. Это особенно важно для задач, где требуется оценивать не общую категорию вроде «яблоко», а состояние конкретных товаров. Внутри одного SKU могут существенно различаться:

- оттенок;
- размер;

- форма;
- текстура поверхности;
- расположение характерных элементов;
- признаки зрелости;

типы повреждений. А между разными сортами продукции эти различия могут быть ещё сильнее.

Для человека два яблока легко остаются «яблоками». Для модели компьютерного зрения светло-зелёный экземпляр, крупное красное яблоко и плод с повреждённой поверхностью представляют собой заметно разные комбинации визуальных признаков.

Поэтому вопрос для ML-команды должен звучать не так: «Сколько изображений яблок у нас есть?» а так: «Какие варианты яблок и в каких условиях представлены в этих изображениях?»

Минимум 1000 изображений — это только начало требований

Для каждого SKU в проекте требовалось собрать не менее 1000 изображений. Однако техническое задание не ограничивалось этим числом. Объект должен был:

- находиться примерно в центре изображения;
- занимать 60–90% площади кадра;
- сниматься на однородном фоне;
- быть представлен с разных ракурсов;
- находиться в фокусе;

сниматься с разрешением не менее 2 МР. Но главное — в датасет необходимо было включить характерные для данного SKU дефекты и состояния продукции. То есть тысяча изображений не являлась целью сама по себе. Цель — добиться того, чтобы внутри этой тысячи присутствовало достаточно разнообразия для обучения модели.

Один объект нужно показать модели по-разному

Особенно жёсткие требования предъявлялись к изображениям дефектов. Они должны были сниматься с нескольких направлений, включая приблизительно: вертикальный ракурс → около 30° → около 60°. Всего для дефектных объектов требовалось не менее пяти ракурсов.

Многоракурсная съёмка необходима потому, что визуальное проявление одного и того же дефекта существенно зависит от положения объекта относительно камеры.

Небольшое повреждение поверхности, которое хорошо заметно при прямом взгляде, при изменении угла может:

- частично скрыться;
- слиться с текстурой;
- попасть в тень;
- изменить визуальную форму;

стать похожим на нормальный элемент объекта. Если модель видела дефект только в одном положении, есть риск, что она выучит не его устойчивые признаки, а особенности конкретного ракурса.

Освещение — это часть данных, а не просто условие съёмки

В проекте отдельно задавались несколько режимов освещения. Слабое освещение — примерно до 500 лк. В таких условиях могут теряться детали в тенях, снижаться контрастность и искажаться цветопередача. Нормальное освещение — от примерно 500 лк. Объект и детали его поверхности хорошо различимы. Переосвещение — порядка 10 000 лк и выше. На изображении появляются локальные пересвеченные области, в которых теряются цвет и текстура. Это не попытка сделать часть фотографий намеренно плохими. Напротив, если будущая система способна столкнуться с подобными условиями, модель должна получить их примеры до запуска в production. Иначе хорошее качество на «идеальном» датасете может оказаться плохим качеством на реальном объекте.

Даже цвет света имеет значение

Отдельно учитывалась цветовая температура внешнего освещения:

Условие	Цветовая температура
Тёплый свет	2700–3500 К
Нейтральный	4000–5000 К
Холодный	5500–6500 К

Для задачи определения зрелости это особенно существенно.

Если один из признаков класса — цвет продукта, то изменение источника света способно изменить значения пикселей, хотя сам физический объект остаётся тем же.

Для человека яблоко под холодной лампой и то же яблоко под тёплой лампой — один объект.

Для модели это два несколько разных распределения визуальных признаков.

Поэтому задача подготовки данных заключается не в том, чтобы сделать все кадры максимально одинаковыми, а в том, чтобы контролируемо показать модели допустимую вариативность.

Что на самом деле означает «покрыть SKU»



Что означает покрыть SKU · исходный материал US-DATA

Покрывание датасета — это комбинация условий, а не одно количество изображений. Чем больше параметров способно влиять на работу модели, тем важнее контролировать присутствие этих вариантов в обучающих данных.

Баланс классов тоже проектируется заранее

Ещё одна частая ошибка — собрать данные в тех пропорциях, в которых их проще всего получить. Но реальная доступность изображений и полезность этих изображений для обучения — не одно и то же. В рассматриваемом проекте фотографии первоначально делились на два основных класса:

- «в заказ» — 60%
- «под списание» — 40%

Если для конкретного SKU применялась оценка степени зрелости, класс «в заказ» дополнительно делился на:

- менее спелые;
- среднеспелые;

более спелые. Эти категории, согласно требованиям проекта, должны были быть представлены примерно в равных пропорциях. Это очень важный элемент проектирования

датасета.

Если подавляющее большинство изображений показывает нормальный товар, а дефектных объектов крайне мало, модель может научиться очень хорошо распознавать наиболее частый класс и при этом недостаточно хорошо решать именно ту задачу, ради которой создаётся система.

Почему высокая общая метрика может скрывать проблему

Представим условный датасет: 950 нормальных товаров + 50 товаров с дефектами. Модель, которая практически всегда отвечает «товар нормальный», уже может показать очень высокий общий процент правильных ответов. Но для бизнеса такая система бесполезна: она почти не выполняет функцию обнаружения дефектов. Поэтому качество датасета нельзя оценивать только количеством файлов или даже одним итоговым показателем модели. Нужно смотреть: какие классы представлены → сколько примеров есть в каждом → насколько разнообразны примеры внутри класса → какие редкие ситуации присутствуют.

Train, validation и test должны быть действительно независимыми

После сбора данные в проекте делились на:

- 80% — train
- 10% — validation
- 10% — test

Но само соотношение 8:1:1 ещё не гарантирует корректную оценку модели.

Например, если сделать десять почти одинаковых снимков одного яблока подряд, а затем случайным образом распределить их по train и test, модель может увидеть практически один и тот же объект и во время обучения, и во время проверки.

Метрика получится хорошей, но она будет отвечать не на тот вопрос. Нас интересует: «Сможет ли модель работать с новым объектом?» А тест фактически проверит: «Сможет ли модель узнать ещё одну фотографию объекта, который она уже видела?» Поэтому при формировании split важно учитывать происхождение и связанность изображений, а не просто делить список файлов случайным образом.

Metadata превращают папку фотографий в управляемый датасет

Для каждого изображения в проекте требовалось сохранять metadata:

METADATA / ПОЛЯ ИЗОБРАЖЕНИЯ

filename
object_type
ripeness_stage
defect_type
angle
lighting_condition
split
light_temperature

На первый взгляд это может выглядеть как вспомогательная таблица. На практике metadata позволяют ответить на гораздо более важные вопросы. Например: Есть ли в test достаточно кадров при слабом освещении? Какие дефекты представлены только под одним углом? Не оказался ли холодный свет почти исключительно в train? На каком типе дефекта модель ошибается чаще всего? Хватает ли менее спелых объектов для конкретного SKU? Без metadata ответить на такие вопросы приходится вручную, просматривая тысячи изображений. С metadata можно анализировать датасет как структурированный объект.

Ближняя и дальняя камера решали разные задачи

В проекте использовалось два сценария съёмки.

Основная ближняя RGB-камера с разрешением от 2 MP должна была давать детальные изображения продукции. Объект занимал 60–90% кадра, а камера должна была хорошо фокусироваться в ближней зоне.

Дополнительно использовалась дальняя RGB-D камера, установленная примерно в 50 см от объекта. Её задача отличалась: получить данные, больше похожие на изображения, которые система увидит уже при реальной работе оборудования. Это очень полезный принцип: датасет должен содержать не только удобные изображения для обучения, но и изображения, похожие на будущий production-поток. Иначе может возникнуть классическая ситуация:

Иначе возможен разрыв между лабораторной оценкой и реальной эксплуатацией: высокое качество на подготовленных данных не воспроизводится на production-потоке.

Вариативность объектов важнее количества кадров одного объекта

В технических требованиях проекта прямо отмечался ещё один важный принцип: не нужно делать слишком много фотографий одной и той же единицы продукции. Гораздо полезнее увеличить разнообразие самих экземпляров внутри SKU. Например, вместо:

- 200 фотографий двух практически одинаковых яблок
- полезнее получить: десятки разных яблок — крупные, мелкие, светлые, тёмные, с разной формой и разными повреждениями. Модель должна научиться распознавать класс, а не

запоминать несколько конкретных объектов.

Характерные элементы тоже должны быть представлены в разных состояниях

У некоторых товаров существуют детали, которые легко превращаются в случайный shortcut для модели. Например, у яблока это может быть черенок. Если в обучающем наборе случайно окажется так, что:

- почти все хорошие яблоки сняты с видимым черенком,

а

- почти все повреждённые — с другой стороны,

модель потенциально может начать использовать наличие черенка как один из признаков класса. Технические требования поэтому предусматривали, чтобы характерные элементы SKU присутствовали как на нормальных, так и на дефектных экземплярах. Это важный принцип построения ML-датасета: Нужно следить не только за тем, что модель должна выучить, но и за случайными признаками, которые она выучить не должна.

Где здесь роль разметчика

На первый взгляд может показаться, что всё описанное относится только к инженерам, которые проектируют съёмку. Но участие команды разметки на раннем этапе даёт ещё одно преимущество. Разметчики и QA заранее видят, какие характеристики объекта в дальнейшем придётся формализовать. Например:

- где заканчивается нормальная неоднородность поверхности и начинается дефект;
- как определить границу между стадиями зрелости;
- как описывать несколько дефектов на одном товаре;
- что делать с частично видимым повреждением;

как трактовать спорные экземпляры.

Если подобные вопросы начинают обсуждаться только после того, как собраны десятки тысяч изображений, может оказаться, что часть необходимой информации во время съёмки вообще не была зафиксирована.

Поэтому для сложных CV-проектов процесс выгоднее строить не как последовательность:

- не как жёсткую последовательность, в которой сбор данных полностью отделён от разметки,

а как совместный цикл:

а как итеративный процесс, где постановка ML-задачи определяет требования к сбору, разметке и QA, а результаты анализа возвращаются в требования следующей итерации.

Главный вывод

Хороший датасет нельзя определить одной цифрой. 1000, 10 000 или 100 000 изображений сами по себе не гарантируют модели ничего. Нужно понимать, какое пространство реальных ситуаций скрывается за этим числом: какие объекты → какие состояния → какие дефекты → какие ракурсы → какое освещение → какие камеры → какие редкие случаи. И только после этого можно отвечать на вопрос, достаточно ли данных для обучения. Но даже правильно собранного датасета недостаточно, если люди по-разному понимают, что именно на этих изображениях нужно размечать.

В следующем проекте US-DATA эта проблема проявилась особенно наглядно: при разметке более 100 тысяч изображений продукции для сети пиццерий пришлось формализовать, где заканчивается нормальная особенность продукта и начинается дефект — например, чем подгоревший край отличается от нарушения формы борта и как трактовать плохо расплавившийся сыр.

Если два разметчика видят объект по-разному: проблема может быть не в людях, а в ТЗ

GUIDELINE

Разногласие между разметчиками может означать проблему в инструкции.

Кейс US-DATA · Разметка пицц

Отрасль	Сеть пиццерий, клиент не раскрывается
Задача	Разметка более 100 000 изображений: маски, полигоны и bounding boxes
Проблема	Неоднозначные признаки дефектов и недостаток пограничных примеров
Решение	Уточнение guideline, ручная модерация и Human-in-the-Loop
Результат	Около 97% по метрике проекта

Одна из самых неприятных ситуаций в разметке данных выглядит так: два исполнителя получают одно и то же изображение и ставят разные метки. Первая реакция обычно простая — кто-то из них ошибся. Но на практике причина часто находится глубже.

Если объект действительно пограничный, инструкция неполная, классы пересекаются или признаки дефекта описаны слишком абстрактно, то оба разметчика могут действовать логично — просто каждый по своей интерпретации.

Тогда проблема не в качестве исполнителя. Проблема в том, что задача недостаточно формализована. И если такую неопределённость не устранить на старте, то при масштабировании команды она только усилится.

Хорошая инструкция должна отвечать не только на вопрос «что размечать»

Она должна отвечать и на более сложные вопросы:

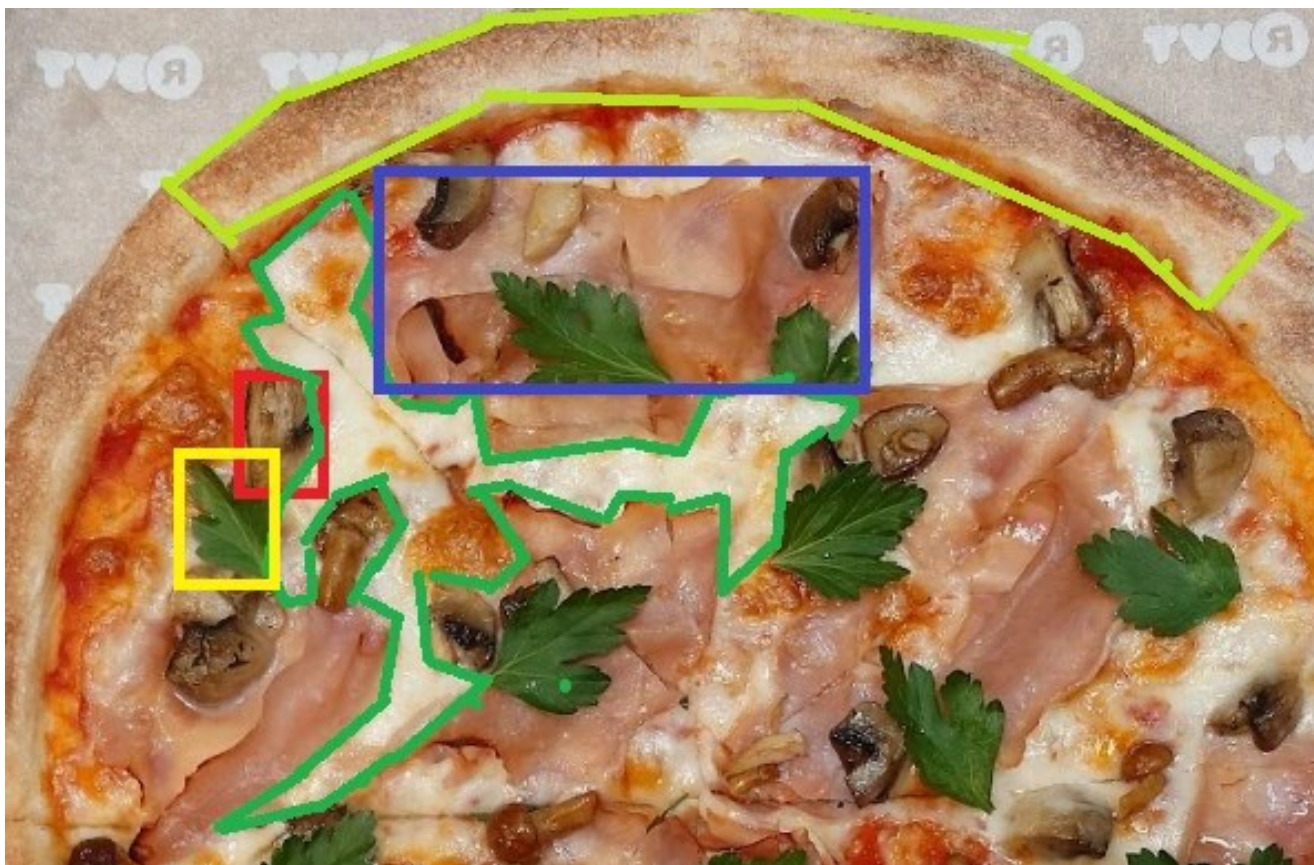
- где начинается и заканчивается объект;

- где проходит граница дефекта;
- что считать нормой;
- что делать с частично видимыми элементами;
- как трактовать смешанные случаи;
- что делать, если объект подходит сразу под несколько классов;

какие случаи отправлять на эскалацию. Чем сложнее визуальная задача, тем реже бывает достаточно формулировки вроде: «Размечайте дефекты пиццы». Для ML это почти бесполезная инструкция.

Нужно определить, что именно считается дефектом, как он выглядит, где проходит его граница и чем он отличается от похожего, но допустимого состояния продукта.

Реальный пример: проект по разметке пицц



Пример разметки пиццы: маски, полигоны и рамки · исходный материал US-DATA

В одном из проектов US-DATA требовалось разметить большой массив изображений продукции сети пиццерий. Объём был значительным — более 100 000 изображений. В рамках задачи использовались разные типы аннотации. Для части объектов требовались маски или полигоны, в том числе для:

- борта пиццы;
- подложки;

разрезов. Для дефектов использовались bounding boxes. Но уже на этапе разбора ТЗ возникли вопросы, на которые невозможно было ответить только по короткому описанию

задачи. Например, в требованиях был дефект: «место, где не полностью расплавился сыр». Но по исходным изображениям не всегда было очевидно, где именно проходит граница такого состояния. Команда прямо запросила дополнительные примеры. Похожая ситуация возникла с дефектом борта.

Нужно было определить место, где тесто «закручено», но на части изображений край был ещё и подгоревшим. Возникал логичный вопрос: это дефект формы, дефект выпечки или сразу несколько признаков?

Такие случаи нельзя качественно решить простым указанием: «Будьте внимательнее». Нужна новая версия правил.

Пограничный случай — это не исключение, а часть задачи

Для production-разметки важны не только идеальные примеры. Именно пограничные ситуации чаще всего определяют итоговое качество модели. Например, если в инструкции есть только два изображения:

- правильно
- и

- неправильно,

но нет промежуточных состояний, разметчики начинают самостоятельно достраивать правила. Один человек может решить: «Подгоревший край — уже дефект». Другой: «Пока форма борта сохранена, это допустимо». Третий: «Если есть одновременно подгорание и деформация, нужно ставить два класса». Если проект включает сотни тысяч изображений, такие различия быстро превращаются в системный шум.

Что происходило в проекте дальше

По мере работы требования уточнялись. Например, отдельно было зафиксировано:

- сыр как начинку не размечать;
 - посторонний предмет в кадре обводить рамкой, но не классифицировать;
- видимый разрез размечать маской или полигоном.

Подобные уточнения могут выглядеть частными, однако на масштабе они определяют однородность ground truth и воспроизводимость решений разных разметчиков.

Но именно такие уточнения и превращают общее описание задачи в рабочий annotation guideline. До уточнения разметчик вынужден интерпретировать изображение. После уточнения он следует единому правилу.

Важный принцип: разметчик не должен угадывать

Если инструкция построена хорошо, исполнителю не приходится каждый раз принимать самостоятельное решение о смысле класса. Его задача — применить заранее согласованное правило. Это особенно важно в больших проектах. Когда работают десятки или сотни annotators, нельзя рассчитывать, что все они будут одинаково «чувствовать» дефект. Стабильность появляется только тогда, когда субъективное решение переводится в формализованное. Например, вместо: «Сыр плохо расплавился» нужно стремиться к правилу вида:

«Дефект отмечается, если на поверхности присутствуют участки X типа, занимающие Y область и визуально отличающиеся от нормального состояния по таким-то признакам».

Не всегда возможно довести правило до математической точности.

Но чем меньше пространства для интерпретации, тем выше согласованность.

Почему примеры важнее длинного текста

Для визуальной разметки хороший guideline почти всегда должен содержать изображения. Причём не только положительные. Нужны как минимум четыре типа примеров:

- явный положительный случай
- явный отрицательный случай
- пограничный случай
- случай, который нужно отправить на эскалацию

Это намного полезнее, чем несколько страниц текста без визуальных примеров. В проекте по пиццам именно недостаток примеров по отдельным дефектам стал причиной дополнительных вопросов.

Human-in-the-Loop для проекта пицц



Неопределённость ТЗ и Human-in-the-Loop · исходный материал US-DATA

Цель — не передать человеку весь поток, а оставить человеку именно те случаи, где его решение даёт модели новую информацию.

Почему просто увеличить команду разметчиков недостаточно

Предположим, инструкция неоднозначна. Можно подключить: 10 человек. 100 человек. 1000 человек. Но если каждый трактует правило немного по-своему, увеличение команды просто увеличит количество противоречивых labels. Поэтому масштабировать нужно не только workforce. Нужно масштабировать:

- правила
- QA
- обратную связь
- эскалацию
- обновление guideline
- контроль версий

Именно это превращает разметку из ручного труда в управляемый производственный процесс.

Три признака плохого ТЗ

Есть простой способ понять, что инструкция требует доработки.

- 1. Одни и те же вопросы повторяются

Если разные annotators независимо задают одинаковый вопрос, почти всегда проблема в guideline.

- 2. QA регулярно исправляет один и тот же тип ошибок

Значит, правило либо непонятно, либо плохо проиллюстрировано.

- 3. После каждого обсуждения появляются новые исключения

Это сигнал, что taxonomu или сама постановка задачи ещё недостаточно зрелая. В таких случаях выгоднее остановить масштабирование на несколько часов или дней и уточнить правила, чем потом переделывать десятки тысяч аннотаций.

Самый дорогой сценарий — исправлять неоднозначность поздно

Представим два варианта.

- Вариант А

На пилоте из 500 изображений выяснилось, что два класса пересекаются. Правила уточнили. Команда продолжила работу.

- Вариант Б

Та же проблема обнаружилась после 100 000 изображений. Теперь нужно:

- найти затронутые данные;
- определить, какие annotations неверны;
- выполнить reannotation;
- повторить QA;
- пересобрать датасет;

возможно, переобучить модель. Сама ошибка одинаковая. Цена её исправления — совершенно разная. Поэтому pilot annotation и ранняя калибровка команды — это не формальность, а способ снизить стоимость ошибок на масштабе.

Главный вывод

Когда два разметчика ставят разные метки, не нужно автоматически считать одного из них плохим исполнителем. Сначала стоит проверить:

- есть ли однозначное правило;
- есть ли визуальные примеры;
- описаны ли edge cases;
- не пересекаются ли классы;

понятно ли, куда эскалировать спорный случай. Хорошая разметка начинается не с большого количества annotators. Она начинается с того, что одна и та же ситуация имеет одно и то же правило для всей команды. И чем сложнее задача, тем важнее заранее предусмотреть, что инструкция будет развиваться вместе с проектом. Следующий логичный вопрос — даже при хорошем guideline и качественной разметке: как понять, какие именно ошибки в датасете действительно опасны для модели и как построить QA так, чтобы находить их до обучения?

ГЛАВА 04 / 10

Контроль качества разметки: что проверять кроме «правильно / неправильно»

01 / Техническая корректность	02 / Полнота	03 / Правильность класса
04 / Соблюдение guideline	05 / Согласованность	06 / Геометрия
07 / Баланс классов	08 / Покрытие срезов	09 / Golden set

Проверка разметки часто выглядит слишком просто: объект отмечен верно — принять; объект отмечен неверно — исправить. Но для ML-проекта этого недостаточно.

Разметка может выглядеть визуально аккуратной и при этом оставаться плохой для обучения модели. Причина в том, что качество датасета складывается не только из правильности отдельной метки, но и из полноты, согласованности, баланса, соблюдения правил и покрытия сложных случаев.

Именно поэтому контроль качества лучше строить не как финальную проверку готового массива, а как отдельный процесс внутри production-разметки.

Первый уровень QA — техническая корректность

Это самый очевидный слой. Нужно проверить, что annotation вообще соответствует формату задачи. Для computer vision это, например:

- bounding box не выходит за границы изображения;
- polygon замкнут;
- mask не имеет разрывов;
- объект не размечен дважды;
- координаты корректны;
- класс существует в справочнике;
- файл аннотации соответствует изображению;

экспорт соответствует требуемому формату. Часть таких ошибок можно проверять автоматически. Например:

пустые annotations; неизвестный class ID; координаты за пределами изображения; нулевая площадь

bounding box;битые ссылки на image ID;отсутствующие файлы.

Но техническая валидность — это только нижний уровень качества.

Второй уровень — полнота разметки

Annotation может быть корректной, но неполной. Например, на изображении находится пять объектов, а размечено четыре. Формально четыре bounding box могут быть построены идеально. Но для обучения object detection пропущенный пятый объект превращается в очень неприятный сигнал. Модель видит объект на изображении, но в ground truth его нет. То есть она получает противоречивое сообщение: здесь визуально присутствует объект этого класса, но считать его объектом не нужно. Поэтому QA должен отвечать не только на вопрос: «То, что размечено, размечено правильно?» но и: «Всё ли, что должно быть размечено, вообще найдено?»

Особенно опасны пропуски редких объектов

Если аннотатор пропускает один объект из тысячи одинаковых коробок, это одна ситуация. Если он пропустил единственный редкий дефект в выборке — влияние может быть намного выше. Поэтому контроль качества лучше привязывать не только к количеству ошибок, но и к критичности класса. Например:

- массовый простой класс — стандартная выборочная проверка;
- редкий дефект — усиленный QA;
- новый класс — двойная проверка;

спорный edge case — экспертная проверка. Именно такой подход позволяет не расходовать одинаковое количество ресурсов на все данные.

Третий уровень — правильность класса

Это как раз хорошо видно на примере с сыром и маслом из первой главы. Геометрия объекта может быть размечена почти идеально, но если class label указан неверно, annotation остаётся плохой. То есть QA геометрии и QA семантики — это разные проверки.

Четвёртый уровень — соблюдение guideline

Даже правильная с точки зрения здравого смысла annotation может быть неправильной относительно правил проекта. Например, в одном проекте принято размечать частично видимый объект только если видно более 50%. В другом — размечается любая видимая часть. Оба подхода имеют право на существование. Ошибка возникает, если разные части датасета размечаются по разным правилам. Поэтому QA должен проверять не абстрактную «правильность», а соответствие annotation текущей версии guideline. Это особенно важно после изменения инструкции.

Что происходит, если правило изменилось посередине проекта

Представим: первые 30 000 изображений размечались по guideline v1.2. Затем выяснилось, что один edge case трактуется неправильно. Появилась v1.3. Возникает вопрос: нужно ли возвращаться к старым данным? Если изменение касается редкой ситуации — возможно, достаточно найти только затронутый subset. Если изменилось базовое правило класса — может потребоваться массовый recheck. Поэтому QA тесно связан с версионированием датасета. Нужно понимать:

- какие данные размечены по какой инструкции;
 - когда правило поменялось;
 - какие annotation потенциально затронуты;
- нужно ли выполнять reannotation.

Пятый уровень — согласованность между разметчиками

Один и тот же guideline может применяться разными людьми немного по-разному. Именно поэтому полезно периодически давать нескольким annotators одинаковую выборку. Затем сравнивать результаты. Если расхождения минимальны — правила работают стабильно. Если расхождения концентрируются в конкретных классах, это сигнал. Причиной может быть:

- плохо описанный класс;
- слабые примеры;
- неоднозначная граница;
- недостаточное обучение команды;

слишком субъективная задача. То есть disagreement — это не только показатель качества людей. Это ещё и диагностический инструмент для самого ТЗ.

Шестой уровень — геометрическая консистентность

Для segmentation и polygon-разметки важно не только выбрать правильный класс, но и одинаково проводить границы. Например, один annotator обводит объект максимально плотно. Другой оставляет вокруг него заметный отступ. Третий игнорирует небольшие выступы. Каждая маска по отдельности может выглядеть «нормально», но вместе они создают разнородный ground truth. Модель начинает учиться на разных правилах геометрии. Для таких задач полезно заранее определить:

- насколько плотно идти по контуру;
- учитывать ли мелкие выступы;
- что делать с тенями;
- включать ли прозрачные части;
- как работать с окклюзией;

где считать границу объекта.

Седьмой уровень — баланс классов

QA датасета — это не только проверка отдельных annotations. После разметки нужно смотреть на распределение данных. Например:

Класс	Количество
Норма	45 000
Дефект А	6 000
Дефект В	1 200
Дефект С	90

Формально датасет может быть размечен идеально. Но для класса С данных может быть недостаточно. Поэтому после QA annotations полезно проводить dataset-level QA. Проверять:

- распределение классов;
- редкие классы;
- слишком частые классы;
- количество изображений без annotation;
- распределение по источникам;

покрытие условий.

Восьмой уровень — проверка срезов

Это особенно важно для production-систем. Общая метрика может быть высокой, но качество сильно различаться между сегментами. Например: день — хорошо;ночь — плохо.

Или: камера А — хорошо;камера В — плохо.

Или: обычная упаковка — хорошо;новый дизайн — плохо.

Поэтому QA датасета должен уметь работать со срезами: класс × источник × камера × освещение × регион × время × состояние объекта. Именно metadata из предыдущей главы позволяют это делать.

- QA — это не одна проверка



Девять уровней контроля качества разметки · исходный материал US-DATA

Качественная разметка — это не только отсутствие явных ошибок.

Девятый уровень — golden set

Для больших проектов полезно иметь небольшой набор эталонных данных. Так называемый golden set. Это изображения, по которым:

- заранее согласованы правильные labels;
- описаны сложные случаи;
- есть экспертное решение;

результат считается эталонным. Golden set можно использовать: для обучения новых annotators; для проверки текущей команды;

для калибровки QA;

для контроля после изменения

guideline. Особенно полезно добавлять туда не только простые примеры, но и edge cases.

QA должен идти во время проекта, а не после него

Одна из самых дорогих ошибок — накопить весь объём и только потом проверять. Представим: команда разметила 100 000 изображений. После этого выясняется, что один класс трактовался неправильно. Теперь нужно искать и переделывать большой кусок массива. Гораздо дешевле построить процесс итеративно:

Например, сначала проверяется пилотная партия примерно из 500 изображений; после корректировки правил объём увеличивается до нескольких тысяч и только затем масштабируется.

Так ошибка обнаруживается до того, как она превращается в десятки тысяч неправильных annotations.

Как может выглядеть многоуровневый QA-процесс

Практически полезная схема:

- 1. Annotator
выполняет разметку.
- 2. Автоматическая проверка
форматы, координаты, class IDs, пустые annotations.
- 3. Reviewer
проверяет выборку или весь объём сложных классов.
- 4. Senior QA
разбирает спорные cases.
- 5. Domain expert
решает семантически сложные вопросы.
- 6. Dataset-level validation
проверяется структура всего набора.

Не весь датасет нужно проверять одинаково

Это ещё один важный принцип. Можно использовать risk-based QA. То есть глубина проверки зависит от риска. Например:

- простые изображения
→ выборочная проверка.
- новый annotator
→ повышенный процент QA.
- новый класс
→ повышенный процент QA.
- редкий дефект
→ почти полный контроль.

- edge cases
- экспертная проверка.
- стабильный массовый класс
- автоматизация + sampling. Такой подход обычно эффективнее, чем пытаться вручную перепроверить абсолютно всё.

Контроль качества должен давать обратную связь

QA без обратной связи превращается в бесконечное исправление одних и тех же ошибок. Если reviewer просто исправляет annotation, но annotator не узнаёт причину, ошибка повторится снова. Поэтому хороший QA-процесс выглядит так:

Эффективный QA должен замыкать обратную связь: найденная ошибка классифицируется по причине, правило при необходимости уточняется, команда получает обратную связь, а изменённый участок проходит повторную проверку.

Например: «Разметчик ошибся» — слабая формулировка. Гораздо полезнее: «В guideline не описано, как работать с объектом при окклюзии более 50%». Тогда исправляется не один annotation, а причина целого класса ошибок.

- Цикл QA

Цель QA — не только исправить ошибку, но и сделать так, чтобы она не повторилась.

Что проверять перед передачей датасета в ML

Перед финальной передачей полезно пройти короткий checklist.

Файлы

- все изображения доступны;
 - нет битых файлов;
- filenames уникальны.

Annotations

- нет пропущенных классов;
 - координаты валидны;
 - masks/polygons корректны;
- class IDs совпадают со справочником.

Семантика

- соблюдается guideline;
 - пограничные случаи разрешены;
- спорные примеры прошли QA.

Dataset

- классы распределены ожидаемо;
 - редкие категории не потеряны;
 - train/validation/test разделены корректно;
- нет очевидного leakage.

Metadata

- заполнены обязательные поля;
 - категории стандартизированы;
- отсутствующие значения понятны.

Самый важный вопрос QA

В конце полезно задать не: «Сколько процентов annotations проверено?» а другой вопрос: «Какие ошибки мы можем пропустить и насколько они опасны для модели?» Например, 99% правильно размеченных простых объектов могут быть менее важны, чем ошибки в 1% редких, но критичных случаев. Поэтому зрелый QA — это управление риском, а не просто подсчёт исправлений.

Главный вывод

Контроль качества разметки нельзя свести к бинарному: правильно / неправильно. Нужно проверять как минимум четыре слоя:

- annotation-level — правильно ли размечен конкретный объект;
- consistency-level — одинаково ли применяются правила;
- dataset-level — сбалансирован и покрыт ли датасет;

production-level — соответствует ли датасет реальным условиям работы модели. И чем раньше этот контроль встроен в процесс, тем меньше вероятность, что ошибка дойдёт до обучения модели или production. Следующий вопрос логично вытекает отсюда: если разметка уже выполнена и датасет оказался проблемным, обязательно ли начинать проект заново? В большинстве случаев — нет.

ГЛАВА 05 / 10

Как исправить уже размеченный датасет, не начиная проект с нуля

RISK-BASED REANNOTATION

Исправлять проблемный сегмент, а не весь массив.

01Локализация	02Аудит	03Причина ошибки
04Reannotation	05Обновление guideline	06Версия датасета
07Дообучение	08Валидация	

Матрица приоритета ошибок

↑ Влияние на модель

Высокое влияние · низкая частота Экспертная проверка	Высокое влияние · высокая частота Исправлять первой
Низкое влияние · низкая частота Отложить	Низкое влияние · высокая частота Автоматизировать

Частота ошибки →

Визуальная схема приоритизации: частота и влияние рассматриваются совместно.

Обнаруженные дефекты в уже размеченном датасете не означают автоматически, что весь массив необходимо собирать и размечать заново. В инженерной практике сначала оценивают локальность проблемы, её происхождение и влияние на модель, а затем выбирают минимальный объём корректирующих работ.

Это одна из самых дорогих ошибок в управлении ML-проектом: обнаружив шум, некорректные классы или неоднородную разметку, команда решает, что существующие данные «испорчены» целиком.

На практике чаще правильнее действовать иначе: не переписывать весь датасет, а найти проблемные сегменты, понять причину ошибки и исправить именно их. Особенно если датасет уже большой, модель обучена, а production даёт достаточно информации о том, где

именно возникают ошибки.

Сначала нужно понять масштаб проблемы

Первый вопрос: ошибка локальная или системная? Например, если модель путает только две похожие категории, нет смысла заново проверять весь датасет. Если ошибка встречается только на новой упаковке — нужно искать именно этот сегмент. Если проблема возникла после изменения guideline — нужно определить, какие части датасета размечались по старой версии инструкции. Но если выясняется, что базовые правила классов были неверными с самого начала, тогда объём rework действительно может оказаться большим. Поэтому исправление датасета начинается не с reannotation, а с аудита причин.

Шаг 1. Найти проблемный срез

Это может быть:

- конкретный класс;
- пара похожих классов;
- определённый дефект;
- одна камера;
- один период;
- новый дизайн упаковки;
- конкретный annotator;
- одна версия guideline;
- один источник данных;

один production-сценарий. То есть вместо: «У нас плохой датасет» нужно получить более точную формулировку: «У нас проблема с классами А и В на изображениях, собранных после такого-то изменения». Чем точнее сформулирован дефект данных, тем меньше объём исправлений.

Шаг 2. Сопоставить ошибки модели с данными

Лучший источник информации о слабых местах датасета — сама модель. Если в production модель регулярно ошибается в одном и том же месте, это повод собрать такие случаи отдельно. Именно так происходило в одном из проектов US-DATA для продуктового ритейла. Клиент видел расхождение между данными AI и фактическими продажами. После этого проблемные видео проверялись вручную, а сложные примеры передавались на доработку.

В результате ошибка модели связывалась с конкретным примером, затем определялась причина, корректировалась разметка и обновлённые данные использовались для следующей итерации обучения.

Это гораздо эффективнее, чем случайно пересматривать весь архив.

Не исправляйте то, что и так работает

Представим: датасет содержит 200 000 изображений. Проблема найдена в двух похожих категориях, которые вместе занимают 12 000 изображений. Из них реальные спорные случаи составляют примерно 2 000. Тогда разумная стратегия:

- не проверять 200 000 файлов,

а:

- отфильтровать 12 000 релевантных;
- найти 2 000 наиболее рискованных;
- провести reannotation;

усилить этот сегмент новыми hard cases. Такой подход можно назвать risk-based reannotation.

Шаг 3. Исправить не только labels, но и правило

Это критически важно. Если просто исправить конкретные annotation, но не устранить причину, ошибка появится снова. Поэтому каждое массовое исправление должно сопровождаться вопросом: Почему эти ошибки вообще возникли? Причина может быть в:

- плохом guideline;
- пересекающихся классах;
- отсутствии примеров;
- новой версии продукта;
- слабом QA;
- неправильной taxonomy;

неоднозначном дефекте. После reannotation guideline должен измениться. Иначе команда просто чинит симптомы.

Шаг 4. Создать новую версию датасета

Исправленный датасет нельзя хранить как:

- dataset_final_final2_new.zip

Для управляемой ML-разработки нужны версии. Например: Dataset v1.0 исходная версия.

Dataset v1.1 исправлены

labels для двух классов. Dataset v1.2

добавлены hard cases. Dataset v2.0 изменилась

taxonomy. Это позволяет понимать:

- на каком датасете обучалась модель;
 - какие изменения были внесены;
 - что именно улучшило качество;
- можно ли воспроизвести предыдущий эксперимент.
- Исправление датасета без полного перезапуска



Исправление датасета без полного перезапуска · исходный материал US-DATA

Исправляем причину и проблемный сегмент, а не весь массив целиком.

Шаг 5. Добавить hard cases

Исправление старых ошибок — только половина работы. Если модель уже показала, где ей сложно, эти случаи нужно превратить в отдельный источник данных. Например:

- необычный ракурс;
- новая упаковка;
- редкий дефект;
- частичное перекрытие;
- плохое освещение;

очень похожий соседний класс. Такие данные особенно ценны, потому что они уже доказали свою сложность.

Почему hard cases важнее случайного расширения

Допустим, модель плохо различает два визуально близких товара. Можно добавить ещё 10 000 обычных изображений. Но если большинство из них простые, качество может почти не измениться. Гораздо полезнее добавить 500–1000 примеров, где различие действительно минимально. То есть при исправлении датасета важно не просто увеличивать объём, а увеличивать плотность полезной информации.

Шаг 6. Проверить, не изменился ли production

Иногда проблема не в старом датасете. Просто environment изменился. Например:

- появилась новая камера;
- поменялось освещение;
- обновилась упаковка;
- изменился фон;
- продукт выглядит иначе;

изменилась география. В таком случае старый датасет может быть вполне качественным — просто он уже не полностью соответствует текущей реальности. Это уже ближе к data drift. И в такой ситуации правильное решение — не «чинить старую разметку», а собрать новый актуальный слой данных.

Хороший датасет не обязан быть вечным

С управленческой точки зрения это означает, что датасет следует рассматривать как версионизируемый актив с ограниченным периодом актуальности, а не как неизменяемый архив.

Иногда команды воспринимают датасет как готовый актив: «Мы один раз собрали данные — теперь используем их несколько лет». Для production ML это редко работает. Особенно в ритейле, промышленности, видеонаблюдении и других задачах, где среда меняется. Датасет должен обновляться вместе с реальностью.

Можно ли исправить датасет без переобучения модели?

Иногда — да, если цель состоит только в подготовке следующей версии данных. Но если исправления затрагивают значимые labels или добавляются новые hard cases, модель обычно нужно переобучить или хотя бы дообучить. Иначе исправленные данные никак не повлияют на production. Корректирующий цикл включает аудит, целевую переразметку, выпуск новой версии датасета, переобучение и последующую валидацию.

Как понять, что reannotation закончена

Не по количеству исправленных файлов. Нужно проверить:

- исчез ли проблемный паттерн;
- улучшились ли метрики именно на нужном срезе;
- не ухудшились ли другие классы;
- появились ли новые ошибки;

сохраняется ли качество на production-подобных данных. Особенно важно сравнивать не только общую метрику. Например:

- до:общая метрика — 97%
- после:общая — 97,4%

На первый взгляд прирост небольшой. Но если проблемный класс вырос с 72% до 94%, это может быть очень значимым результатом.

Не все ошибки одинаково опасны

При исправлении датасета полезно ранжировать дефекты. Например:

- Критичные
- неправильный class label;
- пропущенный объект;
- системное смещение двух классов;

train/test leakage.

- Средние
- небольшая геометрическая неточность;
- неполные metadata;

редкие нарушения guideline.

- Низкий риск

косметические различия, не влияющие на модель. Так можно сформировать приоритет
gework. низкая частота / низкое влияние→ можно отложить

высокая частота / низкое влияние→ автоматизировать

низкая частота / высокое влияние→ экспертная проверка

высокая частота / высокое влияние→ исправлять в первую очередь

Что делать с уже размеченными данными после изменения guideline

Если правило изменилось, необязательно переразмечать всё. Лучше сначала найти затронутые случаи. Например: новое правило касается только:

- окклюзий;
- класса «дефект В»;

фотографий после определённой даты. Тогда можно отфильтровать именно эти данные. Metadata и versioning здесь сильно экономят время.

Когда всё-таки проще начать заново

Такие случаи тоже бывают. Полная переразметка может быть оправдана, если:

- taxonomy полностью изменилась;
- большая часть классов пересекается;
- guideline был принципиально неправильным;
- отсутствует связь между изображениями и annotations;
- dataset provenance потерян;

невозможно понять, по каким правилам размечались разные части. Но это скорее крайний случай.

Итог

Проблемный датасет — не всегда «плохой актив». Часто это просто датасет, в котором нужно локализовать слабые места. Эффективный процесс выглядит так:

Сначала локализуют ошибки и выделяют проблемный сегмент, затем устанавливают причину, исправляют разметку и правила, выпускают новую версию датасета и только после этого переобучают модель и проверяют результат.

Главное — не начинать reannotation вслепую. Чем лучше команда понимает, где именно модель ошибается и почему, тем меньше данных приходится переделывать. Именно поэтому исправление существующего датасета часто обходится быстрее и дешевле, чем запуск нового проекта с нуля. Следующая логичная тема — одна из самых частых причин скрытой деградации: данные были хорошими, но мир изменился.

Data drift: почему модель работала вчера, а сегодня начала ошибаться

PRODUCTION

**Модель не изменилась.
Изменились данные.**

Train	Production
Старая упаковка	Новая упаковка
Камера А	Камера В
Дневной поток	Вечерний
Старые SKU	Новые SKU

Иногда команда делает всё правильно. Датасет собран аккуратно. Разметка проверена. Модель показывает хорошие результаты на train, validation и test. Пилот проходит успешно.

А через несколько месяцев качество начинает снижаться. При этом код модели не менялся. В таких случаях проблема может находиться не в алгоритме и не в старой разметке, а в том, что данные в production стали другими.

В прикладном смысле data drift означает изменение распределения входных данных относительно того, на котором модель обучалась и валидировалась. Сам алгоритм при этом может оставаться неизменным, однако его прежняя оценка качества перестаёт полностью описывать текущую среду эксплуатации.

Проще говоря:

модель продолжает работать по старым правилам мира, а сам мир уже изменился.

Data drift — это не «модель испортилась»

Модель не обязательно стала хуже сама по себе. Она может по-прежнему отлично решать задачу на данных, похожих на те, что использовались при обучении. Проблема возникает, когда реальный поток начинает заметно отличаться от обучающей выборки. Например:

- было: старая упаковка товара;
стало: новый дизайн.
- Или:

- было: одна камера;

стало: новое оборудование.

Или:

- было: дневное освещение;

стало: часть кадров снимается вечером или при другой температуре света.

Для бизнеса это может выглядеть как одна и та же задача. Для модели — уже другой набор входных признаков.

- Хороший пример — смена упаковки в ритейле

В проекте US-DATA для продуктовой сети модель работала с набором товарных категорий и в целом показывала высокий результат. Но упаковки регулярно менялись. В отдельные месяцы происходило от 3 до 25 изменений дизайна. Для покупателя новая упаковка сыра остаётся тем же сыром. Для модели могут измениться:

- цвет;
- расположение логотипа;
- изображение продукта;
- шрифт;
- форма упаковки;
- контрастность;
- фон;

декоративные элементы. Если подобных примеров не было в обучении, модель может начать ошибаться даже на знакомом товаре. Именно поэтому production-проект нельзя считать законченным после одной успешной версии модели.

Кейс US-DATA: когда новая упаковка ломает уже работающую модель

В проекте для продуктовой сети модель уже работала в production и показывала высокий результат по внутренней метрике проекта. Но дизайн упаковок регулярно менялся — в отдельные месяцы происходило от 3 до 25 обновлений.

После таких изменений часть товаров начинала хуже распознаваться. Команда клиента замечала расхождение между данными AI и фактическими продажами через кассы, затем выборочно проверяла видео и передавала проблемные случаи US-DATA на доработку.

Цикл выглядел так: новая упаковка → рост ошибок → ручная проверка → отбор hard cases → доработка → дообучение → повторный запуск.

Вывод: даже качественный датасет может быстро устаревать, если внешний вид объектов в production меняется.

Data drift бывает разным

Не все изменения данных одинаковы. Для практической диагностики полезно разделять несколько сценариев.

- 1. Изменился внешний вид объекта

Например:

- новая упаковка;
- новый дизайн;
- другой цвет товара;
- новая форма;

новый поставщик.

- 2. Изменились условия съёмки

Например:

- новая камера;
- другой объектив;
- другое разрешение;
- новая экспозиция;

другое освещение.

- 3. Изменился состав объектов

Например:

- появились новые SKU;
- изменилось распределение категорий;
- редкий класс стал частым;

часть товаров исчезла.

- 4. Изменился сам бизнес-процесс

Например:

- камера установлена в другом месте;
- сотрудник начинает взаимодействовать с объектом иначе;
- изменилась выкладка;

поменялся маршрут объекта через систему. Все эти изменения могут ухудшить качество модели, даже если архитектура осталась прежней.

Data drift и concept drift — не одно и то же

Эти понятия часто смешивают. Упрощённо: Data drift — изменились входные данные. Например: упаковка стала другой. Concept drift — изменилось само соответствие между признаками и правильным ответом. Например: состояние, которое раньше считалось допустимым, теперь по бизнес-правилам относится к дефекту. То есть в одном случае меняется внешний мир, а в другом — ещё и смысл класса. Для команды разметки это принципиальная разница. При data drift часто достаточно: добавить новые данные и дообучить модель. При concept drift может потребоваться: изменить taxonomy, guideline и labels.

Почему test set не спасает от drift

Test set отвечает на вопрос: «Как модель работает на данных, которые мы заранее выделили для проверки?» Но если сам production изменился, старый test set продолжает проверять старый мир. Представим: модель обучалась и тестировалась на упаковках 2025 года. В 2026 году часть брендов полностью поменяла дизайн. Модель всё ещё может показывать прекрасный результат на старом test. Но это уже не гарантирует качество на текущем production. Отсюда важный вывод: test set тоже должен обновляться вместе с реальностью.

Что нужно мониторить после запуска модели

После deployment полезно отслеживать не только итоговую accuracy или F1. Нужно смотреть на сам поток данных. Например:

- появились ли новые значения классов;
- изменилось ли распределение категорий;
- выросло ли количество low-confidence predictions;
- увеличилось ли число ручных исправлений;
- появились ли новые типы ошибок;
- изменились ли камеры;
- изменилось ли освещение;
- появились ли новые SKU;

выросло ли расхождение с бизнес-метриками.

В проекте с товарами сигналом служили продажи

В кейсе продуктовой сети качество контролировалось не только через техническую метрику. Команда сопоставляла:

- что показала модель
- и что реально прошло через кассы. Если статистика начинала расходиться, это становилось поводом искать причину. Далее:

Расхождение с бизнес-метрикой служило триггером для ручной проверки: проблемные фрагменты выделялись, доразмечались и использовались в следующей итерации обучения.

Это хороший пример того, как drift можно обнаруживать не только ML-метриками, но и через бизнес-данные.

- Как выглядит drift в production

Модель не изменилась. Изменились данные.

Drift редко происходит одновременно

Чаще он накапливается постепенно. Сегодня изменился один товар. Через неделю — второй. Потом заменили камеру. Потом изменилось освещение. Каждое изменение по отдельности может быть небольшим. Но вместе они постепенно уводят production-данные от первоначального train distribution. Из-за этого качество может снижаться медленно, и команда замечает проблему уже поздно.

Поэтому важны срезы по времени

Если есть metadata, полезно сравнивать:

- январь vs март
- старая камера vs новая
- до редизайна vs после
- дневной поток vs вечерний
- SKU version 1 vs SKU version 2

Так можно найти момент, когда началось изменение. Это гораздо полезнее, чем смотреть только одну общую метрику за весь период.

Не каждый drift требует срочного переобучения

Иногда изменение не влияет на качество. Например, фон немного поменялся, но модель осталась устойчивой. Поэтому разумная схема такая:

После обнаружения изменения сначала оценивают его влияние на качество. Если деградация подтверждается, собирают актуальные примеры, размечают их и включают в следующую итерацию обучения.

Не нужно переобучать модель после каждого небольшого изменения среды.

Но некоторые изменения критичны

Особенно опасны:

- новые классы;
- новые типы дефектов;
- резкое изменение камеры;
- полный редизайн упаковки;
- изменение бизнес-правил;

изменение условий, которых вообще не было в train. В таких случаях drift может быстро превратиться в системную ошибку.

Где здесь роль разметки

Разметка становится частью постоянного цикла поддержки модели. После запуска задача уже не выглядит как: «разметить 100 000 изображений». Она выглядит скорее так: «регулярно находить новые сложные случаи и превращать их в обучающие данные». Это принципиально другой режим работы.

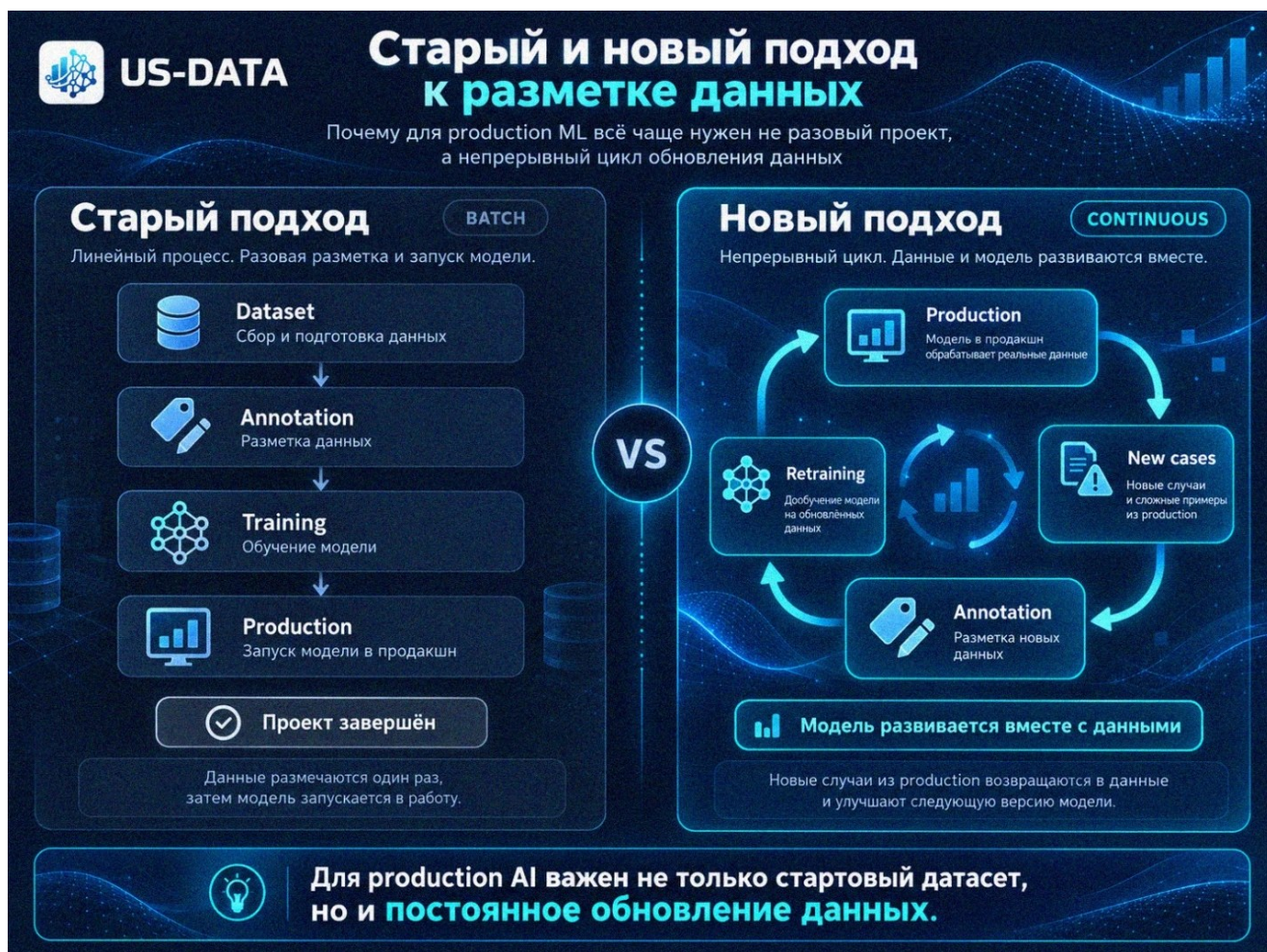
От batch annotation к continuous annotation

Классический подход: собрали большой датасет → разметили → обучили → закончили. Для production ML всё чаще нужен другой подход:

В production-подходе новые ошибки используются для отбора данных, разметки и последующего переобучения, после чего обновлённая модель снова проверяется в эксплуатации.

То есть разметка становится непрерывным процессом.

Старый и новый подход



Старый и новый подход к разметке данных · исходный материал US-DATA

Модель развивается вместе с данными

Drift связан не только с CV

Хотя примеры выше относятся к computer vision, тот же принцип работает и в других модальностях.

- NLP

Меняется язык пользователей, появляются новые термины, продукты, сленг.

- Audio

Меняются микрофоны, акустика, шумы, акценты пользователей.

- Рекомендательные системы

Меняется поведение аудитории.

- Fraud detection

Меняются схемы мошенничества. То есть drift — универсальная проблема production ML.

Кто должен отвечать за drift

Это не только задача ML-инженера. Сигнал может появиться у:

- бизнеса;
- поддержки;
- QA;
- annotators;
- data team;

пользователей. Например, бизнес первым замечает падение конверсии. Оператор — рост ручных исправлений. Annotator — появление нового типа объекта. ML-команда — изменение confidence. Хорошая система должна собирать эти сигналы вместе.

Практический checklist: когда пора обновлять датасет

Стоит проверить необходимость обновления, если:

- модель стала чаще ошибаться на одном классе;
- увеличилось количество low-confidence cases;
- появились новые SKU;
- поменялась упаковка;
- изменились камеры;
- изменилось освещение;
- изменились бизнес-правила;
- появились новые дефекты;
- выросла ручная модерация;

production-метрика расходится с test. Если совпадают несколько пунктов, проблема почти наверняка требует работы с данными.

Главный вывод

Хороший датасет не остаётся актуальным навсегда. Даже если он был идеально подготовлен на момент запуска, production продолжает меняться. Поэтому зрелая ML-система должна уметь: обнаруживать изменения → находить новые hard cases → обновлять датасет → переобучать модель → снова проверять production. Именно здесь заканчивается идея «однократной разметки» и начинается полноценный data lifecycle. Следующий вопрос логичен: как именно находить те самые hard cases, которые дают модели больше всего новой информации?

Hard cases: какие примеры сильнее всего улучшают модель

HARD CASE

Не просто плохое изображение, а пример, значимый для реального production.

Когда модель уже обучена и показывает приемлемое качество, дальнейшее улучшение редко достигается простым увеличением объёма данных. Добавить ещё 50 000 типичных изображений — не всегда значит получить заметный прирост. Гораздо полезнее найти те случаи, на которых модель уже ошибается, сомневается или работает нестабильно. Именно такие примеры обычно называют hard cases. Проще говоря: Hard case — это пример, который несёт для модели больше новой информации, чем очередной “обычный” объект.

Что делает пример сложным

Сложность может возникать по разным причинам. Для computer vision это, например:

- слабое освещение;
- блики;
- размытие;
- окклюзия;
- нестандартный ракурс;
- частично видимый объект;
- мелкий объект;
- похожие классы;
- редкий дефект;
- необычная упаковка;
- нестандартный фон;
- новое оборудование;

комбинация нескольких сложных факторов одновременно. Один и тот же объект может быть простым в одном кадре и сложным в другом. Например: сыр на чистом фоне при хорошем освещении — простой случай. тот же сыр частично перекрыт рукой, снят под углом и похож на упаковку масла — уже hard case.

Почему hard cases ценнее случайной выборки

Допустим, модель уже правильно распознаёт 99% типичных изображений. Если добавить ещё тысячу очень похожих примеров, они лишь подтверждают уже выученный паттерн. Но если добавить несколько сотен случаев, где модель путает два класса, именно они могут изменить decision boundary между категориями. То есть проблема не в том, что данных мало вообще. Проблема в том, что мало данных в нужной области пространства признаков.

Production как источник hard cases

В ритейл-кейсе из первой главы сложные примеры выявлялись по расхождениям между результатами AI и фактическими продажами. После ручной проверки подтверждённые ошибки передавались на доработку. Такой feedback loop превращает реальные ошибки модели в приоритетный источник данных для следующей итерации обучения.

Hard cases можно искать несколькими способами

Есть несколько практических источников.

- 1. Ошибки модели

Самый очевидный вариант. Берём:

- false positives;
- false negatives;
- confusion между похожими классами;
- ошибки сегментации;

пропущенные объекты. Это уже готовый пул сложных данных.

- 2. Low-confidence predictions

Даже если модель формально дала правильный ответ, низкая уверенность может быть сигналом. Например:

- class A — 0.51
- class B — 0.47

Модель практически не различает варианты. Такие случаи полезно отправлять на ручную проверку.

- 3. Расхождения с бизнес-данными

Как в ритейл-кейсе: AI говорит одно → бизнес-система показывает другое. Источником проверки могут быть:

- продажи;
- складской учёт;
- ручная инспекция;
- CRM;
- производственные данные;

операторские решения. Это особенно ценно, потому что позволяет связывать ошибки модели не только с ML-метрикой, но и с реальным процессом.

- 4. Ручная модерация

Иногда *hard cases* автоматически появляются в процессе работы. Например, если модель не уверена — изображение передаётся человеку. Именно так был устроен цикл в проекте по пиццам. Система обрабатывала поток изображений, а случаи, которые AI не смог уверенно определить, поступали на ручную модерацию. После этого сложные примеры можно было возвращать в обучающий датасет.

Это и есть основа *active learning*

В классическом сценарии данные для разметки выбираются случайно.

В *active learning* отбор данных становится управляемым: для разметки приоритетно выбираются примеры, по которым модель наиболее неуверенна, которые представляют редкие области распределения или способны уточнить границу между классами.

размечать прежде всего те примеры, которые потенциально дадут модели больше всего новой информации. Это могут быть:

- *uncertain samples*;
- редкие классы;
- новые кластеры;
- ошибки;
- *outliers*;

пограничные случаи. То есть *annotation budget* расходуется не равномерно. Он концентрируется там, где эффект от каждой новой метки выше.

Но высокая неопределённость не всегда означает полезный пример

Здесь требуется дополнительная фильтрация: неопределённость модели сама по себе ещё не доказывает, что пример полезен для обучения. Если модель не уверена, причиной может быть:

- действительно сложный объект;
- плохое качество изображения;
- мусорный кадр;
- неправильный источник;

объект вообще вне задачи. Например: полностью смазанный кадр может иметь низкий *confidence*. Но разметка такого изображения не обязательно улучшит модель, если подобные данные вообще не должны попадать в *production*. Поэтому *hard case mining* должен включать ещё один вопрос: Этот случай важен для будущей работы модели или это просто шум?

Hard cases нужно группировать, а не собирать в одну папку

Если все сложные примеры складывать вместе, быстро получится хаотичный набор. Полезнее создавать категории. Например:

- similar classes
- low light
- occlusion
- new packaging
- rare defect
- small object
- motion blur
- new camera

Так можно понять, какая именно проблема наиболее частая. А затем решать её системно.



Карта hard cases · исходный материал US-DATA

Сложный пример — это не просто “плохое изображение”, а пример, важный для реального production.

Как понять, что **hard case** действительно полезен

Я бы использовал три критерия.

- 1. Модель на нём нестабильна

Например:

- низкий confidence;
- разные версии модели дают разные ответы;

часто возникает confusion.

- 2. Такой случай реально встречается

Не лабораторная экзотика, а production-сценарий.

- 3. Ошибка имеет цену

Например:

- влияет на бизнес;
- вызывает ручную проверку;
- влияет на безопасность;
- портит качество отчёта;

создаёт ложное решение. Если совпадают все три пункта — это приоритетный **hard case**.

Не все **hard cases** нужно размечать одинаково

Можно ввести приоритизацию.

Например:

Приоритет А частые и критичные ошибки. → размечать в первую очередь.	Приоритет В редкие, но дорогие ошибки. → экспертная проверка.	Приоритет С частые, но малозначимые. → можно автоматизировать или отложить.	Приоритет D редкие и мало влияющие. → низкий приоритет. Это позволяет не перегружать команду.
--	---	---	---

Hard cases помогают не только модели, но и **guideline**

Сложные примеры часто выявляют слабые места инструкции. Например: модель путает два класса. Команда смотрит данные. Оказывается, и **annotators** трактуют границу между ними по-разному. Тогда проблема уже не только в модели. Нужно:

В такой ситуации сначала уточняют **guideline**, пересматривают затронутые **labels** и добавляют эталонные примеры; дообучение имеет смысл проводить уже на согласованной версии данных.

То есть **hard case** может показать дефект всей цепочки.

В проекте по пиццам сложные случаи были частью постоянного цикла

В проекте по пиццам этот принцип был встроен в рабочий цикл: сложные изображения после пилота возвращались в ручную разметку и затем в обновлённый датасет. Итерации продолжались до достижения примерно 97% по метрике проекта; важен здесь не сам показатель, а механизм систематического возврата ошибок в данные.

Ошибки модели можно превращать в очередь на разметку

Практически это можно организовать как управляемую очередь: результаты inference проходят фильтрацию, релевантные сложные случаи попадают в пул на разметку и QA, после чего используются в следующей итерации обучения.

Ключевой элемент здесь — фильтрация. Не все ошибки идут на разметку. Сначала нужно убрать:

- дубликаты;
- мусор;
- нерелевантные случаи;
- технические сбои;

уже известные паттерны. И только потом отправлять человеку.

Дубликаты особенно опасны

Представим, что одна и та же сложная ситуация встречается на видео 500 раз подряд. Если все 500 кадров отправить в разметку, бюджет будет потрачен неэффективно. Поэтому полезно:

- кластеризовать похожие случаи;
- удалять near-duplicates;

выбирать репрезентативные примеры. Так hard-case dataset остаётся разнообразным.

Лучше собирать разнообразие внутри ошибки

Если модель ошибается на новом дизайне упаковки, не нужно размечать только один товар в одном положении. Полезнее собрать:

- разные экземпляры;
- разные камеры;
- разные углы;
- разные условия света;

разные соседние объекты. То есть даже hard cases должны иметь внутреннее покрытие.

Где заканчивается hard case и начинается out-of-distribution

Иногда модель получает данные, которых вообще не было в постановке задачи. Например, система должна распознавать продукты, а в кадре появляется объект, которого нет в taxonomu. Это уже может быть не просто сложный случай, а out-of-distribution sample. Такие данные полезно выделять отдельно. Потому что решение может быть разным:

- добавить новый класс;
 - отправить в unknown;
 - исключить из задачи;
- изменить бизнес-логику.

Hard cases — это мост между production и данными

Именно здесь становится видно, почему современная работа с датасетами всё меньше похожа на одnorазовую разметку. Production постоянно создаёт новые примеры. Модель показывает, где ей сложно. Эти случаи возвращаются в данные. Датасет обновляется. Модель улучшается. Поэтому лучший источник следующей версии датасета часто находится уже не в исходном архиве, а в ошибках текущей модели.

Практический чек-лист hard case mining

Если модель уже работает, полезно регулярно собирать:

- false positives;
- false negatives;
- low-confidence predictions;
- confusion между похожими классами;
- новые категории;
- новые упаковки;
- новые камеры;
- случаи ручной модерации;
- расхождения с бизнес-данными;

редкие, но критичные ошибки. После этого:

Главный вывод

Для зрелой модели качество улучшается не за счёт бесконечного роста датасета. Оно улучшается за счёт более полезных данных. Именно *hard cases* позволяют направить ресурсы туда, где модель реально слабее всего. Поэтому вопрос: «Сколько ещё изображений нам нужно?» часто менее полезен, чем: «Какие именно примеры модель сейчас понимает хуже всего?» Следующая глава уже будет не столько про данные, сколько про организацию процесса: кто должен принимать решения по качеству, кто отвечает за спорные случаи и почему хороший ML-датасет — это совместная работа нескольких ролей.

ГЛАВА 08 / 10

Кто отвечает за качество данных: ML, QA, разметчики и domain expert

Business**Что требуется системе****ML****Что должна предсказывать модель****Domain Expert****Что правильно по смыслу****PM****Как синхронизируются изменения****Annotator****Как правило применяется к данным****QA****Насколько стабильно оно применяется**

Когда модель начинает ошибаться, ответственность часто пытаются найти в одном месте. Разметчик поставил неправильный label. QA пропустил ошибку.

ML-инженер выбрал неудачную архитектуру.

Заказчик недостаточно подробно описал задачу.

Но в реальном ML-проекте качество датасета почти никогда не принадлежит одному человеку или одной команде. Оно возникает на стыке нескольких ролей: Если хотя бы одна связь между ними работает плохо, ошибка может пройти дальше по pipeline и проявиться уже после запуска модели. Поэтому хороший процесс начинается не с вопроса: «Кто виноват в ошибке?» а с вопроса: «На каком этапе должно было быть принято решение и кто отвечает за этот тип решения?»

Разметчик не должен самостоятельно определять смысл класса

У исполнителя есть важная, но ограниченная зона ответственности. Разметчик должен:

- соблюдать guideline;
- правильно применять утверждённые классы;
- соблюдать геометрию разметки;
- замечать неоднозначные случаи;

отправлять спорные примеры на эскалацию. Но разметчик не должен сам решать, что бизнес считает дефектом. Например, в проекте по пиццам возникали вопросы:

- где именно считать сыр недостаточно расплавленным;
- является ли подгоревший край тем же дефектом, что и закрученное тесто;
- нужно ли размечать посторонний предмет;

нужно ли отдельно отмечать ингредиенты. Это не вопросы аккуратности annotator. Это вопросы постановки задачи. Если каждый разметчик отвечает на них самостоятельно, команда получает разные labels при одинаковых данных.

Роль ML-команды: определить, чему вообще должна научиться модель

ML-инженер или data scientist должен понимать:

- какой prediction нужен системе;
- какие классы реально различимы;
- какая гранулярность разметки необходима;
- какие ошибки критичны;
- какую метрику оптимизирует проект;

какие данные нужны для обучения и оценки. Например, технически можно создать двадцать очень похожих классов дефектов. Но если визуально они почти неразличимы даже для человека, хорошей taxonomu это не станет. Поэтому ещё до масштабной разметки нужно проверить: Можно ли вообще устойчиво отличить эти классы по тем данным, которые увидит модель? Если нет, проблема не решится наймом дополнительных разметчиков.

ML-команда также должна возвращать ошибки обратно в данные

После обучения роль ML-инженеров не заканчивается. Именно модель может показать, где датасет слаб. Например:

- confusion между двумя классами;
- низкий recall редкого объекта;
- ошибки только ночью;

- падение качества после новой камеры;

увеличение low-confidence predictions. Эти сигналы должны попадать обратно к data/annotation team. Иначе разметка развивается отдельно от модели, а модель — отдельно от данных.

Domain expert: человек, который знает, что считается правильным в реальном мире

Чем специализированнее предметная область, тем важнее эта роль. Domain expert может быть:

- технологом на производстве;
- врачом;
- товароведом;
- оператором оборудования;
- специалистом контроля качества;

сотрудником клиента, хорошо знающим бизнес-процесс. Он отвечает не за polygon и не за формат COCO. Он отвечает за смысл. Например: Этот объект действительно является дефектом? Эта степень зрелости ещё допустима? Этот вид повреждения требует списания? Эти две категории нужно различать или для бизнеса они эквивалентны? Без такого специалиста annotation team может идеально реализовать технически неверную постановку.

Иногда expert нужен только на 1% данных

Это важный момент для экономики проекта. Необязательно отправлять domain expert все изображения. Обычно его участие особенно ценно для:

- новых классов;
- спорных edge cases;
- конфликтов между разметчиками;
- golden set;

изменений guideline. То есть workflow может выглядеть так:

- 90–95% понятных случаев

→ обычная разметка.

- несколько процентов сложных

→ Senior QA.

- самые неоднозначные

→ domain expert. Так дорогой экспертный ресурс используется только там, где действительно нужен.

Роль QA: не «исправлять за разметчиками», а поддерживать стабильность системы

Хороший reviewer не просто ищет ошибки. Он должен понимать:

- какие ошибки повторяются;
- почему они повторяются;
- какие правила вызывают вопросы;
- какие annotators требуют калибровки;
- какие классы требуют усиленного контроля;

где нужно обновить guideline. Например, если десять разных разметчиков независимо ошибаются одинаково, вероятность того, что проблема только в людях, невелика. Скорее всего: правило плохо сформулировано. И тогда задача QA — не исправить десять annotations. Задача QA — устранить источник следующей тысячи ошибок.

Роль PM: соединить всех участников процесса

В больших data annotation проектах PM — это не просто человек, который следит за сроками. Он часто становится центром информационного обмена. Через него проходят:

- изменения требований;
- новые версии guideline;
- вопросы annotators;
- ответы клиента;
- результаты QA;
- изменения taxonomu;

новые приоритеты ML-команды. Если эта коммуникация не формализована, быстро возникает несколько версий истины. Например: клиент уточнил правило в чате. Senior annotator узнал об этом. Часть команды — нет. Через неделю половина новых данных размечена по старому правилу, половина — по новому. Поэтому каждое значимое решение должно переходить: Значимое решение должно быть перенесено из переписки в актуальную версию guideline, зафиксировано в истории изменений и доведено до всей команды.

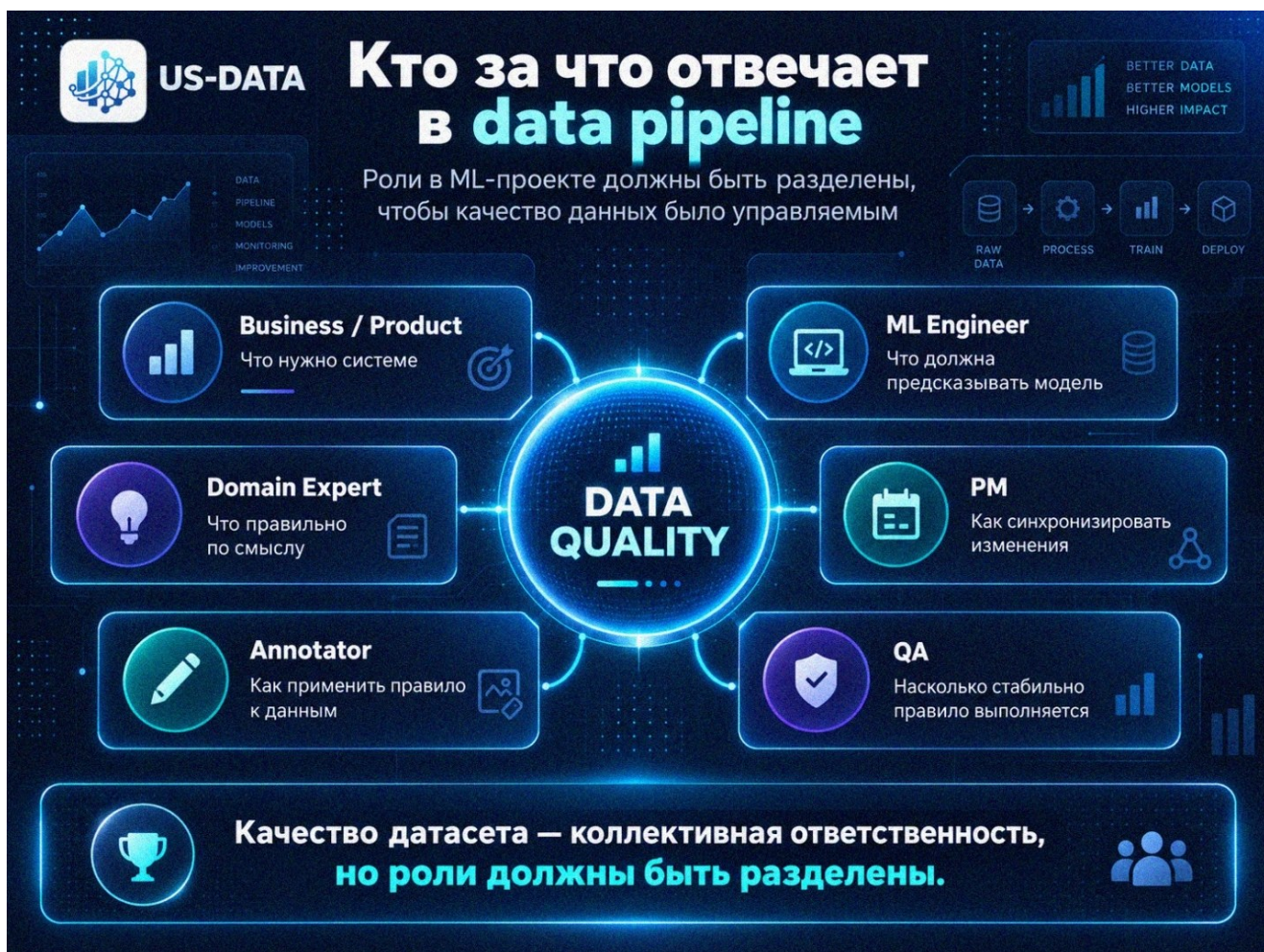
Кто должен менять guideline

Не один человек. Лучше, когда изменение проходит короткий цикл согласования. Например:

01Annotator / QA	02Обнаружен спорный случай	03PM
04ML / Domain expert / клиент	05Принято решение	06Guideline обновлён
07Команда уведомлена		

Это особенно важно, если изменение влияет на уже размеченные данные. Тогда нужно ещё определить: Нужно ли пересматривать старый dataset?

- Кто за что отвечает в data pipeline



Распределение ответственности в data pipeline · исходный материал US-DATA

Качество датасета — коллективная ответственность, но роли должны быть разделены.

Почему нельзя смешивать все роли

Иногда один человек пытается выполнять всё сразу. Например, annotator одновременно:

- размечает;
- трактует неоднозначные требования;
- решает спорные случаи;

проверяет себя. Это опасная схема. Причина проста: человек редко замечает собственную устойчивую интерпретацию как ошибку. Если он неправильно понял правило, он может очень последовательно размечать тысячи объектов одинаково неправильно. Поэтому нужны независимые уровни контроля.

Хороший процесс строится на разделении решений

Можно условно разделить решения на четыре уровня.

- Уровень 1. Бизнес

Что является правильным результатом?

- Уровень 2. ML

Как превратить бизнес-задачу в классы и данные?

- Уровень 3. Annotation

Как применить эти правила к конкретному объекту?

- Уровень 4. QA

Насколько стабильно и правильно правила применяются на масштабе? Так гораздо легче понимать, где возникла проблема.

Golden set должен утверждаться не только QA

Мы уже упоминали golden set — эталонную выборку. Но его полезность сильно зависит от того, кто его подготовил. Если golden set создаётся только annotators, он отражает их интерпретацию. Сильнее схема: annotation team + QA + domain expert / клиент + ML. Тогда эталон одновременно проверяется:

- визуально;
- семантически;
- технически;

с точки зрения задачи модели.

На старте проекта полезна калибровка команды

Перед масштабной разметкой стоит провести небольшую совместную итерацию. Например: 100–500 изображений. Несколько annotators выполняют задачу независимо. После этого команда смотрит:

- где совпали;
- где разошлись;
- какие вопросы появились;
- какие классы нестабильны;

какие примеры нужно добавить в guideline. И только после этого начинается большой объём. Это намного дешевле, чем впервые увидеть системное расхождение после 50 000 annotations.

Что происходит после изменения задачи

ML-проект редко остаётся неизменным. Может появиться:

- новый класс;
- новый дефект;
- новый SKU;
- новая камера;

новый бизнес-критерий. Любое такое изменение должно запускать управляемый процесс. Например:

01Новое требование	02Анализ ML-команды	03Проверка domain expert
04Новые примеры	05Обновление guideline	06Калибровка команды
07Разметка	08QA	

Иначе новое правило просто «просачивается» в проект через переписку и устные договорённости.

Версионировать нужно не только датасет

Нужны как минимум три связанные версии:

- Dataset version
- Guideline version
- Model version

Например, версия guideline 1.4 может быть связана с Dataset 2.1 и Model 3.7; такая связь позволяет воспроизвести условия, в которых была получена конкретная модель.

Тогда через несколько месяцев можно понять: Почему именно эта модель давала такой результат? Без этой связи расследовать старые ошибки значительно сложнее.

Управленческий риск: знания находятся в голове одного человека

Одна из наиболее опасных ситуаций: «Спросите Машу, она знает, как размечать сложные случаи». Пока Маша работает в проекте — всё нормально. Если она уходит или переключается на другую задачу, вместе с ней исчезает часть правил. Поэтому знание должно переноситься: Критические знания необходимо переводить из индивидуальной экспертизы в документацию, визуальные примеры и формализованный workflow. Это одна из причин, почему guideline является не просто инструкцией для новичков, а частью инфраструктуры проекта.

Производительность и качество нельзя управлять отдельно

Разметка часто измеряется через:

- изображений в час;
- объектов в день;

стоимость annotation. Но увеличение скорости может ухудшать качество. Например, если норматив слишком жёсткий, annotators начинают:

- пропускать сложные случаи;
- реже эскалировать вопросы;
- принимать решения «на глаз»;

сокращать время на проверку. Поэтому операционные KPI должны включать не только throughput. Нужны одновременно: Поэтому операционные KPI полезно оценивать в комплексе: скорость, качество, количество возвратов, согласованность разметки и долю спорных случаев.

Чем сложнее задача, тем важнее feedback loop

У зрелого процесса информация постоянно движется в обе стороны. Не только: Коммуникация не должна быть односторонней — только от клиента к разметчикам. но и: Наблюдения разметчиков должны через QA и PM возвращаться клиенту и ML-команде как структурированная обратная связь. Потому что annotators ежедневно видят тысячи примеров. Именно они часто первыми замечают:

- новый тип объекта;
- неожиданный дефект;
- конфликт правил;

новую вариацию данных. То есть разметчик — не только «руки». Он ещё и датчик проблем в dataset.

- Пример с пиццами хорошо показывает эту логику

Кейс с пиццами показывает организационную сторону этого процесса: вопросы по дефектам и объектам разметки не решались отдельными исполнителями «на глаз», а превращались в согласованные правила. После уточнения спорного случая решение фиксировалось в guideline и становилось единым для всей команды.

Когда процесс масштабируется правильно

Хороший annotation pipeline должен выдерживать рост: 10 разметчиков → 50 → 200 без пропорционального роста хаоса. Для этого нужны:

- единый guideline;
- обучение;
- golden set;
- QA;
- версия правил;
- система эскалации;
- обратная связь;

контроль изменений. Без этой инфраструктуры увеличение команды может увеличивать объём быстрее, чем качество.

Главный вывод

Качество данных нельзя полностью «отдать на аутсорс» annotators и нельзя полностью переложить на ML-команду. Каждая роль отвечает за свой слой:

- бизнес — за цель;
- ML — за формализацию задачи;
- domain expert — за смысл;
- PM — за согласованность процесса;
- annotator — за применение правил;

QA — за стабильность результата.

Именно взаимодействие этих ролей определяет, станет ли разметка просто большим массивом labels или управляемым датасетом, пригодным для production ML.

Следующий уровень — уже не столько про качество самой разметки, сколько про управление рисками данных: откуда они получены, можно ли их использовать, какие персональные или чувствительные данные находятся внутри и что нужно документировать.

Правовые и управленческие риски датасета: provenance, персональные данные и governance

Источник · Правовое основание · Персональные данные · Доступ · Версии · История разметки · Срок хранения

Даже технически хороший датасет может оказаться проблемным, если команда не знает: откуда пришли данные; на каком основании их можно использовать; содержат ли они персональные данные; кто имел к ним доступ; по каким правилам они очищались, размечались и обновлялись.

Для ML-проекта это уже не вопрос «юристов где-то рядом». Это часть качества самого датасета. Если происхождение данных непрозрачно, невозможно нормально оценить ни юридические риски, ни bias, ни пригодность набора для production. Поэтому современный data governance начинается с простого вопроса:

Можем ли мы объяснить происхождение каждого значимого слоя данных и историю его изменений?

Provenance: у датасета должна быть биография

Под data provenance обычно понимают происхождение данных и историю того, что с ними происходило. Для проекта полезно знать как минимум:

- источник данных;
- период сбора;
- кто выполнял сбор;
- при каких условиях;
- какая версия исходников использовалась;
- были ли данные очищены;
- выполнялась ли анонимизация;
- кто размечал;
- по какой версии guideline;
- какие преобразования применялись;

в какую версию датасета вошёл объект. По сути, provenance отвечает на вопрос: «Как этот файл оказался в модели именно в таком виде?» Без этого расследование ошибок становится

намного сложнее.

Почему происхождение данных влияет не только на право, но и на качество

Представим два датасета. В первом изображения:

- получены с production-камер;
- собраны в нескольких магазинах;
- имеют известные даты;
- привязаны к типу камеры;

прошли одинаковый pipeline подготовки. Во втором:

- часть скачана из интернета;
- часть получена от клиента;
- часть собрана вручную;

происхождение некоторых файлов неизвестно. Даже если число изображений одинаковое, управляемость этих наборов совершенно разная. В первом можно построить срезы и понять причину ошибки. Во втором происхождение само становится неизвестной переменной.

Для high-risk AI требования к data governance уже формализуются

В EC AI Act прямо связывает качество high-risk AI systems с тем, как организованы training, validation и test datasets. В статье 10 среди элементов data governance перечислены происхождение и сбор данных, annotation и labelling, cleaning, updating, enrichment, оценка пригодности и количества данных, выявление возможных biases и работа с data gaps.

То есть подход постепенно меняется: раньше достаточно было сказать: «У нас хороший датасет». Теперь всё важнее уметь объяснить: почему он хороший, откуда он взялся и как это контролируется.

Персональные данные могут находиться там, где их не ожидали

Не каждый ML-проект работает с персональными данными. Но они могут появиться в исходниках неожиданно. Например, в computer vision:

- лица;
- автомобильные номера;
- бейджи сотрудников;
- экраны;
- документы в кадре;

адреса и таблички. В аудио:

- голос;
- имя;
- номер телефона;

содержание разговора. В текстах:

- email;
- ФИО;
- адрес;

идентификаторы клиентов. Поэтому перед передачей данных в annotation pipeline полезно сначала определить: есть ли там вообще персональные или чувствительные данные.

«Мы используем данные только для обучения» — недостаточное объяснение

Если в датасете есть персональные данные, важно понимать:

- для какой цели они были собраны;
- на каком основании обрабатываются;
- действительно ли весь объём нужен;
- сколько времени нужно их хранить;
- кому они передаются;

можно ли уменьшить идентифицируемость.

В европейском контексте GDPR строится вокруг принципов lawfulness, fairness, transparency, purpose limitation, data minimisation, accuracy, storage limitation, integrity/confidentiality и accountability.

Для ML-команды здесь особенно практичны три идеи:

не собирать больше, чем нужно; не хранить дольше, чем нужно;
не давать доступ большему числу людей, чем нужно.

Анонимизация и псевдонимизация — не одно и то же

Это полезно различать.

Псевдонимизация уменьшает прямую идентифицируемость, но связь с человеком потенциально может быть восстановлена при наличии дополнительной информации.

Анонимизация предполагает значительно более сильный результат: данные больше нельзя разумными средствами связать с конкретным человеком.

EDPB отдельно подчёркивает, что оценка анонимности AI-моделей не должна быть формальной: нужно учитывать возможность извлечения информации о training data и связи output с субъектами данных. [Европейский комитет по защите данных](#)

Для статьи достаточно практического вывода:

Удалить ФИО из CSV ещё не всегда значит сделать данные анонимными.

Что можно минимизировать ещё до разметки

Если идентификация человека не нужна для ML-задачи, полезно рассмотреть:

- blur лиц;
- blur номеров;
- crop лишних областей;
- удаление metadata;
- замену ID;
- удаление ненужных текстовых полей;

доступ annotators только к необходимому фрагменту данных. Это одновременно снижает: юридический риск; объём чувствительной информации; последствия возможной утечки.

- Data governance вокруг датасета



Data governance вокруг датасета · исходный материал US-DATA

Хороший датасет должен быть не только точным, но и прослеживаемым.

Контроль доступа — часть data pipeline

Если проект чувствительный, вопрос «кто видит данные» должен решаться системно.

Например: Annotator видит только назначенную задачу.

QA видит рабочий сегмент.

PM управляет проектом, но не обязательно получает полный доступ ко всем исходникам.

ML team работает с финальным набором.

Это принцип least privilege — давать ровно тот доступ, который нужен человеку для выполнения своей роли.

Обезличивание не отменяет организационные меры

Даже если данные частично обезличены, нужны:

- разграничение доступа;
- журналирование;
- защищённая передача;
- правила хранения;
- удаление временных файлов;
- контроль выгрузок;

NDA и договорные ограничения при необходимости. То есть безопасность нельзя решить одной кнопкой «blur faces».

Нужно документировать не только данные, но и решения

Важный governance-слой — история решений. Например: Почему появился новый класс? Почему изменили границу дефекта? Почему этот source больше не используется? Почему переехали с guideline 1.3 на 1.4? Если такие решения существуют только в чатах, через полгода восстановить логику проекта будет сложно. Поэтому полезно вести простой change log.

- Пример минимального change log

Версия	Изменение	Причина	Что затронуто
v1.2	Добавлен новый класс	Новый production case	4 500 изображений
v1.3	Уточнено правило окклюзии	Низкий agreement	класс A
v2.0	Изменена taxonomy	Новая бизнес-логика	весь dataset

Это одновременно помогает:

ML;QA;
PM;
аудиту;
воспроизводимости.

Governance нужен не только крупным корпорациям

Можно подумать, что provenance и документация актуальны только для банков, медицины или high-risk AI. Но даже небольшой ритейл-проект быстро сталкивается с теми же проблемами. Например: через восемь месяцев модель начинает ошибаться. Команда открывает старый dataset. И выясняется:

- неизвестно, откуда часть изображений;
- непонятно, по какой инструкции они размечались;
- guideline несколько раз менялся;

старые и новые labels лежат вместе. Это уже не юридическая проблема. Это обычная техническая проблема воспроизводимости.

Юридический риск и ML-риск часто связаны

Например, неизвестное происхождение данных означает сразу две вещи.

- Юридически

непонятно, можно ли их использовать.

- Технически

непонятно, насколько они репрезентативны. Или:

- Юридически

нужно минимизировать персональные данные.

- Технически

удаление ненужных признаков иногда помогает снизить риск случайных shortcuts. Поэтому governance нельзя рассматривать отдельно от ML quality.

Нужно ли хранить всё навсегда

Сохранять все промежуточные данные бессрочно не требуется: сроки хранения должны вытекать из назначения данных, требований безопасности и применимых правовых оснований.

Иногда команды сохраняют:

- все исходные файлы;
- все промежуточные exports;
- каждую временную версию;

старые персональные данные. «На всякий случай». Но хороший lifecycle должен определять: что хранится;зачем;

сколько;

кто имеет доступ;когда удаляется.

В GDPR storage limitation прямо относится к базовым принципам обработки персональных данных.

Отдельный риск — передача данных подрядчику

Если разметка выполняется внешней командой, до старта проекта полезно определить:

- какие данные передаются;
- что нужно обезличить;
- где происходит обработка;
- кто получает доступ;
- можно ли скачивать исходники;
- как удаляются данные после проекта;

какие требования фиксируются договором. Это лучше решать до загрузки первого архива, а не после.

Dataset card как практический инструмент

Для сложных проектов можно завести короткую карточку датасета. Не обязательно огромный документ. Например:

ПРИМЕР КАРТОЧКИ ДАТАСЕТА

Название	Retail Products Dataset v2.4
Назначение	Распознавание SKU в production.
Источник	Видео с камер магазинов.
Период	Январь–июнь.
Классы	10 SKU.
Ограничения	Не покрывает упаковки после июля.
Разметка	Guideline v1.7.
QA	Double review для hard cases.
Особые данные	Лица покупателей исключаются / обезличиваются.
Split	Train / validation / test.

Такой документ сильно упрощает жизнь через несколько месяцев.

Чем зрелее AI, тем важнее документация данных

Это хорошо видно и в AI Act: для high-risk systems data governance рассматривается не как факультативная документация, а как часть требований к training, validation и test data. Данные должны быть релевантными, достаточно репрезентативными и учитывать контекст, в котором система будет использоваться.

Это совпадает с тем, что мы уже увидели технически в предыдущих главах. Хороший датасет должен отвечать сразу на два вопроса: что внутри? и почему именно это внутри?

Практический checklist перед запуском разметки

До передачи данных annotators я бы проверил:

- Происхождение
- откуда данные;
- кто их собрал;

можно ли подтвердить источник.

- Назначение
- зачем они используются;

соответствует ли dataset будущей задаче.

- Персональные данные
 - присутствуют ли они;
- действительно ли необходимы.

- Минимизация
- что можно удалить или скрыть заранее.

- Доступ
- кому нужны исходники.

- Версии
- как связаны source, guideline и dataset.

- Хранение
- когда временные данные удаляются.

Практика в России

Для российских ML-проектов базовая рамка — Федеральный закон № 152-ФЗ «О персональных данных». Если в датасете есть лица, голоса, номера автомобилей, документы, контактные данные или другие идентификаторы, ещё до передачи массива на разметку нужно определить правовое основание обработки, круг лиц с доступом и необходимость обезличивания.

С 1 сентября 2025 года в России действуют новые требования и методы обезличивания персональных данных. Роскомнадзор утвердил отдельные требования к обезличиванию приказом № 140 от 19 июня 2025 года, а Правительство РФ — дополнительные правила и методы постановлением № 1154 от 1 августа 2025 года.

Для ML-команды практический вывод простой: нельзя считать данные обезличенными только потому, что из таблицы удалили ФИО. Нужно оценивать весь набор признаков и возможность повторной идентификации. Если идентифицирующие признаки модели не нужны, безопаснее минимизировать их ещё до разметки: скрывать лица и номера, удалять ненужные metadata, ограничивать доступ к исходникам и фиксировать правила хранения.

Отдельно стоит документировать, какие данные были переданы подрядчику, кто получил к ним доступ, какие преобразования выполнялись и когда рабочие копии должны быть удалены. Для production ML это одновременно и правовой, и технический контроль provenance.

Главный вывод

Качество датасета — это не только accuracy labels. Зрелый dataset должен быть: точным; репрезентативным;

версионизируемым; прослеживаемым;
защищённым;

понятным с точки зрения происхождения и правил использования.

Чем ближе ML-система к production и чем выше цена ошибки, тем меньше можно позволить себе подход: «У нас просто есть папка с данными». Нужна управляемая история: откуда появились данные → что с ними происходило → кто принимал решения → какая версия использовалась для конкретной модели. И это выводит нас к последней главе. После всех технических, производственных и governance-вопросов остаётся понять: как будет выглядеть работа с датасетами через 3–5 лет и что ML-командам стоит менять уже сейчас.

ГЛАВА 10 / 10

Датасеты 2030: как изменится разметка данных в ближайшие 3–5 лет

01 / FOCUS

Model-assisted annotation

02 / FOCUS

Continuous datasets

03 / FOCUS

Data governance

В ближайшие 3–5 лет работа с обучающими данными будет всё меньше напоминать разовый этап подготовки датасета и всё больше — непрерывный контур эксплуатации ML-системы. Причина не только в росте автоматизации разметки: production постоянно создаёт новые состояния среды, ошибки и пограничные случаи, которые необходимо превращать в проверенные обучающие данные.

Классический pipeline был преимущественно линейным: данные собирали, размечали, использовали для обучения и после запуска модели считали проект завершённым.

В production-подходе модель и датасет развиваются совместно: ошибки и новые состояния среды становятся источником данных для разметки, QA и последующего дообучения.

Human-in-the-Loop останется, но роль человека изменится

Автоматическая предразметка и model-assisted annotation будут забирать всё больше простых и типовых случаев. Человек будет концентрироваться на том, где автоматизация наиболее уязвима:

- hard cases;
- редкие классы;
- спорные дефекты;
- новые категории;
- low-confidence predictions;

критичные для бизнеса ошибки. То есть ценность ручной разметки будет всё меньше определяться количеством обработанных объектов и всё больше — качеством решений на сложных данных.

Датасет станет живым объектом

В зрелых ML-системах важно будет отслеживать не только версии модели, но и:

- Dataset versions: v1, v2, v3
- Guideline versions: v1, v2
- Production conditions
- Hard cases
- Drift

Модель и датасет фактически будут развиваться вместе. Поэтому всё большую роль будут играть:

- versioning;
 - metadata;
 - provenance;
 - dataset monitoring;
 - continuous QA;
- data governance.

Собирать будут не больше данных, а более полезные данные

Один из главных сдвигов — переход от вопроса: «Сколько ещё изображений нужно?» к вопросу: «Какие данные сильнее всего улучшат следующую версию модели?» Это означает больший интерес к active learning, targeted annotation, error-driven sampling и автоматическому поиску hard cases.

Что проверить до смены архитектуры модели

Если качество модели перестало устраивать команду, перед полной перестройкой ML-решения стоит проверить:

- где именно концентрируются ошибки;
- корректны ли labels;
- достаточно ли hard cases;
- покрывает ли датасет реальные условия;
- нет ли class imbalance;
- не изменился ли production;
- одинаково ли annotators понимают guideline;
- правильно ли сформированы train / validation / test;

- есть ли актуальная версия датасета;
- можно ли локально исправить проблемный сегмент. И только после этого решать, действительно ли узкое место находится в архитектуре.

Итог

Хорошая ML-модель начинается не с максимального количества данных.

Она начинается с управляемого цикла сбора данных, разметки, QA, обучения, анализа production-ошибок и возврата новых примеров в следующую версию датасета.

Чем быстрее команда умеет находить слабые места этого цикла и возвращать их обратно в датасет, тем устойчивее становится модель.

Именно поэтому работа с данными сегодня — это уже не вспомогательный этап ML-разработки, а часть постоянной эксплуатации AI-системы.

Модель ошибается — возможно, проблема находится в данных

US-DATA помогает ML-командам собирать, размечать, проверять и улучшать датасеты для production AI.

Обсудить проект / заказать тестовую разметку → usdataml.com