

Автоматизированная оценка качества денситометрических исследований

Техническое описание решения

Задача	Контроль качества DXA-исследований: бинарная оценка «качественное / есть нарушение», определение анатомической области, тип нарушения, структурированный отчёт
Дата документа	27 сентября 2026 г.
Команда	Грачев Денис — разработка; Грачев Татьяна — капитан
Версия решения	1.0.0 (src/__init__.py)
Рабочий чекпоинт	models/dxa_model.pth : ResNet18 + линейная голова, разметка table, seed 42, эпоха 39, порог логита -0.4930 (вероятность 0.379)
Ключевые метрики	ROC-AUC 0.6764 [0.6309, 0.7218], F1 0.5676 [0.5270, 0.6082] — пять seed'ов на фиксированном разбиении по исследованиям
Время обработки	0.021 с на изображение (медиана, CPU), 548 файлов за 12.1 с; требование «≤ 3 мин на исследование» выполняется с запасом

Документ описывает фактическое состояние репозитория на дату, указанную на титуле: все параметры, пороги, метрики и имена полей приведены по коду и по выполненным измерениям, а не по проектным намерениям. Там, где значение измерено на конкретной машине, указано, на какой. Расхождения и ограничения перечислены явно — разделы 15 и 16.

Содержание

- 1. Назначение и область применения
- 2. Соответствие требованиям задания
- 3. Архитектура решения
- 4. Состав данных и разметка
- 5. Предобработка изображений
- 6. Модель и процедура обучения
- 7. Определение анатомической области
- 8. Таксономия нарушений качества
- 9. Форматы входных и выходных данных
- 10. API сервиса
- 11. Веб-интерфейс
- 12. Метрики качества
- 13. Производительность и системные требования
- 14. Сборка, запуск и развёртывание
- 15. Тесты и проверки

16. Известные ошибки и их обработка

17. Ограничения и достоверность результатов

18. План развития

Приложение А. Структура репозитория

Приложение Б. Команды

1. Назначение и область применения

Сервис принимает денситометрическое исследование в формате DICOM и выполняет автоматизированный контроль его качества: определяет анатомическую область, решает бинарную задачу «изображение пригодно для клинической интерпретации / содержит нарушение качества», относит нарушение к категории и формирует структурированный отчёт (XLSX/CSV) с одной строкой на изображение.

Задача решается как цифровой помощник, а не как замена врача: результат предназначен для отбора исследований, требующих внимания специалиста, и для единообразной регистрации нарушений. Области анализа — поясничный отдел позвоночника и проксимальный отдел бедренной кости (левая и правая стороны рассматриваются как отдельные снимки, поскольку сторона влияет на трактовку критериев).

Решение работает полностью офлайн: изображения не передаются во внешние сервисы, все зависимости и веса фиксированы, доступ в сеть при инференсе не требуется. Это прямое требование методики (п. 3.2 задания) и одновременно условие применимости в закрытом контуре медицинской организации.

2. Соответствие требованиям задания

Таблица связывает пункты задания с реализацией и со способом проверки. Формулировки требований приведены по тексту методических рекомендаций.

Требование	Реализация	Чем подтверждается
Области: поясничный отдел позвоночника и проксимальный отдел бедренной кости	Область определяется автоматически по кадру; бедро разделяется на левое и правое	Разд. 7; на валидации 99/99 для позвоночника
Бинарная классификация «качественное / есть нарушение»	ResNet18 (ImageNet) + линейная голова, порог по логиту, подобранный по F1	Разд. 6, 12; tests/test_preprocess_and_model.py
Определение типа нарушения	Единый словарь кодов src/dxa/violations.py; отнесение к категории — по измеряемым признакам снимка	Разд. 8; tests/test_violations.py
Оценка корректности разметки анатомических структур	В наборе нет ROI-разметки в DICOM, поэтому оценивается геометрия видимой зоны измерения и её границы	Разд. 8, 17 (ограничение 4)
Отчёт .xlsx / .csv, одна строка на изображение, столбцы из п. 2.5	CLI-инференс и POST /api/v1/export; обязательные восемь столбцов плюс три диагностических	Разд. 9 (состав столбцов проверен на реальном прогоне)
Время обработки одного исследования ≤ 3 мин	Один снимок — один прямой проход сети 224×224, без итеративных процедур	Разд. 13: 0.021 с медиана на CPU, запас более чем трёхкратный
Отсутствие необработанных исключений; ошибки фиксируются в отчёте	Ошибка на файле не прерывает пакет: строка получает processing_status = Failure: ...	Разд. 16
Воспроизводимость при повторном запуске	Фиксированный seed, зафиксированное разбиение в файле, версии зависимостей, самодостаточный чекпоинт	Разд. 6, 14; tests/test_labels.py
Пакетная обработка архива + общая таблица + zip с дополнительными сериями	Обход каталога, лог прогресса, XLSX/CSV, опциональный zip с визуализацией	Разд. 9, 13
API для пакетной обработки тестового набора	FastAPI: восемь маршрутов, включая /api/v1/batch и /api/v1/export	Разд. 10
Обязательная контейнеризация, скрипт сборки и запуска в Linux	Dockerfile (python:3.11-slim), Dockerfile_cuda для GPU, run.sh	Разд. 14; сборка проверена, см. 14
Фиксация зависимостей, включая базовый контейнер	requirements.txt с точными версиями, базовый образ по тегу, torch с CPU-индексом	Разд. 14

Работа без обращения к внешним сервисам	Офлайн-ассеты фронтенда, отсутствие сетевых вызовов в коде предсказания	tests/browser/ui_offline.js
Полный комплект документации	README.md (обзор и воспроизведение), assets/labeling.md (разметка и её обоснование), docs/technical-description.html и .pdf (данный документ)	—

3. Архитектура решения

3.1. Слои

Слой	Файлы	Ответственность
Прикладной ядерный слой	src/dxa/	Чтение DICOM, предобработка, модель, метрики, определение области, разметка данных, обучение, пакетный инференс
Сервис	src/main.py	FastAPI: загрузка чекпоинта при старте, маршруты анализа, пакетной обработки, экспорта, здоровья и карточки решения
Веб-интерфейс	src/api/static/	Загрузка файлов, таблица результатов, панель деталей, панель «О модели»; все подписи и метрики получает с сервера
Эвристики качества	src/quality/	Числовые измерения снимка (резкость, плотные включения, полнота и поворот области) для панели деталей
Инфраструктура	run.sh, Dockerfile, Dockerfile_cuda, Jenkinsfile	Единая точка входа для сборки, обучения, инференса и тестов; сборка образа и выкладка

3.2. Поток обработки одного изображения

Ниже — фактический путь данных от файла до строки отчёта. Ключевое свойство: обучение и сервис используют *один и тот же* код предобработки и предсказания, поэтому расхождение между офлайн-оценкой и работой API невозможно по построению, а не по договорённости.



Порядок операций и точка принятия решения:

- 1. Чтение.** Из файла берутся пиксельные данные и идентификаторы исследования и снимка (StudyInstanceUID, SOPInstanceUID). Персональные данные не используются: решение не зависит

от их наличия.

2. **Предобработка.** Приводится к виду, ожидаемому предобученным backbone (описание — разд. 5). Параметры предобработки хранятся в чекпоинте и берутся из него, а не из кода по умолчанию.
3. **Оценка качества.** Один прямой проход сети даёт логит. Решение принимается по логиту: `quality_class = 1`, если логит выше порога. Порог подобран по F1 на валидации и сохранён в чекпоинте (текущее значение -0.4930 , что соответствует вероятности 0.379).
4. **Анатомическая область.** Определяется независимо от модели, по геометрии кадра (разд. 7), с оценкой уверенности.
5. **Измерения и тип нарушения.** Для снимков с нарушением вычисляются числовые признаки снимка, по которым нарушение относится к категории словаря (разд. 8).
6. **Отчёт.** Строка собирается со статусом обработки и временем; ошибка на файле не прерывает пакет.

В `README.md` приведён полный набор проектных диаграмм (компоненты, потоки данных, последовательность, развёртывание). Они описывают целевую архитектуру, включая блоки, которые в текущей реализации не используются, поэтому в настоящем документе поток описан текстом по коду.

4. Состав данных и разметка

4.1. Набор

Показатель	Значение	Пояснение
Файлов на диске	544	в обучающем наборе (<code>dataset_hack/</code> <code>НД_для_обучения</code>)
Уникальных снимков	252	по пиксельному содержимому; 292 файла — побайтные дубли
Исследований	100	разбиение выполняется по исследованиям, а не по снимкам
Позвоночник / бедро правое / бедро левое / не определено	99 / 79 / 73 / 1	голосование по именам файлов после склейки дублей
Нарушений по экспертной таблице	74 (29.4 %)	три снимка таблица не оценивала
Нарушений в рабочей разметке	77 (30.6 %)	74 по таблице плюс 3 снимка с пометкой в имени файла

Дубли не пересекают границы исследований, конфликтов меток при склейке не возникает. Два побайтных дубля названы по-разному, поэтому область определяется голосованием по именам файлов. Изучение набора выявило две особенности, которые пришлось учесть в разметке: один и тот же снимок встречается под несколькими именами (без склейки он попадал бы одновременно в оба класса), и имена файлов местами расходятся с оценкой эксперта.

4.2. Единица разметки и правило получения метки

Экспертная таблица описывает **исследование**, а не отдельный снимок. Однако каждая анатомическая область встречается в исследовании ровно один раз (после склейки дублей), поэтому вердикт исследования по области переносится на снимок однозначно — не требуется решать, какой из нескольких снимков «плохой». Так получены метки и типы нарушений

(labels/labels_images.csv); каждый источник свидетельства сохранён в отдельном столбце, поэтому правило можно переиграть без повторного разбора данных.

Возможны были два правила: учитывать только экспертную таблицу либо дополнительно учитывать пометки `_good / _bad`, проставленные вручную в именах файлов. Пометки расходились с экспертом в 15 случаях из 252, поэтому правило выбиралось измерением: одно зафиксированное разбиение, пять seed'ов обучения, один эталон (табл. в разд. 12.3). Выбрано правило «только экспертная таблица».

4.3. Разбиение выборки

Часть	Снимков	Исследований	Нарушений	Файл
Обучение	199	81	61	labels/split_expert_seed42.json
Валидация	53	19	16	

Разбиение стратифицировано по эталону и **фиксировано в файле**: все сравниваемые варианты обучаются и оцениваются на одном и том же held-out наборе, что делает сравнения корректными. Снимки одного исследования не попадают одновременно в обе части — это исключает утечку, при которой метрики растут за счёт запоминания конкретных исследований. Наличие утечки и воспроизводимость разбиения проверяются тестами (`tests/test_labels.py`).

5. Предобработка изображений

Предобработка описана декларативно в `src/dxa/preprocess.py` (неизменяемая конфигурация `PreprocessConfig`) и используется и при обучении, и при работе API. Параметры хранятся в чекпоинте и восстанавливаются из него, поэтому вход модели не может разойтись между двумя режимами.

Параметр	Значение	Назначение
<code>norm</code>	<code>percentile</code>	приведение интенсивностей к $[0, 1]$; допустимо также <code>minmax</code>
<code>p_low / p_high</code>	<code>0.5 / 99.5</code>	отсекаемые перцентили — устойчивость к выбросам и шуму
<code>imagenet_norm</code>	<code>true</code>	стандартизация mean (0.485, 0.456, 0.406), std (0.229, 0.224, 0.225), как ожидает предобученный backbone
<code>input_size</code>	<code>224</code>	сторона квадратного входа сети

Порядок операций над одним файлом:

- Чтение.** `pydicom.dcmread` и `pixel_array` в `float32`; многокадровое изображение усредняется по кадрам, иная размерность — ошибка.
- Коррекция интенсивностей.** Учитываются `RescaleSlope` и `RescaleIntercept`; для `MONOCHROME1` яркость инвертируется.
- Нормировка.** По перцентильям `0.5 / 99.5` с обрезкой в $[0, 1]$ и защитой от вырожденного диапазона ($hi \leq lo$).
- Приведение к тензору.** Умножение на 255 и `uint8`, дублирование одного канала в три, `resize 224×224` билинейно, перевод в CHW, деление на 255, затем нормировка ImageNet.

Если в чекпоинте нет блока `preprocess` (старый файл), применяются значения по умолчанию — `percentile` с нормировкой ImageNet; устаревший профиль `minmax` без ImageNet доступен флагом `--`

no-imagenet-norm.

Порядок «resize в uint8, затем нормировка» сохранён намеренно: он совпадает с историческим кодом проекта, поэтому смена реализации не сдвинула распределение входа и не обесценила ранее подобранный порог.

6. Модель и процедура обучения

6.1. Архитектура

Элемент	Реализация
Backbone	ResNet18 с весами ImageNet (IMAGENET1K_V1), все слои кроме fc ; признак — 512 чисел. Флагом --backbone допустим также ResNet34
Голова качества	по умолчанию linear: Flatten + Linear(512, 2) — линейный зонд, 1026 обучаемых параметров. Режим mlp (флаг --head): Linear(512, 256) + ReLU + Dropout(0.3) + Linear(256, 2)
Вспомогательная голова области	Flatten + Linear(512, 4) : три анатомические области плюс «неизвестно». Обучается с весом 0.3 как дополнительная задача и заставляет backbone различать анатомию; логиты качества она не сдвигает
Стандартизация входа головы	признаки нормируются по среднему и СКО обучающей выборки, зафиксированным в буферах feat_mean / feat_std . Без неё логиты смещены, вероятности скучены у нуля и подобранный порог теряет смысл

6.2. Гиперпараметры и процедура обучения

Параметр	Значение по умолчанию	Комментарий
Оптимизатор	AdamW	<code>lr 3e-4, weight_decay 5e-2</code> . Заметный <code>weight decay</code> нужен не только против переобучения: без него логиты за 100 эпох насыщаются и порог вырождается
Планировщик	ReduceLROnPlateau	множитель 0.5, терпение 3, следит за <code>val_loss</code>
Функция потерь	CrossEntropy	балансировка классов выключена (<code>--balance none</code>); доступны <code>loss</code> (вес <code>n_good / n_bad</code>) и <code>sampler</code>
Батч / эпохи	16 / 100	ранняя остановка после 25 эпох без улучшения
Отбор чекпоинта	окно 5	оценка эпохи — ROC-AUC, сглаженный по последним 5 эпохам; до заполнения окна чекпоинт не сохраняется, при равном AUC решает F1
Вес головы области	0.3	<code>--region-loss-weight</code> ; снимки с неизвестной областью в её <code>loss</code> не входят
Обрезка градиентов	норма 5.0	защита от выбросов
Заморозка backbone	всегда	<code>--freeze-epochs -1</code> по умолчанию (линейный зонд); при размораживании <code>lr</code> падает в 10 раз
Seed / устройство	42 / авто	порядок выбора устройства: CUDA → MPS → CPU

Почему линейный зонд. При ~250 уникальных снимках полный fine-tune ResNet18 за несколько эпох запоминает обучающую выборку: train F1 стремится к 1.0, а val AUC падает к 0.5. Поэтому по умолчанию backbone заморожен и обучается только голова на признаках ImageNet. При заморозке слои BatchNorm остаются в режиме `eval`: иначе бегущие статистики продолжали бы обновляться и признаки «уезжали» бы от тех, на которых рассчитывалась стандартизация.

Почему порог по логиту. Порог хранится и применяется в логитах (log-odds): после обучения линейный зонд разделяет обучающую выборку почти идеально, вероятности насыщаются в 0/1, и порог в единицах вероятности вырождается. Для человека порог переводится в вероятность через сигмоиду. ROC-AUC и PR-AUC считаются по вероятностям и от порога не зависят.

Как подбирается порог. На каждой эпохе по валидации: кандидаты — все наблюдаемые логиты плюс края диапазона, выбирается максимум F1; при равном F1 берётся меньший логит, чтобы не терять recall. Аргумент `--min-recall` позволяет вместо этого взять максимальный порог с recall не ниже заданного.

Рабочий чекпоинт. Лучшая эпоха — 39 из прогона в 64 эпохи (обучение остановлено по терпению), порог логита -0.4930 (вероятность 0.379). Метрики этой эпохи — в разделе 12, честная оценка варианта на пяти seed'ax — ROC-AUC 0.6764.

6.3. Чекпоинт и отчёты

Чекпоинт самодостаточен: кроме весов в нём лежат архитектура, параметры предобработки, порог и происхождение данных, поэтому инференс не может рассинхронизироваться с обучением.

Поле	Содержимое
<code>model_state_dict</code>	веса сети; <code>format_version</code> — 2
<code>backbone</code> , <code>head</code>	архитектура; при явном расхождении с запрошенной загрузка завершается ошибкой
<code>preprocess</code>	параметры предобработки (раздел 5)
<code>threshold</code> , <code>threshold_logit</code>	рабочий порог по логитам
<code>labels_csv</code> , <code>split_file</code>	на какой разметке и на каком разбиении обучена модель
<code>epoch</code> , <code>val_metrics</code> , <code>selection_score</code>	эпоха, её метрики и сглаженная оценка, по которой выбран чекпоинт
<code>history</code> , <code>optimizer_state_dict</code>	история обучения и состояние оптимизатора

Рядом с чекпоинтом обучение пишет `train_report.md` и `train_report.json`: гиперпараметры, метрики на валидации, метрики по областям и историю по эпохам. Артефакты кладутся в `--output-dir` (по умолчанию `models`).

7. Определение анатомической области

Область определяется по содержимому снимка, а не по имени файла: на закрытом наборе соглашение об именах может отсутствовать. Решение принимается по геометрии кадра (`resolve_region` в `src/dxa/inference.py`) независимо от модели качества; обученная вспомогательная голова области привлекается только как уточнение.

Признаки считает функция `heuristic_signals` по маске яркой (костной) ткани с порогом по 95-му перцентилю:

Признак	Что измеряет
<code>width</code>	ширина кадра в пикселях — основной признак области на этом оборудовании (у позвоночника 300 px, у бедра 280 px)
<code>bbox_aspect</code>	отношение высоты яркой области к её ширине (вытянутость)
<code>left_right_ratio</code>	перевес светимости левой половины над правой — наклон в сторону бедра
<code>symmetry</code>	симметрия изображения относительно вертикальной оси

Порог ширины задан константой `SPINE_MIN_WIDTH = 295`. Порядок решений:

1. если ширина кадра ≥ 295 — `spine`, уверенность 0.8;
2. иначе (бедра) — сторона по перевесу светимости: `left_right_ratio > 1.3` даёт `hip_right`, < 0.7 — `hip_left`, уверенность 0.6;
3. при промежуточном перевесе — предсказание головы области, если её уверенность ≥ 0.5 ;
4. если и оно неуверенно — общая область `hip` с уверенностью 0.4.

Если ширина кадра неизвестна, используется предсказание головы, а затем форма яркой области (`bbox_aspect` и `symmetry`).

Точность. Сопоставление с областью из рабочей разметки (252 снимка, раздел 4):

Что проверялось	Результат
Позвоночник	99 / 99
Сторона бедра (правое / левое)	135 / 152 (88.8 %)
Итого по всем областям	234 / 251 (93.2 %)

Различение позвоночника и бедра по ширине кадра работает безошибочно, а сторона бедра определяется менее надёжно: перевес светимости путает левое и правое в 17 случаях из 152. Сторона не проверялась по тегам DICOM — `Laterality` в наборе пуст, — поэтому это ограничение (раздел 17, п. 7), а не измеренная ошибка модели.

Признак «ширина кадра» привязан к текущему аппарату. Он безошибочен на этом наборе, но при смене оборудования порог потребует калибровки — раздел 17, ограничение 8.

8. Таксономия нарушений качества

Коды типов нарушений собраны в одном модуле `src/dxa/violations.py`. Раньше словарь был свой в трёх местах (инференс, `main.py` и веб-интерфейс), и ни одна копия не знала кодов экспертной таблицы; теперь коды, русские подписи и пояснения заданы один раз, а сервер отдаёт их клиенту.

Код	Подпись	Область	Источник
<code>positioning</code>	Некорректная укладка	<code>spine</code>	экспертная таблица
<code>axis_deviation</code>	Отклонение оси	<code>spine</code>	экспертная таблица
<code>artifact</code>	Артефакты и импланты	любая	экспертная таблица
<code>rotation</code>	Ротация / позиционирование	<code>hip</code>	экспертная таблица
<code>roi_incorrect</code>	Некорректная область интереса	любая	экспертная таблица
<code>motion</code>	Движение, размытие	любая	<code>condition_doctor.txt</code>
<code>incomplete_anatomy</code>	Анатомия видна не полностью	любая	<code>condition_doctor.txt</code>
<code>labeling_error</code>	Ошибка разметки	<code>spine</code>	<code>condition_doctor.txt</code>
<code>unspecified</code>	Нарушение без уточнения	любая	системный код

У каждого кода есть пояснение, чем нарушение мешает измерению (например, для `rotation` — что малый вертел виден слишком хорошо и шейка бедра кажется укороченной). Функция `catalogue()` отдаёт словарь через `/api/v1/model`, поэтому подписи в интерфейсе и в выгрузке не могут разойтись с кодами.

Для текстового отчёта DICOM SR у каждого кода заданы условный числовой код и английская формулировка. Полноценного справочника SNOMED/DICOM для контроля качества DXA в наборе нет, поэтому коды условные и всегда идут рядом с текстом: пригодный снимок кодируется `113001` (DXA image quality acceptable) с флагом `FINAL`, нарушение — с флагом `WARNING`. Устаревшие значения внешних источников (`artifact_motion`, `roi_error`, `position_error`, `quality_violation_detected` и другие) приводит к канону функция `canon_type`.

8.1. Тип нарушения в разметке

Экспертная таблица кодирует не все девять кодов, а пять: `positioning`, `axis_deviation`, `artifact`, `rotation`, `roi_incorrect`. Распределение в рабочей разметке (в четырёх из 77 нарушений указано по два критерия, поэтому сумма больше 77):

Код	Снимков
<code>rotation</code>	36
<code>artifact</code>	17
<code>axis_deviation</code>	10
<code>roi_incorrect</code>	7
<code>positioning</code>	6
<code>unspecified</code>	5

Коды `motion`, `incomplete_anatomy` и `labeling_error` в разметке не встречаются: таблица их не кодирует, хотя словарь их знает ради критериев `condition_doctor.txt`. Пять снимков имеют только `unspecified`, потому что источник не указывает критерий.

8.2. Тип нарушения при инференсе

Модель решает только бинарную задачу, поэтому тип для снимка с нарушением выбирает функция `classify_violation_type` по дешёвым признакам: размытие (`laplacian_variance`), доля плотных пикселей (`bright_fraction`) и вытянутость яркой области (`bbox_aspect` для бедра). При отсутствии выраженного признака возвращается `unspecified`.

На текущем чекпоинте эта ветка практически вырождена. Пороги `motion_threshold` и `artifact_threshold` в вызов не передаются, поэтому действуют значения по умолчанию (0.0 и 1.0), которых признаки достичь не могут, а условие ротации (`bbox_aspect` вне диапазона 0.4–3.0) на наборе не срабатывает. Прогон по всем 548 файлам даёт одинаковый результат: все 341 решение с нарушением помечены `unspecified`. Для содержательного типа нужна разметка типов на уровне снимка и обученный мультилейбл-классификатор (раздел 18), поэтому в ответе API тип всегда идёт с флагом `violation_type_is_heuristic = true`.

9. Форматы входных и выходных данных

9.1. Вход

Один файл DICOM (для API — один HTTP-загрузкой, для CLI — файл или каталог с рекурсивным обходом). Исследование может содержать несколько снимков — каждый обрабатывается отдельно и даёт отдельную строку отчёта. Перед обработкой файл читается целиком; при ошибке чтения строка получает статус сбоя, а пакет продолжается.

9.2. Выходной файл

Состав столбцов проверен на реальных прогонах — и через CLI, и через `/api/v1/export`. Обязательные восемь столбцов задания совпадают в обоих путях по составу и порядку; дополнительно пишутся `confidence` и `violation_reason`, а CLI-инференс добавляет одиннадцатый столбец `region_confidence`. Таблица ниже отражает CLI-вывод; в ответе `/api/v1/export` последнего столбца нет.

№	Столбец	Тип	Содержимое
1	path_to_study	строка	путь к исследованию (для загрузки через API — <code>upload://имя</code>)
2	study_uid	строка	StudyInstanceUID из DICOM
3	image_uid	строка	SOPInstanceUID из DICOM
4	anatomical_region	строка	spine / hip_left / hip_right / hip
5	quality_class	целое	0 — качественное, 1 — есть нарушение
6	violation_type	строка	код нарушения или пусто
7	processing_status	строка	Success либо Failure: <причина>
8	time_of_processing	вещественное	время обработки снимка, секунды
9	confidence	вещественное	уверенность модели в решении
10	violation_reason	строка	текстовое пояснение к отнесённой категории
11	region_confidence	вещественное	уверенность определения области

Время обработки отдаётся в столбце файла результатов; одиночный ответ `POST /api/v1/analyze` этого столбца не содержит — при интеграции время измеряется на стороне вызывающей системы (см. разд. 10 и 13).

Расхождение ровно в один столбец — единственное различие двух путей выгрузки; оно перечислено в разделе 16. Обязательные столбцы и их порядок при этом одинаковы, поэтому автоматический разбор результатов не зависит от того, каким путём получен файл.

10. API сервиса

Сервис на FastAPI. Маршруты и состав полей ответов ниже приведены по фактическим ответам на запросах к тестовым файлам, а не по схеме из документации кода.

Метод	Путь	Назначение
GET	/	веб-интерфейс
GET	/api/v1/health	статус сервиса; поля ответа: <code>status</code> , <code>device</code> , <code>model_loaded</code> , <code>model</code>
GET	/api/v1/model	карточка решения: <code>model</code> , <code>labels</code> , <code>dataset</code> , <code>comparison</code> , <code>violations</code> , <code>limitations</code> , <code>sources</code> , <code>snapshot_date</code> , <code>loaded</code>
POST	/api/v1/analyze	анализ одного файла (multipart-загрузка)
POST	/api/v1/analyze/detailed	расширенный отчёт, опционально с маской
POST	/api/v1/analyze/sr	текстовое представление отчёта DICOM SR: <code>format</code> , <code>sr_content</code> , <code>UID</code>
POST	/api/v1/batch	пакетная обработка нескольких файлов
POST	/api/v1/export	пакетная обработка и выгрузка в XLSX

10.1. Состав ответа /api/v1/analyze

Поля: filename, study_uid, image_uid, anatomical_region, region_confidence, quality_class, quality_label, overall_quality, severity, confidence, confidence_per_class, threshold_probability, violation_type, violation_type_label, violation_type_is_heuristic, violation_type_note, reason, metrics, samples, processing_status.

Два поля заслуживают пояснения при интеграции:

- violation_type_is_heuristic = true — тип нарушения определён по измеряемым признакам снимка, а не предсказан моделью; модель решает только бинарную задачу. Флаг отдаётся клиенту явно, чтобы это различие не терялось в интеграции.
- quality_label, violation_type_label, violation_type_note — готовые подписи для интерфейса. Они приходят с сервера: клиент не держит собственных копий словаря, поэтому подписи не могут разойтись с кодами.

10.2. Расширенный ответ /api/v1/analyze/detailed

Дополнительно к предыдущему набору: image (визуализация снимка), mask (маска зоны измерения, при запросе), reasons, metrics_note, view_quality, а также измерения по областям: spine_completeness, hip_completeness, hip_rotation.

Измерения в расширенном ответе сопровождаются пометкой о том, что их пороги не калиброваны (разд. 17), и не превращаются в вердикты «да/нет»: интерфейс показывает числа.

11. Веб-интерфейс

Интерфейс предназначен для ручной проверки работы сервиса и для демонстрации: загрузка файлов, таблица результатов со статистикой, панель деталей по выбранной строке и панель «О модели» с источником загруженного чекпоинта.

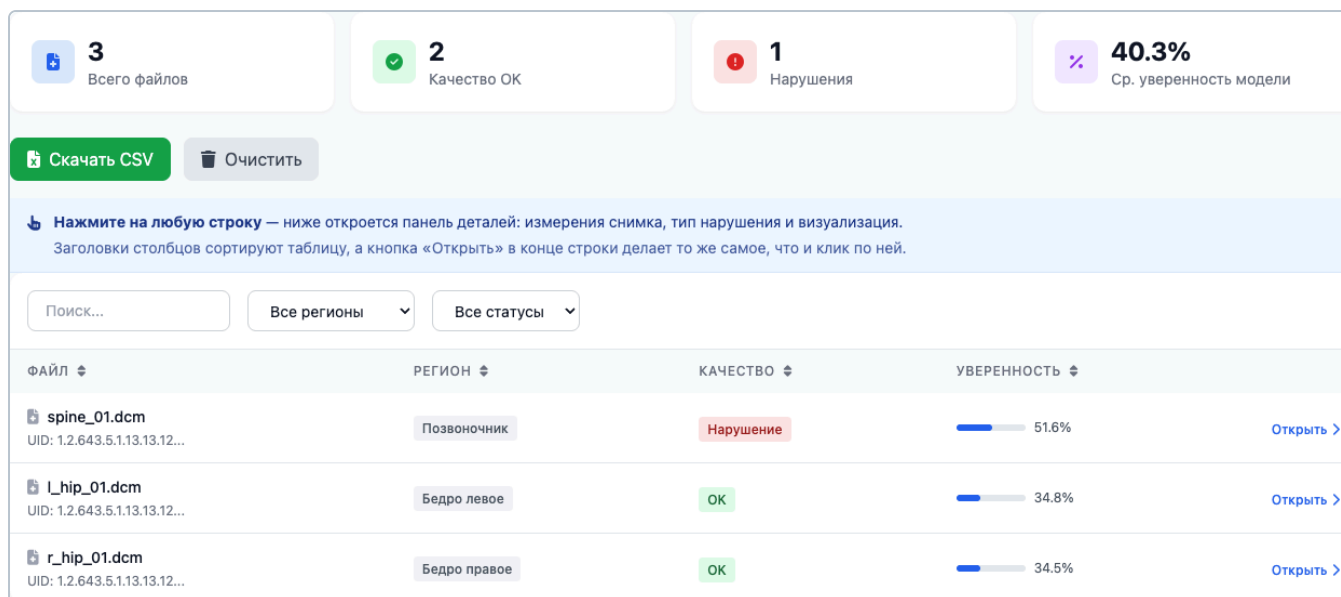


Рис. 1. Результаты обработки: полоса состояния показывает, какой чекпоинт отвечает (разметка, эпоха, порог), рядом — статистика и таблица снимков.

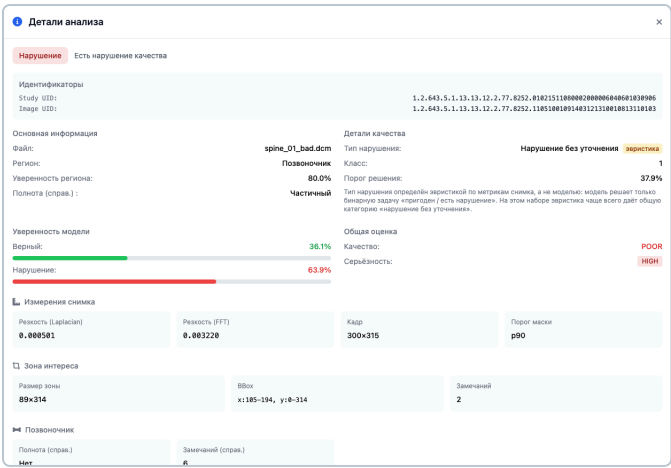


Рис. 2а. Нарушение: уровень, заключение с вероятностью и порогом, тип нарушения с пометкой «эвристика».

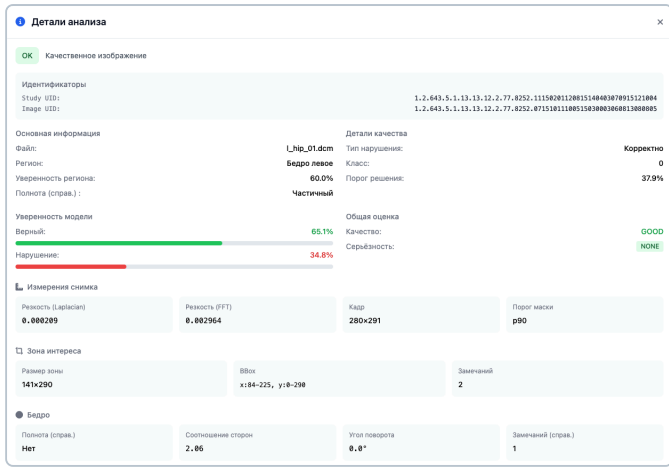


Рис. 2б. Качественный снимок: измерения подписаны как справочные.

Отдельный принцип — **интерфейс не утверждает больше, чем известно решению:**

- тип нарушения показан с пометкой «эвристика» и пояснением, что модель решает только бинарную задачу;
- метрика рабочего чекпойнта помечена как завышенная (чекпойнт выбирался лучшим из пяти seed'ов), а честная оценка варианта приведена рядом — в панели «О модели»;
- плитка средней уверенности не называется точностью: точность требует эталонных меток, которых для произвольного файла нет;
- числовые измерения панели деталей идут под дисклеймером о некалиброванности порогов.

О модели и разметке

×

Рабочий чекпоинт:models/dxa_model.pth

Архитектура:resnet18 + linear (линейный зонд, backbone заморожен)

Разметка:labels/labels_images.csv

Разбиение:по умолчанию, стратифицированное по исследованиям

Файл чекпоинта изменён:2026-09-26T22:36:14

Порог решения:логит -0.493013 → вероятность 37.9%

⚠ Это обычный прогон с seed по умолчанию, а не лучший из выборки. Незавышенная ожидаемая оценка того же варианта разметки — среднее по пяти seed'ам: ROC-AUC 0.6764 [0.6309, 0.7218]. Его собственная валидационная ROC-AUC 0.6706 близка к этому среднему.

Снимок сведений: 2026-09-26

Качество: выбор правила разметки

МЕТРИКА	ТОЛЬКО ЭКСПЕРТНАЯ ТАБЛИЦА	ТАБЛИЦА И СЛУЖЕБНЫЕ ПОМЕТКИ
ROC-AUC	0.6764 [0.6309, 0.7218]	0.6199 [0.5840, 0.6559]
PR-AUC	0.4759 [0.4141, 0.5377]	0.4046 [0.3702, 0.4391]
F1	0.5676 [0.5270, 0.6082]	0.5426 [0.5073, 0.5778]
Δ ROC-AUC	+0.0564 [0.0403, 0.0725]	
Δ PR-AUC	+0.0713 [0.0398, 0.1028]	
Δ F1	+0.0251 [-0.0056, 0.0558]	

Метку можно брать только из экспертной таблицы или дополнительно учитывать служебные пометки, предоставленные при подготовке набора. Пометки расходились с оценкой эксперта в 15 случаях из 252, поэтому правило нужно было выбрать измерением. Валидация: 53 снимков, 16 нарушений по эталону; seed'ы: 0, 1, 2, 3, 4. 5 из 5 seed'ов в пользу экспертной таблицы Принято правило 'table': служебные пометки в метках не участвуют. Эталон — вердикт эксперта из labels/labels_images_expert.csv.

Данные

Уникальных снимков

252

Исследований

100

Нарушений по экспертной таблице

74

Нарушений в обучающей разметке

77

Снимков без экспертной оценки

3

Позвоночник / бедро R / бедро L

99 / 79 / 73

Файлов на диске

544

Доля нарушений

30.6%

Рис. 3. Панель «О модели»: источник чекпоинта, метрики с доверительными интервалами, состав данных, словарь нарушений и список ограничений. Данные приходят из `/api/v1/model`.

12. Метрики качества

12.1. Основная оценка

Оценка получена на фиксированном разбиении по исследованиям (обучение 199 снимков / 81 исследование, валидация 53 снимка / 19 исследований, 16 нарушений), пять seed'ов обучения, эталон — вердикт эксперта из таблицы. Приоритетные по заданию метрики приведены с 95 % доверительными интервалами.

Метрика	Значение	95 % ДИ	Комментарий
ROC-AUC	0.6764	[0.6309, 0.7218]	приоритетная метрика задания
PR-AUC	0.4759	[0.4141, 0.5377]	базовый уровень при доле нарушений 30 % — около 0.30
F1	0.5676	[0.5270, 0.6082]	порог подбирался по F1 на той же валидации, поэтому значение смещено вверх
Recall / Precision	0.700 / 0.486	—	рабочая точка выбранного порога

12.2. Оценка по областям и контрольные проверки

Что измерено	Значение	Как получено
ROC-AUC: позвоночник / бедро правое / бедро левое	0.943 / 0.576 / 0.550	рабочий чекпоинт, собственная валидация
Контрольная задача «позвоночник / бедро»	AUC 1.00	проверка работоспособности пайплайна, а не клиническая метрика
Модель против правила «позвоночник = нарушение»	0.854 против 0.529	оценка на всём наборе, включая обучающие снимки, поэтому смещена вверх

Разбивка по областям нужна потому, что нарушения распределены неравномерно: в позвоночнике 33 из 99 снимков, у бёдер 21–22 из 73–79, а сама область почти однозначно определяется по ширине кадра. Поэтому общий AUC частично отражает различие области, а не только распознавание дефекта. Чтобы отделить одно от другого, выполнена проверка: правило «позвоночник = нарушение» даёт внутри областей 0.50 (подсказки нет), модель — 0.85–0.93. Это означает, что модель использует содержимое снимка, а не только область.

12.3. Выбор правила разметки

Метрика (эталон)	только таблица	с суффиксами имён
ROC-AUC	0.6764 [0.6309, 0.7218]	0.6199 [0.5840, 0.6559]
PR-AUC	0.4759 [0.4141, 0.5377]	0.4046 [0.3702, 0.4391]
F1	0.5676 [0.5270, 0.6082]	0.5426 [0.5073, 0.5778]
Recall / Precision	0.700 / 0.486	0.863 / 0.404

Парная разница (только таблица минус с суффиксами): ROC-AUC **+0.0564** [+0.0403, +0.0725], PR-AUC **+0.0713** [+0.0398, +0.1028] — знаки **+++++**, то есть преимущество на всех пяти seed'ах. Принято правило «только экспертная таблица». Оговорка, важная для интерпретации: эталон — та же таблица, поэтому вариант, обучавшийся на ней, находится в выигрышном положении; значимо здесь то, что добавление ненадёжных пометок согласие с экспертом **снижает**, а не повышает.

13. Производительность и системные требования

13.1. Измерения

Измерения выполнены на рабочей машине разработчика (macOS, Apple Silicon) на полном наборе данных — 548 файлов в `dataset_hack` (544 обучающих плюс 4 тестовых). Инференс принудительно переведён на CPU параметром `--device cpu`, чтобы числа не зависели от наличия ускорителя.

Показатель	Значение	Условия
Время на изображение (медиана)	0.021 с	CPU, включает чтение DICOM, предобработку и проход сети
Время на изображение (максимум)	0.131 с	тот же прогон; первый снимок включает прогрев
Пакет целиком (548 файлов)	12.1 с	от запуска процесса до записи XLSX, включая загрузку чекпоинта
Доля успешно обработанных файлов	548 / 548 (100 %)	ошибок чтения на этом наборе нет
Пиковая память процесса	≈ 640 МБ	maximum resident set size, пакетный CPU-инференс
Ответ API на один файл	16–18 мс	вызов внутри процесса, после прогрева; первый запрос 390 мс. Здесь устройство выбрано автоматически (MPS), на CPU значение того же порядка — см. медиану пакетного прогона выше
Время старта сервиса	≈ 3 с	импорт библиотек и загрузка чекпоинта

Отсюда следует прямой ответ на требование «не более 3 минут на исследование»: даже если исследование содержит максимальные по заданию три снимка, обработка занимает доли секунды, а трёхминутный бюджет расходуется практически только на передачу файлов и накладные расходы интеграции. Запас более чем трёхкратный, поэтому узким местом решение не ограничивает.

13.2. Минимальная и рекомендуемая конфигурация

Ресурс	Минимальная конфигурация	Рекомендуемая
Процессор	2 ядра x86-64 или ARM64	4 ядра и более; инференс векторизован и масштабируется по кадрам
Оперативная память	1.5 ГБ (измеренный пик ≈ 640 МБ + запас на ОС и веб-слой)	4 ГБ
Графический ускоритель	Не требуется: torch устанавливается из CPU-индекса, инференс на CPU укладывается в требование по времени	Любой GPU с поддержкой CUDA 11.8 — вариант сборки <code>Dockerfile_cuda</code>
Дисковое пространство	базовый образ <code>python:3.11-slim</code> (150 МБ) + зависимости и код; чекпоинт 43 МБ монтируется отдельно	плюс место под входные архивы DICOM и результаты
Сеть	Не требуется при работе: образ и веса фиксированы, обращений во внешние сервисы нет	Нужна только для сборки образа и загрузки базовых зависимостей

Размер образа (измерено): 1.41 ГБ (1 412 385 143 байт) для сборки под `linux/arm64`. Базовый образ `python:3.11-slim` занимает 150 МБ, остальное — зависимости из `requirements.txt`, включая `torch` и

torchvision из CPU-индекса; CUDA-колёса в образ не попадают. Данные, тесты, `labels/` и `assets/` в образ не входят, чекпоинт (43 МБ) монтируется отдельно.

14. Сборка, запуск и развёртывание

14.1. Состав образа

Элемент	Значение
Базовый образ	<code>python:3.11-slim</code> (фиксированный тег)
Зависимости	<code>requirements.txt</code> с точными версиями; <code>torch</code> и <code>torchvision</code> — из CPU-индекса PyTorch, чтобы в образ не попали CUDA-колёса
Что копируется в образ	только <code>src/</code> , <code>requirements.txt</code> и <code>run.sh</code>
Что не копируется	данные, тесты, <code>labels/</code> , <code>assets/</code> , <code>docs/</code> (см. <code>.dockerignore</code>), а также чекпоинт
Чекпоинт	монтируется при запуске в <code>/app/models</code> ; путь задаётся переменной <code>DXA_MODEL_PATH</code> (по умолчанию <code>/app/models/dxa_model.pth</code>)
Проверки на этапе сборки	импорт приложения (<code>import src.main</code>) и наличие офлайн-ассетов фронтенда; сборка падает при их отсутствии
Контроль состояния	<code>HEALTHCHECK</code> обращается к <code>/api/v1/health</code> каждые 30 с
Точка входа	<code>uvicorn src.main:app</code> на порту 8000

Если чекпоинт не смонтирован, сервис всё равно поднимается: `/api/v1/health` сообщает `model_loaded: false`, а запросы анализа возвращают ошибку вместо тихой неверной оценки. Это сознательный выбор: отсутствие модели должно быть заметно сразу, а не проявляться как «странные» результаты.

Что проверено на самом образе. Сборка прошла обе внутренние проверки (`app import ok`, `frontend assets ok`). Образ запущен и проверен в двух режимах: без смонтированного чекпоинта `/api/v1/health` отдаёт `model_loaded: false` и `exists: false`, а запрос анализа — HTTP 500 с телом `{"error": "Model not loaded"}`; с примонтированным каталогом моделей `/api/v1/health` сообщает `model_loaded: true`, `device: cpu`, эпоху 39 и файл разметки `labels/labels_images.csv`, запрос анализа отвечает корректной строкой результата, а `/api/v1/export` возвращает XLSX с ожидаемым набором столбцов. Сквозная проверка выполнялась в контейнере на той же машине, где снимались измерения производительности.

14.2. Воспроизводимость

- Версии всех зависимостей зафиксированы, базовый образ задан тегом, версии `torch` и `torchvision` указаны явно.
- Обучение детерминировано по `seed`; разбиение выборки сохраняется в файл и переиспользуется.
- Чекпоинт самодостаточен: внутри хранятся `backbone`, `head`, параметры предобработки, `threshold`, а также `labels_csv` и `split_file`. Поэтому по файлу видно, на какой разметке и на каком разбиении получена модель, а инференс не может рассинхронизироваться с обучением.
- Отказ от зависимости от рабочего каталога: путь к модели и рабочие каталоги задаются переменными окружения.

14.3. Сборочный конвейер

Jenkinsfile выполняет три шага: сборка образа, публикация в реестр и выкладка в кластер. Скрипт `run.sh` — единая точка входа для разработки и эксплуатации: обучение, разметка, приведение имён файлов, инференс, сервис, сравнение вариантов разметки и тесты (перечень команд — приложение Б).

15. Тесты и проверки

Тесты — `pytest`, 209 проверок в шести файлах; запуск — `./run.sh test` или `python -m pytest tests/ -q`. Сверка выполнена на дату документа: 209 passed.

Файл	Тестов	Что проверяет
<code>tests/test_labels.py</code>	38	разбор имён, определение области и метки по имени, стратифицированное разбиение по исследованиям, отсутствие утечки, фиксация и переиспользование разбиения в файле, сверка с реальным датасетом
<code>tests/test_excel_labels.py</code>	50	разметка по экспертной таблице: чтение критериев, правило «1 = нарушение», голосование по области, перенос оценки на единственное бедро, три правила метки (<code>table / union / expert</code>), их согласованность и подключение к обучению
<code>tests/test_rename_files.py</code>	36	приведение имён DICOM: разбор и канонизация, поиск свободного номера при конфликте, отказ угадывать область, цикл «применить → откатить»
<code>tests/test_preprocess_and_model.py</code>	32	конфигурация и нормировка, сборка тензора и нормировка ImageNet, метрики, подбор порога, контракт модели, BatchNorm при заморозке, roundtrip чекпоинта, проверка на реальных снимках
<code>tests/test_violations.py</code>	28	единый словарь типов: приведение устаревших значений, подписи, коды SR и каталог для API, согласованность с критериями экспертной таблицы
<code>tests/test_api_contract.py</code>	25	поля ответов, которые читает веб-интерфейс: базовый и детальный анализ, здоровье и карточка модели, выгрузка XLSX, обработка ошибок; различимость метрик между снимками и валидность PNG-визуализаций

Отдельный контур — браузерные проверки интерфейса (`tests/browser/*.js`, Node и `playwright-core`): они открывают интерфейс во временном профиле Chrome, загружают DICOM, кликают по строкам таблицы и снимают содержимое панели деталей. Требуется запущенный сервер на `127.0.0.1:8123`; профиль пользователя не затрагивается.

Сценарий	Что проверяет
<code>ui_check.js</code>	подсказку о кликабельности строк и её скрытие при пустом фильтре; открытие панели деталей из строки; изменение значений при переключении строк; наполнение панели «О модели» метриками и словарём
<code>ui_violation.js</code>	ветку «нарушение»: бейдж, P00R, HIGH, текст заключения
<code>ui_offline.js</code>	отсутствие обращений страницы к внешним хостам

16. Известные ошибки и их обработка

Требование задания — отсутствие необработанных исключений: сбой на одном файле фиксируется в отчёте и не прерывает пакет. Ниже — фактические режимы отказа и способ обработки.

Ситуация	Поведение	Где обрабатывается
Ошибка чтения или обработки файла (CLI)	строка получает <code>quality_class = -1</code> , область <code>unknown</code> , пустые <code>UID</code> и <code>processing_status = Failure</code> : <Тип>: <сообщение> (до 120 символов); остальные файлы обрабатываются дальше	<code>process_one</code> , <code>src/dxa/inference.py</code>
Неподдерживаемая форма <code>pixel_array</code> (не 2D и не 3D)	<code>ValueError</code> , превращается в <code>Failure</code> : согласно строке выше	<code>preprocess.py</code>
Чекпоинт отсутствует или не читается	сервис поднимается, <code>/api/v1/health</code> отдаёт <code>model_loaded: false</code> , а запросы анализа — HTTP 500 <code>{"error": "Model not loaded"}</code>	<code>load_model()</code> , <code>src/main.py</code>
Архитектура чекпоинта не совпала с запрошенной	<code>ValueError</code> с указанием сохранённых <code>backbone / head</code>	<code>load_model</code> , <code>src/dxa/inference.py</code>
Чекпоинт — «сырой» <code>state_dict</code> без метаданных	<code>ValueError</code> : файл не содержит <code>model_state_dict</code>	<code>DXAQualityModel.load</code>
Сбой в одиночном запросе <code>/analyze</code> , <code>/analyze/detailed</code> , <code>/analyze/sr</code>	HTTP 500 с телом <code>{"error": <текст>, "processing_status": "Failure: <первые 50 символов>"}</code> ; детальный эндпоинт дополнительно печатает стек в лог сервера	обработчики маршрутов
Сбой одного файла в <code>/api/v1/batch</code>	элемент становится <code>{"filename", "error", "processing_status": "Failure"}</code> , остальные файлы возвращаются нормально	<code>batch_analyze</code>
Сбой одного файла в <code>/api/v1/export</code>	строка со статусом <code>Failure</code> : <до 80 символов> и <code>quality_class = -1</code> попадает в XLSX, выгрузка не прерывается	<code>export_results</code>
Файл разметки не найден при обучении	предупреждение в лог и откат на метки из имён файлов; откат залогирован явно, потому что вместе с источником меток меняются метрики	<code>resolve_labels_csv</code> , <code>src/dxa/train.py</code>
Пустое разбиение (нет обучающих или валидационных снимков)	<code>RuntimeError</code> с числами <code>train / val</code> до начала обучения	<code>train()</code>

Сервис без чекпоинта намеренно не пытается «угадывать»: он поднимается, но честно сообщает `model_loaded: false` и отклоняет запросы анализа. Молчаливая неверная оценка вместо явной ошибки была бы опаснее в медицинском контуре.

17. Ограничения и достоверность результатов

Ограничения сформулированы так, чтобы читатель мог оценить границы применимости решения, а не только увидеть итоговые метрики.

1. **Разметка выведена из оценки исследования, а не снимка.** Экспертная таблица описывает исследование; перенос вердикта на снимок однозначен (область встречается один раз), но поштучной экспертной оценки снимков в наборе нет. Оценка качества модели упирается в качество этой разметки, а не только в объём данных.
2. **Мало данных.** 252 уникальных снимка, 77 нарушений. Доверительные интервалы широкие (ROC-AUC 0.6764 [0.6309, 0.7218] по пяти seed'ам), поэтому оценка на закрытом наборе может отличаться.
3. **Эталон — та же таблица.** Независимой истины нет, поэтому сравнение правил разметки частично благоприятствует варианту «только таблица» (раздел 12.3).
4. **В DICOM нет разметки областей интереса.** Ни overlay, ни graphic annotation в файлах нет, поэтому корректность нанесённых областей измерения нельзя проверить прямым сравнением с эталоном: оценивается геометрия видимой зоны.
5. **Тип нарушения определяется признаками снимка, а не обученной моделью.** Модель решает только бинарную задачу, поэтому в выгрузке тип содержателен только для размеченных снимков, а на инференсе эвристика на текущих данных всегда даёт «не уточнён» (раздел 8.2). Для честного мультикласса нужна разметка типов на уровне снимка; пять снимков имеют только код «не уточнён», потому что источник не указывает критерий.
6. **Три снимка размечены по пометке в имени файла** — экспертная таблица их область не оценивала. Такие строки помечены признаком `filename_fallback`.
7. **Сторона бедра в семи исследованиях с единственным снимком не проверяема:** теги `Laterality` пусты, оценка взята из единственного заполненного столбца таблицы.
8. **Порог определения области привязан к текущему оборудованию.** Признак «ширина кадра» безошибочно работает на этом наборе, но при смене аппарата потребует калибровки.
9. **Числовые эвристики панели деталей не калиброваны.** Их пороги рассчитаны на другой масштаб интенсивностей, поэтому в интерфейсе показываются измерения, а не вердикты.
10. **Рабочая точка порога даёт высокий recall при умеренной точности.** На обучающем наборе при пороге, подобранном по F1, модель относит к нарушениям 341 строку из 548 ($\approx 62\%$) при фактической доле нарушений 30.6 %, что согласуется с precision 0.486. Порог выбран в пользу полноты: пропустить непригодное исследование дороже, чем показать лишнее.

18. План развития

- Разметить типы нарушений на уровне снимка и обучить мультилейбл-классификатор — это снимает главное ограничение (тип нарушения определяется признаками, а не моделью).
- Расширить набор до 500+ исследований: доверительные интервалы сузятся, появится возможность честной валидации без пересечения с обучением.
- Заменить признак «ширина кадра» на калибровку по метаданным аппарата, чтобы определение области не зависело от конкретного оборудования.
- Добавить локализацию нарушения (тепловая карта или контур) — в задании это отнесено к дополнительному функционалу, и для него нужна соответствующая разметка.

- Собрать пилотную интеграцию с АРМ: отбор исследований, требующих внимания, и журналирование подтверждений специалиста.

Приложение А. Структура репозитория

bone_2026/	
├─ src/	
│ └─ main.py	FastAPI: маршруты и загрузка модели
│ └─ dxa/	действующий модуль оценки качества
│ └─ preprocess.py	DICOM → тензор (общий путь для обучения и API)
│ └─ model.py	сеть, метрики, подбор порога, чекпоинт
│ └─ train.py	обучение и отчёт (md/json)
│ └─ dataset.py	DXADataset, DataLoader
│ └─ inference.py	пакетный инференс, определение области, визуализация
│ └─ labels.py	разбор имён, метки, склейка дублей, разбиение
│ └─ excel_labels.py	разметка снимков по экспертной таблице
│ └─ rename_files.py	приведение имён DICOM к единому виду
│ └─ violations.py	единый словарь типов нарушений
│ └─ model_card.py	карточка решения для API и интерфейса
│ └─ compare_labels.py	сравнение источников разметки
│ └─ discriminator.py	проверка вклада содержимого снимка
│ └─ render.py	рендер снимков и контактных листов
│ └─ quality/	эвристики: измерения снимка
│ └─ api/static/	веб-интерфейс (офлайн-ассеты)
│ └─ utils/	выбор устройства
├─ labels/	размеченные данные, разбиение, карта переименований
├─ assets/	схемы, скриншоты, обоснование разметки
├─ models/dxa_model.pth	рабочий чекпоинт и отчёт обучения
├─ tests/	pytest + браузерные проверки интерфейса
├─ docs/	настоящий документ (HTML и PDF)
└─ run.sh, Dockerfile, Dockerfile_cuda, Jenkinsfile, requirements.txt	

Приложение Б. Команды

# обучение и разметка	
./run.sh label	разметить датасет по экспертной таблице
./run.sh rename	план приведения имён файлов (--apply для применения)
./run.sh train	обучить модель (режим меток по умолчанию)
./run.sh split && ./run.sh compare	сравнить варианты разметки на одном разбиении
# инференс и сервис	
./run.sh infer "dataset_hack/Для теста" results.xlsx	
./run.sh serve	API и веб-интерфейс на порту 8000
python -m src.dxa.discriminator	проверка вклада содержимого снимка
# проверки	
./run.sh test	pytest (209 тестов)
node tests/browser/ui_check.js	проверки интерфейса (нужен сервер на 127.0.0.1:8123)
# контейнер	
docker build -t dxa-quality .	
docker run -v /path/to/data:/data -v /path/to/models:/app/models -p 8000:8000 dxa-quality	

Документ подготовлен по фактическому состоянию репозитория; машинные отчёты обучения — `models/train_report.json`, обоснование разметки — `assets/labeling.md`.