Multi-label guardrail-классификатор для детектирования небезопасного контента

Архитектура, обучение с асимметричной функцией потерь, сравнительная оценка и оптимизация инференса

Danil Kapustin*
danilkapustin01@gmail.com

Nikita Oblakov
nikobl25@gmail.com

Sabrina Sadiekh
sadsobr7@gmail.com

Evgeniy Kokuykin
evgeniy.kokuykin@raftds.com

* Corresponding author

Сентябрь 2026

Аннотация

Мы представляем ориентированный на промышленную эксплуатацию multi-label классификатор, который выдаёт 15 независимых оценок по категориям — 11 относятся к вреду (насилие, оружие, наркотики, трудоэксплуатация детей, разжигание ненависти и дискриминация, финансовые преступления, киберпреступность, самоповреждение и другие) и 4 являются настраиваемыми тематическими флагами — для текстов на русском и английском языках, включая смешанные и намеренно обфусцированные тексты. Модель объединяет многоязычный трансформерный энкодер (`mmBERT-base`) с 15 параллельными экспертными головами (по одной голове на категорию) и лёгким модулем взаимодействия категорий, моделирующим статистические корреляции между метками. При обучении используются Asymmetric Loss (ASL), явно штрафующая пропуски (false negative) сильнее ложных срабатываний (false positive), подбор порогов по категориям перебором по сетке и экспоненциальное скользящее среднее весов (EMA). Обучение и подбор порогов выполнены на multi-label стратифицированном k-fold разбиении; приводимые в отчёте цифры получены в независимом сравнительном бенчмарке против 9 открытых guardrail- и toxicity-моделей сторонних разработчиков и ещё одной нашей собственной модели (всего 11 участников) на внешнем сбалансированном по безопасности наборе из 7,480 примеров, оценённых как в бинарном режиме (безопасно/небезопасно), так и в режиме multi-label классификации по 14 меткам (13 канонических категорий риска плюс производная метка `Safe`). В бинарном детектировании наша модель находится на одном уровне с сильнейшей открытой моделью — разрыв 0.0044 лежит в пределах выборочной погрешности при данном размере выборки, — тогда как по multi-label F1-macro отрыв велик и распределён по всем категориям (0.7026 против 0.4581; 0.682 по 13 категориям риска без `Safe`); при этом покатегорийное качество варьируется от 0.34 до 0.96, что агрегаты скрывают. Нормативное основание четырёх политико-тематических флагов сформулировано явно. Дополнительно мы приводим бенчмарки serving-слоя для трёх бэкендов (PyTorch+FlashAttention2, ONNX Runtime, TensorRT) на NVIDIA RTX 3090: TensorRT при batch=16 является оптимумом

по задержке, пропускной способности и стоимости (12.5 мс P95, 1345.1 текста/с, \$0.062 за 1 млн запросов), что в 2.9× дешевле наивного фиксированного batch=8; при batch $\geq$32 три бэкенда сходятся и преимущество TensorRT практически исчезает; ни одно из изменений не затрагивает веса модели.

Содержание

1. Введение

Любой продукт, предоставляющий интерфейс ввода пользователям или генеративной модели (чат-боты, ассистенты, поиск с дополненной генерацией), рискует получить небезопасный контент: инструкции о том, как совершить насилие, 18+ контент с участием несовершеннолетних, экстремистскую риторику, финансовое мошенничество и прочее. *Guardrail-классификатор* — это автоматический фильтр, который оценивает входящие и/или исходящие сообщения по фиксированной таксономии до того, как контент дойдёт до пользователя или до другой модели.

1.1 Постановка задачи

Одно сообщение может одновременно нести несколько типов риска — например, и финансовое мошенничество, и киберпреступление. Поэтому мы используем **multi-label** постановку: 15 независимых бинарных ответов на каждый вход, а не бинарная классификация по взаимоисключающему набору категорий.

Целевые категории. 15 выходов — неоднородный набор меток, и в отчёте последовательно различаются две группы.

Категории вреда (11) — контент, вредоносность или противоправность которого широко признаётся в тех юрисдикциях, с которыми мы сталкивались: трудоэксплуатация детей (`Child_exploitation`), насильственные действия (`Violent_actions`), оружие (`Armament`), наркотики (`Drugs`), самоповреждение (`Self_harm`), разжигание ненависти и дискриминация (`Hate_discrimination`), нацизм (`Nazism`), ненасильственные преступления (`Non_violent_crime`), финансовые преступления (`Financial_crimes`), киберпреступления (`Cybercrimes`), порнографические материалы (`Pornographic_materials`).

Категории политики и запрещённых в России тем (4) — нецензурная лексика (`Swear_words`), политика (`Policy`), оскорбление религиозных чувств (`Religion`), ЛГБТ-тематика (`LGBT`). Эти четыре — тематические детекторы: положительная оценка означает, что текст относится к пропаганде ЛГБТ, оскорблению чувств верующих и др., какова бы ни была занимаемая в нём позиция. По умолчанию они включены и выключаются по запросу при развёртывании. Они существуют потому, что пользователи, подпадающие под российское право, несут обязанность, на которую им нужно уметь реагировать: `LGBT` включается там, где применяется статья 6.21 КоАП РФ [1], а `Religion` — там, где применяется статья 148 УК РФ [2].

`Nazism` и `Financial_crimes` — независимые выходы классификатора; они были свёрнуты в `Hate_discrimination` и `Non_violent_crime` соответственно только для сравнения во внешнем бенчмарке (раздел 6.2), эталонные таксономии которого не выделяют их в отдельные классы.

1.2 Ключевые сложности

В таблице 1 перечислены пять свойств задачи, определивших проектные решения в остальной части отчёта, вместе с ценой их игнорирования.

Таблица 1: Ключевые технические сложности и их проектные следствия

Сложность	**Техническая природа**	**Последствие, если игнорировать**
Дисбаланс классов	`Child_exploitation` и `Nazism` встречаются существенно реже, чем `Swear_words`	Такая модель не детектирует самые редкие плохие категории
Состязательная обфускация	Замена символов, транслитерация, переключение раскладки клавиатуры	Простые фильтры по ключевым словам обходятся тривиально
Асимметричная цена ошибки	Пропуск (FN) небезопасного контента дороже ложной тревоги (FP)	Оптимизация обычной accuracy/F1 без асимметрии даёт ложное ощущение безопасности
Многоязычность	Русский, английский и смешанные входы	Одноязычная модель теряет покрытие части продуктового трафика
Экономика serving	Инференс должен быть быстрым и дешёвым при масштабировании	Стоимость неоптимизированного serving растёт линейно с трафиком

1.3 Нормативная рамка и границы таксономии

Guardrail-таксономия — это артефакт политики, а не факт о языке, поэтому мы явно проговариваем, чем является эта конкретная таксономия.

Чья это рамка. 15 категорий — таксономия по умолчанию коммерческого продукта. Она представляет собой объединение того, что просили детектировать клиенты, и того, что правовые режимы, в которых эти клиенты работают, требуют от них уметь детектировать. Четыре категории политик и тем — это конфигурация, а не суждение о вредоносности: каждая поставляется включенной и выключается по запросу.

`LGBT` срабатывает на ЛГБТ-тематический контент независимо от позиции и нужна там, где оператор подпадает под статью 6.21 КоАП РФ [1]; `Religion` включается по статье 148 УК РФ [2].

Конфигурирование при развёртывании. Каждая категория — независимый выход со своим порогом в `thresholds` (раздел 5.3), поэтому развёртывание включает ровно нужное подмножество: категория отключается удалением её колонки из ответа или установкой порога в 1.0, без влияния на остальные выходы и без переобучения. Набор также не замкнут — но добавление категории не является изменением конфигурации: оно требует размеченных данных и запуска обучения (раздел 5), `thresholds` новую категорию ввести не может.

Для чего нужна положительная оценка. Модель выдаёт вероятности по категориям. Она ничего не блокирует, не удаляет и никуда не сообщает. Что происходит при положительной оценке — решение внедряющей системы: логирование, пометка на рассмотрение человеком, маршрутизация в более строгую модель или отказ от генерации, — и конструкция с асимметричной функцией потерь (раздел 5.1) намеренно обменивает ложные срабатывания на меньшее число пропусков, что осмысленно лишь тогда, когда положительная оценка направляет контент на рассмотрение, а не на автоматическую блокировку.

1.4 Вклад работы

- Архитектура параллельных экспертов по категориям поверх общего многоязычного энкодера с трансформерным модулем взаимодействия категорий, корректирующим логиты экспертов с использованием межкатегорийных корреляций (раздел 4);
- Рецепт обучения с Asymmetric Loss, дифференцированными скоростями обучения, усреднением весов EMA и покатегорийным подбором порогов, явно кодирующий асимметрию цены FN/FP (раздел 5);
- Исследование serving-слоя на трёх бэкендах (PyTorch, ONNX Runtime, TensorRT) с измеренными задержками, пропускной способностью и стоимостью, а также политикой маршрутизации по длине входа при фиксированных размерах батча (раздел 7).

Наряду с этим мы приводим сравнительную оценку нашей модели против 9 открытых guardrail- и toxicity-классификаторов сторонних разработчиков и второй нашей собственной модели (всего 11 участников) на внешнем сбалансированном по безопасности наборе из 7,480 примеров, собранном специально для этой цели (раздел 6.2). Оценочный набор не публикуется, и сравнение воспроизводимо по методу, но не по данным; все приводимые для нашей модели цифры получены из этого прогона, а не из отдельно приводимого отложенного разбиения.

2. Обзор существующих решений

Guardrail-модели на базе LLM. Доминирующее в последнее время направление строит классификаторы безопасности поверх генеративных языковых моделей, которым в промпте передают текст политики и просят вынести вердикт: Llama Guard [4], ShieldGemma [5], WildGuard [6] и семейство Qwen3Guard [12, 13, 14] следуют этой схеме. Их сильная сторона — гибкость: таксономия живёт в промпте, поэтому развёртывание может переформулировать её без переобучения. Издержки — экономика инференса (миллиарды параметров на каждое решение по каждому запросу и ответу) и то, что качество классификации по категории ограничено тем, насколько верно понят текст заложенной в промпте политики. Наш подход занимает противоположную позицию по обеим осям: фиксированная таксономия из 15 категорий, обученная в веса энкодера на 307M параметров — на порядок меньше, чем 8B-генеративный guard, — при этом гибкость перенесена из промпта в покатегорийные пороги (раздел 5.3). Сравнение в разделе 6.2 количественно показывает, что даёт этот обмен на нашем целевом распределении: бинарное детектирование на уровне сильнейшего открытого базлайна и большой отрыв в определении того, *какая* категория применима.

Классификаторы токсичности. Это направление нацелено на единственную ось токсичности: Perspective API [8], многоязычные модели Detoxify/`unitary` [15], а для русского языка — `rubert-tiny-toxicity` [16] и `russian-sensitive-topics` [11]. Они существенно дешевле guardrail-LLM и ближе по размеру к нашей модели, но отвечают на другой вопрос — насколько оскорбителен этот текст, — и, как показывает раздел 6.2, практически не покрывают преступления, оружие, наркотики и политические темы. Сильные результаты по токсичности регулярно принимают за guardrail-покрытие; покатегорийные результаты здесь отчасти являются аргументом против такой подмены.

Промышленные системы модерации. Марков и соавт. [7] описывают развёрнутый multi-label классификатор модерации и окружающий его конвейер данных и приходят к выводам, которые мы независимо воспроизводим: связывающим ограничением является покрытие редких категорий данными, а не архитектура, и ответы для категорий должны задаваться по отдельности. Наш вклад относительно этой работы — двуязычный русско-английский сценарий, где code-switching и состязательная обфускация являются первоклассными целями обучения, и исследование serving-слоя, показывающее, во сколько реально обходится конфигурация.

Методы многометочного обучения. Рецепт обучения собран из известных компонентов, а не из новых: Asymmetric Loss [9] для дисбаланса положительных/отрицательных примеров и намеренной асимметрии FN/FP (раздел 5.1) и итеративная стратификация [10] для разбиения multi-label данных без исчезновения редких категорий из отдельных частей (раздел 3). Энкодер — `mmBERT-base` [3]. Специфичным для данной работы является их сочетание в схеме с отдельными экспертными головами на категорию и явным модулем взаимодействия категорий, а также измерение этого сочетания против открытых базлайнов на общем оценочном наборе.

3. Данные

Основное время разработки занял сбор датасета, а не архитектурные улучшения; качество разметки и покрытие редких категорий определили итоговое качество модели (раздел 10).

3.1 Источники и критерии отбора

Датасет построен преимущественно из открыто лицензированных наборов данных на Hugging Face Hub и в других публичных репозиториях, отобранных по четырём критериям: (i) совместимость лицензии с коммерческим использованием; (ii) двуязычность (русский + английский, включая code-switching); (iii) явное покрытие каждой из 15 категорий, поскольку универсальные бенчмарки безопасности хорошо покрывают частые классы, но во многом игнорируют редкие (`Child_exploitation`, `Nazism`, `Self_harm`); (iv) предпочтение наборам с верифицированной или консенсусной разметкой.

Объём и состав датасета. Исходный датасет содержит 1,836,335 примеров: 900,139 безопасных (49.0%) и 936,196 небезопасных (51.0%). Аугментация применяется ко всей обучающей части, а не только к безопасным или небезопасным промптам; eval и test не аугментируются, поэтому их размер не меняется (см. также раздел 3.4).

Синтетические данные. Для категорий с дефицитом открытых данных мы генерировали синтетические примеры с помощью abliterated-моделей (со снятыми механизмами безопасности) при обязательной двухэтапной верификации: сначала AI-судья, затем проверка экспертом-человеком. Синтетика без второго этапа верификации ухудшала метрики на реальных примерах.

3.2 Очистка текста

Очистка намеренно минимальна (таблица 2): агрессивная нормализация уничтожила бы паттерны обфускации, которые модель обучается детектировать.

Таблица 2: Применяемые преобразования очистки

Операция	**Назначение**
Нормализация Unicode NFKC	Приводит визуально схожие символы к единому представлению
Удаление невидимых символов	Пробелы нулевой ширины, мягкие переносы — частые инструменты обхода фильтров
Замена URL на токен `[URL]`	Сохраняет факт наличия ссылки, не раскрывая её содержимое
Удаление номеров карт и подобных идентификаторов	Убирает самые частые носители персональных данных до обучения
Схлопывание пробелов	Нормализация форматирования

3.3 Стратифицированное разбиение

Случайное разбиение неприменимо к multi-label данным: при наличии редких категорий случайное разбиение рискует дать тестовую часть, в которой некоторый класс отсутствует

полностью. Мы используем `MultilabelStratifiedKFold` (`iterative-stratification`), который сохраняет распределение всех 15 категорий в каждой части.

Таблица 3: Целевые пропорции разбиения данных

Часть	**Доля**	**Назначение**
Train	∼80%	Обучение модели
Eval	∼10%	Мониторинг обучения, подбор порогов
Test	∼10%	Отложенная выборка; — см. раздел 6.2

Без стратификации `Child_exploitation` и `Nazism` периодически полностью исчезали из eval-части или были представлены единичными примерами, что делало обучение для этих классов бесполезным.

3.4 Текстовая аугментация

Аугментация нацелена на устойчивость к обходу фильтров: вместо того чтобы латать обфускацию постфактум, мы включили обфусцированные примеры в обучение. В таблице 4 перечислены применяемые семейства преобразований и техника обхода, которую каждое из них имитирует.

Таблица 4: Реализованные аугментации

Аугментация	**Пример**	**Что имитирует**
`random_char_replace`	drug → dr0g	Замена визуально похожими символами
`random_char_insert`	kill → k·ill	Вставка невидимых/нейтральных символов
`wrong_layout`	*как дела* набрано в EN-раскладке → `rfr ltkf`	Набор кириллического текста в латинской раскладке
`transliterate`	*как убить* → `kak ubit`	Запись кириллического текста латиницей для обхода фильтров
`to_lower/upper case`	КАК УБИТЬ → как убить	Смена регистра
`random_case`	как убить → КаК уБиТь	Случайный регистр
`informal_retelling`	приобрести → достать	Замена формальной лексики сленгом

В ранней версии аугментации применялись только к небезопасным примерам. Это искажало обучающий сигнал: модель выучивала обфусцированный (например, набранный капсом) *безопасный* текст и потом говорила, что текст небезопасный. Равномерное применение аугментации и к безопасным, и к небезопасным примерам сняло проблему.

4. Архитектура модели

4.1 Общая схема

Модель — это один прямой проход в четыре стадии (рисунок 1). Общий многоязычный энкодер превращает входной текст в скрытые состояния. Пятнадцать экспертов по категориям читают эти состояния параллельно, при этом каждый эксперт выдаёт логит для своей категории и представление категории; поскольку эксперты независимы, редкой категории не нужно соревноваться с частой (раздел 4.3). Затем двухслойный модуль взаимодействия категорий читает все пятнадцать представлений вместе и выдаёт поправку для каждой категории, тем самым учитывая информацию о взаимосвязях между метками: контент об оружии обычно встречается в запросах о насильственных действиях, нацизм обычно является и разжиганием ненависти (раздел 4.4). Прибавление поправки к логитам экспертов и применение сигмоиды дают 15 независимых вероятностей, которые сравниваются с покатегорийными порогами из раздела 5.3.

Два следствия этой схемы важны далее в отчёте. Эксперты работают как один батчированный прямой проход, а не как цикл по категориям, — именно это делает 15 голов приемлемыми по стоимости при serving; а наличие у каждой категории собственного выхода и порога позволяет при развёртывании отключить одну категорию, не затрагивая остальные (раздел 1.3).

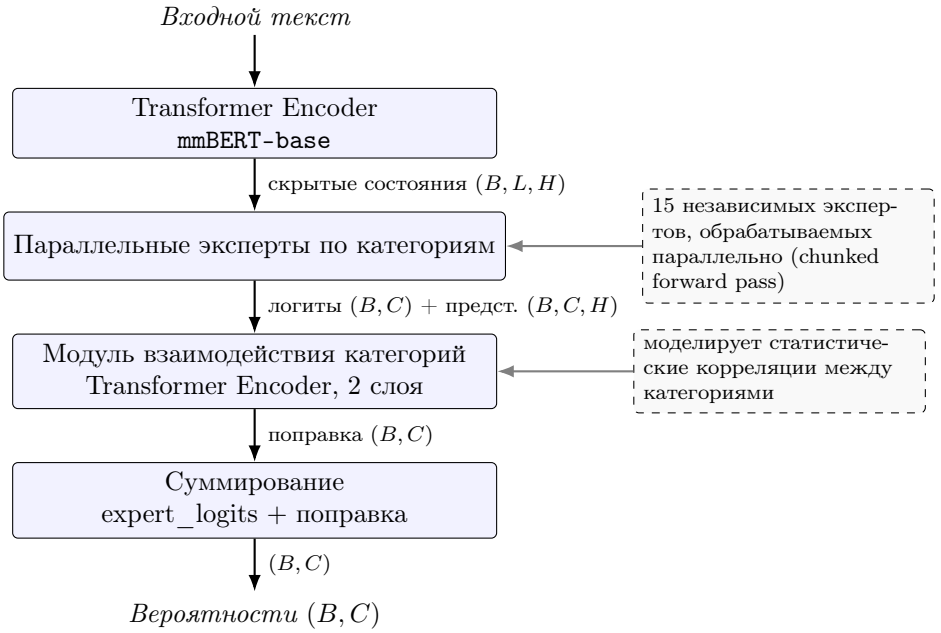

Рис. 1: Архитектура модели: от исходного текста к вероятностям по категориям

4.2 Backbone и длина контекста

Энкодер: `jhu-clsp/mmBERT-base` [3], многоязычная модель семейства BERT. Мы установили ограничение контекста в 1024 токена на основе эмпирического распределения длин обучающего корпуса (таблица 5): половина текстов короче ∼43 токенов, но хвост растёт

нелинейно — P99 составляет ∼555 токенов, а P99.5 — ∼714, — поэтому 1024 покрывает хвост, не заставляя платить за более длинный контекст на каждом запросе.

Таблица 5: Перцентили длины текстов обучающего корпуса (токены); хвост растёт нелинейно, что и мотивирует `max_length=1024`

Перцентиль	Токены
P50	∼43
P75	∼87
P90	∼184
P95	∼279
P99	∼555
P99.5	∼714

При `max_length=1024` покрывается 99.78% полного датасета из 1,836,335 примеров (раздел 3), из которых обучающая часть составляет ∼80% (таблица 3); оставшиеся 3,968 текстов (0.22%) превышают 1024 токена и все являются безопасными, поэтому они полностью исключаются из обучения, а не усекаются до нужной длины. Поскольку каждый исключённый пример безопасен, такое исключение по построению не может снизить полноту по небезопасным категориям, и оно ускоряет и обучение, и инференс.

Следствие исключения вместо усечения: каждый обучающий пример либо полон ($\leq$1024 токенов), либо отсутствует — модель никогда не обучается на усечённом входе. На инференсе длинные запросы также не усекаются: запрос длиннее 1024 токенов разбивается на перекрывающиеся окна по 1024 токена (перекрытие 128 токенов с каждой стороны), каждое окно оценивается как отдельный вход (по одному или батчем), и вердикт по исходному запросу берётся как максимум вероятности по каждой категории среди окон — так что контент не теряется, а небезопасный фрагмент в каком-либо одном окне не размывается остальными. При этом каждое окно всё равно оценивается без окружающего документного контекста, который был бы у полного короткого текста, а обучение никогда не показывало модели окно из середины документа такого рода, поскольку длинные примеры были исключены целиком, а не разбиты на окна.

4.3 Параллельные эксперты по категориям

Каждую из 15 категорий обрабатывает выделенный эксперт с $Q = 4$ обучаемыми query-токенами; эксперты выполняются параллельно за один прямой батч проход (с разбиением на чанки для ограничения пикового потребления памяти). Это даёт ∼3× ускорение относительно последовательного цикла по категориям на GPU.

В первой версии для всех категорий использовалась общая классификационная голова; переход на независимых экспертов повысил F1-macro на eval-части, наиболее заметно на редких категориях, поскольку общая голова не могла сформировать достаточно специализированные представления для классов с малым числом обучающих примеров. Изменение было сделано на раннем этапе разработки одновременно с изменениями в данных, поэтому мы не приводим его как контролируемую абляцию: цифры, на которых стоит остальная часть отчёта, — сквозные (таблица 11), а не атрибуции по компонентам.

4.4 Модуль взаимодействия категорий

`CategoryInteraction` (двухслойный Pre-LN Transformer Encoder) моделирует статистические корреляции между категориями и добавляет поправочный коэффициент к логитам экспертов. В таблице 6 приведены примеры структуры совстречаемости, которую он должен использовать.

Таблица 6: Примеры межкатегорийных корреляций, наблюдаемых в данных

Категория A	**Категория B**	**Характер связи**
`Armament`	`Violent_actions`	Сильная положительная
`Drugs`	`Financial_crimes`	Умеренная положительная
`Child_exploitation`	`Non_violent_crime`	Сильная положительная
`Nazism`	`Hate_discrimination`	Сильная положительная

«Характер связи» — качественная экспертная оценка совстречаемости категорий в данных, без вычисленного статистического коэффициента. «Сильная положительная» означает, что категории систематически встречаются вместе заметно чаще, чем по отдельности; «умеренная положительная» указывает на заметную, но менее устойчивую тенденцию к совстречаемости.

5. Обучение

5.1 Функция потерь

Стандартная BCE неэффективна при сильном дисбалансе классов. Мы сравнили три подхода (таблица 7).

Таблица 7: Сравнение функций потерь

Функция потерь	**Сильные стороны**	**Слабые стороны**	**Итог**
BCE	Простота	Плохо работает при дисбалансе	Бейзлайн
Focal Loss	Фокус на сложных примерах	Ручной подбор γ	Уступает ASL на редких классах
ASL	Встроенная работа с дисбалансом, асимметрия FN/FP	Три гиперпараметра	**Основной выбор**

Asymmetric Loss (ASL):

$$\mathcal{L}_{\text{ASL}} = -\left[\tilde{y}(1-p)^{\gamma_+}\log(p) + (1-\tilde{y})\,p_m^{\gamma_-}\log(1-p_m)\right] \tag{1}$$

где $p_m = \max(p-\delta,\,0)$ — предсказанная вероятность, сдвинутая вниз на δ и отсечённая нулём, а $\tilde{y} = y(1-\varepsilon)+\varepsilon/2$ — метка после сглаживания. Гиперпараметры: $\gamma_- = 4.0$, $\gamma_+ = 1.0$, $\delta = 0.05$, $\varepsilon = 0.02$.

Для правильного негативного примера ($y = 0$, p близко к 0) сдвиг на δ устремляет $p_m \to 0$, что обнуляет и фокусирующий множитель $p_m^{\gamma_-}$, и $\log(1-p_m)$, подавляя его вклад в функцию потерь; для *неправильного* отрицательного примера (p близко к 1) p_m остаётся большим, и вклад в потери остаётся большим. В этом и состоит асимметрия: мы агрессивнее подавляем (множество) простых негативных примеров, чем положительные, поэтому модель терпит больше ложных срабатываний в обмен на меньшее число пропусков.

Формула была сверена с реализацией в обучении `AsymmetricLoss`: код применяет сдвиг на δ к $1-p$ (с отсечением сверху единицей), а не к p напрямую, но обе параметризации алгебраически идентичны — $\min((1-p)+\delta,\,1) \equiv 1-\max(p-\delta,\,0)$ для всех $p \in [0,1]$ — и было подтверждено, что они дают численно совпадающие значения потерь на случайных входах.

5.2 Оптимизация

Backbone (дообучали) и экспертные головы (обучали с нуля) используют дифференцированные скорости обучения; их значения приведены в таблице 8 совместно с остальными гиперпараметрами обучения.

Единая скорость обучения для backbone и экспертов давала две проблемы: при высоком LR дестабилизировался backbone, при низком — эксперты сходились слишком медленно. Дифференцированные скорости обучения устранили обе проблемы одновременно.

Таблица 8: Ключевые гиперпараметры обучения

Параметр	**Значение**	**Обоснование**
LR backbone	3e-5	Дообучение предобученных весов
LR экспертных голов	5e-4	Обучение с нуля
Размер батча	44	Баланс стабильности градиента и VRAM
Эпохи	12 (ранняя остановка, patience=5)	Предотвращение переобучения
Точность	BF16	Широкий динамический диапазон

5.3 Регуляризация и подбор порогов

EMA ($\alpha = 0.999$) стабильно превосходит сырые веса последней эпохи: при сравнении двух наборов весов одного и того же прогона на eval-части (раздел 3) F1-macro по 15 нативным категориям выше на 0.5–2 пункта для EMA-весов. Это внутреннее сравнение двух чекпоинтов одного обучающего прогона на eval-части, а не цифра из внешнего бенчмарка раздела 6.2. Мы сохраняем EMA-веса в чекпоинты и на инференсе загружаем преимущественно их.

После каждой эпохи мы независимо подбираем оптимальный порог для каждой из 15 категорий перебором по сетке на eval-части (шаг 0.01, оптимизация F1). Это именно подбор порогов, а не калибровка вероятностей: конвейер не применяет ни temperature scaling, ни какую-либо иную калибровку выходных вероятностей модели.

Прирост F1 на редких категориях в 10–30 пунктов от покатегорийных порогов (раздел 10) — это разница между F1 данной редкой категории при едином глобальном пороге 0.5 и её F1 после индивидуального перебора по сетке, обе величины измерены на eval-части, — а не эффект того, насколько итоговый порог отличается от 0.5. Подобранные пороги сосредоточены в узком диапазоне 0.49–0.62 (таблица 9), однако для категорий с малым числом положительных примеров F1 крайне чувствителен к небольшим сдвигам порога: знаменатель метрики мал, поэтому те несколько предсказаний, у которых сдвиг на сотые доли меняет TP/FP/FN, двигают F1 на целые пункты — при большем числе положительных примеров те же несколько предсказаний растворились бы в округлении.

Пороги хранятся в `thresholds` отдельно от весов модели. Операционно это самый быстрый доступный путь коррекции: систематическая ошибка, замеченная в продакшене — категория срабатывает слишком часто или слишком редко, — устраняется правкой одного числа и перезагрузкой конфигурации, без цикла переобучения; этим же способом развёртывание отключает ненужную ему категорию (раздел 1.3). Изменение порогов не может исправить категорию, которую модель не знает, но помогает менять точность на полноту вдоль фиксированной кривой, а не сдвигать саму кривую.

Таблица 9: Пороги решения по категориям (`thresholds.json`)

Категория	Порог
`Hate_discrimination`	0.54
`Violent_actions`	0.55
`Armament`	0.53
`Drugs`	0.62
`Nazism`	0.58
`Self_harm`	0.55
`Child_exploitation`	0.57
`Swear_words`	0.53
`Financial_crimes`	0.54
`Non_violent_crime`	0.56
`Pornographic_materials`	0.54
`Policy`	0.52
`Cybercrimes`	0.49
`Religion`	0.55
`LGBT`	0.56

6. Оценка качества

6.1 Метрики

Пусть TP_c, FP_c, FN_c обозначают истинно положительные, ложноположительные и ложноотрицательные предсказания для категории c, соответственно. Тогда

$$P_c = \frac{TP_c}{TP_c + FP_c}, \qquad R_c = \frac{TP_c}{TP_c + FN_c}, \qquad F1_c = \frac{2P_cR_c}{P_c + R_c} = \frac{2TP_c}{2TP_c + FP_c + FN_c}.$$

- **F1-micro:** TP_c, FP_c, FN_c суммируются по категориям до вычисления единого F1:

$$F1_{\text{micro}} = \frac{2\sum_c TP_c}{2\sum_c TP_c + \sum_c FP_c + \sum_c FN_c}.$$

  Эквивалентно объединению всех пар «текст–категория» в одну бинарную задачу; результат определяется категориями, у которых больше положительных меток и больше ошибок.

- **F1-macro:** покатегорийные $F1_c$ усредняются без весов:

$$F1_{\text{macro}} = \frac{1}{C}\sum_{c=1}^{C} F1_c.$$

  Каждая категория весит одинаково независимо от частоты, что делает метрику чувствительной к деградации на редких классах.

- **F1-weighted:** покатегорийные F1 усредняются с весами $n_c = TP_c + FN_c$ (поддержка):

$$F1_{\text{weighted}} = \frac{\sum_c n_c F1_c}{\sum_c n_c}.$$

  Сохраняет покатегорийную детализацию, но определяется частыми категориями. F1-

weighted не обязан лежать между F1-micro и F1-macro, поскольку три агрегата по-разному объединяют ошибки.

Замечание о `Safe`. В исходной модели `Safe` не является 16-м независимым выходом; он определяется как отсутствие всех 15 категорий и исключён из multi-label агрегатов. В кросс-модельном бенчмарке ниже (раздел 6.2) `Safe` добавляется как 14-я колонка и включается в приводимые F1-micro/F1-macro.

Таблица 10: Отслеживаемые метрики и назначение каждой (формулы выше)

Метрика	**Зачем отслеживается**
Полнота (по категории)	Доля действительно небезопасного контента этой категории, которую система ловит; 1 − Recall — доля пропусков
Точность (по категории)	Нагрузка, которую ложные срабатывания создают на эскалацию и рассмотрение человеком
FPR	Доля легитимных пользователей, заблокированных без основания: UX и отток
multi-label F1-micro	Общий уровень; определяется частыми категориями
multi-label F1-weighted	Качество категорий на реальных данных
multi-label F1-macro	Равный вес каждой категории; падает при деградации редких
AUC / mAP	Качество ранжирования независимо от выбранного порога

В таблице 10 перечислено то, что мы отслеживаем во время обучения и в продуктовом мониторинге. Три multi-label агрегата стоит читать вместе, а не по отдельности: они вычисляются из одних и тех же покатегорийных счётчиков и различаются только способом их объединения, поэтому расходятся ровно там, где это существенно, — F1-micro и F1-weighted держатся на частых категориях, тогда как F1-macro падает, как только деградирует редкая. Следовательно, модель может выглядеть здоровой по двум метрикам из трёх и при этом отказывать на категориях с наивысшей ценой пропуска — именно этот случай раздел 6.2 показывает для нескольких открытых бейзлайнов.

F1-weighted подходит для мониторинга качества на реальном продуктовом трафике; F1-macro — для равновесного мониторинга по всем 15 категориям. Полноту по высокорисковым категориям (`Child_exploitation`, `Nazism`, `Hate_discrimination`, `Self_harm`, `Financial_crimes`) следует отслеживать на отдельном дашборде, поскольку агрегированные метрики способны маскировать деградацию в высокорисковых категориях.

Все цифры, приводимые для нашей модели — как агрегированные, так и покатегорийные, — получены из прогона бенчмарка раздела 6.2: таблица 11 для агрегированных (бинарный F1 0.9588; F1-micro 0.7403, F1-weighted 0.7987 и F1-macro 0.7026 по 14 меткам бенчмарка) и рисунок 3 для покатегорийного F1. Относительные приросты, приводимые в разделах 5.3 и 10, — это дельты, измеренные на eval-части в ходе разработки, и они несопоставимы с этими цифрами.

Исходная таксономия из 15 категорий (раздел 1) — это то, что выдаёт развёрнутая модель и на чём в ходе разработки вычислялись приведённые выше обучающие метрики; бенчмарк сводит её к 13 категориям риска плюс `Safe`, чтобы другие модели можно было оценить по тем же категориям, и цифры, указанные в отчёте F1-macro, относились к этим 14 категориям бенчмарка.

6.2 Сравнительный бенчмарк: наша модель против открытых guardrail-моделей

Мы провели единое сравнение двух наших собственных моделей — HiveTrace Multilabel (на базе mmBERT, закрытая) и `gliner_guard_omni` (вторая наша модель на базе GLiNER, выпущенная открыто) — против 9 открытых guardrail- и toxicity-моделей сторонних разработчиков на внешнем оценочном наборе из 7,480 примеров, сбалансированном по безопасности: 3,740 безопасных и 3,740 небезопасных. Приведённые ниже бинарные метрики отражают этот баланс. HiveTrace Multilabel предсказывает 15 категорий, но у остальных 10 моделей нет отдельных классов для `Nazism` и `Financial_crimes`. Исключительно ради кросс-модельной сопоставимости мы свели таксономию к 13 каноническим категориям риска. Две категории были влиты в другие: `Nazism` в `Hate_discrimination` и `Financial_crimes` в `Non_violent_crime`. Ещё пять категорий были переименованы без изменения смысла, чтобы соответствовать наименованиям бенчмарка: `Violent_actions` → `Violence`, `Armament` → `Weapons`, `Policy` → `Politics`, `Cybercrimes` → `Cybercrime`, `Pornographic_materials` → `Sexual_content`. Оставшиеся восемь категорий (`Drugs`, `Self_harm`, `Child_exploitation`, `Swear_words`, `Non_violent_crime`, `Hate_discrimination`, `Religion`, `LGBT`) сохраняют исходные названия. Именно эти названия категорий используются в таблицах и на рисунках данного подраздела (таблица 11, рисунок 3); в остальной части отчёта используются исходные названия категорий (раздел 1). Мы добавили категорию `Safe` (отсутствие всех категорий риска) как 14-ю колонку; приводимые multi-label F1-micro и F1-macro агрегированы по этим 14 колонкам. Это затрагивает только сравнение: оно не меняет ни архитектуру, ни обучение, ни выходы какой-либо из моделей. Отметим, что «категория риска» здесь — собственная терминология бенчмарка, применяемая единообразно ко всем 14 колонкам, чтобы модели с разными таксономиями можно было оценить вместе; она не переносит различие вреда и политики из раздела 1. Мы использовали два режима оценки:

- **Бинарный**: способен ли классификатор отделить безопасный запрос от небезопасного вне зависимости от категории;
- **multi-label по 14 меткам бенчмарка**: 13 категорий риска плюс `Safe`, совместно охватывающие и детектирование небезопасности, и определение типа риска.

Откуда взялись 7,480 примеров. Оценочный набор внешен по отношению к обучающему конвейеру: мы собрали и составили его самостоятельно специально для оценки, отдельно от корпуса раздела 3, и он не пересекается с train-, eval- и test-частями этого корпуса. Каждый пример размечали три независимых аннотатора, разногласия разрешались большинством голосов; согласие между тремя составило κ Фляйсса $= 0.82$. Набор не публикуется. Два свойства набора определяют, как следует читать его цифры. Он сбалансирован по безопасности (3,740 / 3,740), а не воспроизводит продуктовую распространённость, поэтому бинарная точность и любые основанные на ней оценки стоимости не переносятся на живой прогон, где небезопасного контента очень мало. И у каждого примера есть роль «запрос/ответ», по которой разделены две последние колонки таблицы 11. Все 11 моделей оцениваются ровно на одних и тех же примерах при одинаковом приведении к канонической

таксономии, поэтому сравнение является парным: различия между моделями не смешаны с различиями выборок — только с тем, как каждая модель была отображена на общий набор меток.

Сравниваемые модели: `apanc_russian_sensitive_topics` [11], HiveTrace Multilabel[1], `Qwen3Guard_0.6B` [12], `Qwen3Guard_4B` [13], `Qwen3Guard_8B` [14], `unitary_multilingual_toxic` [15], `rubert_tiny_toxicity` [16], `gliner_guard_omni`[2] [17], `YuFengXGuard_0.6B` [18], `gliguard_300m` [19], `opir_multitask_multilang` [20].

Таблица 11: Качество предобученных моделей на едином тестовом наборе (n = 7,480; баланс safe/unsafe — 3,740/3,740)

Модель	Бинарный F1	F1-micro (14 меток)	F1-weighted (14 меток)	F1-macro (14 меток)	Бин. F1 (запрос)	Бин. F1 (ответ)
HiveTrace Multilabel (наша)	**0.9588**	**0.7403**	**0.7987**	**0.7026**	0.9598	0.9578
`YuFengXGuard_0.6B`	0.9544	0.7197	0.6625	0.4581	0.9490	**0.9597**
`Qwen3Guard_8B`	0.9337	0.5700	0.6683	0.3952	0.9132	0.9536
`Qwen3Guard_4B`	0.9327	0.5676	0.6678	0.3923	0.9103	0.9541
`Qwen3Guard_0.6B`	0.9235	0.5514	0.6470	0.3746	0.8925	0.9529
`gliner_guard_omni` (наша)	0.9076	0.7111	0.7006	0.5489	0.9189	0.8958
`opir_multitask_multilang`	0.8718	0.2824	0.2564	0.2735	0.9026	0.8440
`gliguard_300m`	0.8025	0.5563	0.5268	0.2395	0.7545	0.8450
`apanc_russian_sensitive_topics`	0.7295	0.4792	0.4997	0.4149	0.7337	0.7255
`rubert_tiny_toxicity`	0.5455	0.5254	0.4114	0.1270	0.5855	0.5035
`unitary_multilingual_toxic`	0.1304	0.5183	0.3584	0.0927	0.0906	0.1685

В бинарном детектировании safe/unsafe HiveTrace Multilabel (0.9588) находится на одном уровне с сильнейшим открытым базлайном `YuFengXGuard_0.6B` (0.9544, разрыв 0.0044); `YuFengXGuard_0.6B` даже слегка опережает по бинарному F1 только на ответах (0.9597 против 0.9578). Лучшая разница — в multi-label оценке: HiveTrace Multilabel достигает F1-micro = **0.7403**, F1-weighted = **0.7987**, F1-macro = **0.7026**, то есть существенно надёжнее определяет, *какая* категория применима, а не только небезопасен ли вход. Среди сторонних моделей ближе всех по F1-macro `YuFengXGuard_0.6B` (0.4581); `gliner_guard_omni` (F1-macro 0.5489) исключена из этого сравнения как наша вторая собственная модель. Одного бинарного F1 недостаточно для выбора модели: `Qwen3Guard_8B` достигает бинарного F1 = 0.9337 при multi-label F1-macro всего 0.3952. Выбор продуктовой guardrail-модели требует совместного рассмотрения бинарного детектирования риска, multi-label агрегатов без `Safe` и покатегорийного качества.

Неопределённость выборки. Все цифры таблицы 11 получены на одной выборке из n = 7,480, и доверительные интервалы для этого прогона мы не вычисляли; далее приводится оценка правдоподобия, а не измеренный интервал. Для объединённой бинарной доли около 0.95 нормальное приближение даёт стандартную ошибку $\sqrt{p(1-p)/n} \approx 0.0023$, то есть 95%-ю полуширину примерно ±0.005. Разрыв в бинарном F1 в 0.0044 между HiveTrace Multilabel (0.9588) и `YuFengXGuard_0.6B` (0.9544) лежит внутри этой полосы, как и разрыв 0.0019 в обратную сторону на ответах: в бинарном детектировании обе модели находятся на одном уровне, и ни один из порядков не следует считать установленным. multi-label разрыв — другого порядка: F1-macro 0.7026 против 0.4581, разница 0.2445. Макроусреднение по

[1]Наша собственная модель, описываемая в этом отчёте; публичного репозитория нет.

[2]Также наша собственная модель: другая архитектура (на базе GLiNER, zero-shot), выпущена открыто, в отличие от HiveTrace Multilabel.

14 меткам несёт существенно большую выборочную ошибку, чем одна объединённая доля, поскольку определяется самыми разреженными колонками, — но не такого размера, чтобы закрыть разрыв, на два порядка превышающий объединённую стандартную ошибку, а рисунок 3 показывает, что разрыв распределён по категориям, а не сосредоточен в одной-двух. Покатегорийный F1 для самых редких категорий опирается на малое число положительных примеров и несёт соответственно широкие интервалы; эти цифры индикативны, и небольшие покатегорийные различия между моделями ранжировать не следует. Парный бутстрэп по общему набору примеров превратил бы эти оценки в измеренные интервалы и является идеей для будущей работы.

Рисунок 2 показывает, где находятся бинарные ошибки каждой модели, что колонка F1 сама по себе скрывает: модели Qwen3Guard почти целиком ориентированы на точность (precision 0.9997 при recall 0.876 для `Qwen3Guard_8B`) — они практически никогда не поднимают ложную тревогу и платят за это пропусками, — тогда как HiveTrace Multilabel находится в точке precision 0.9725 / recall 0.9455, то есть в том балансе, ради которого и выбиралась асимметричная функция потерь.

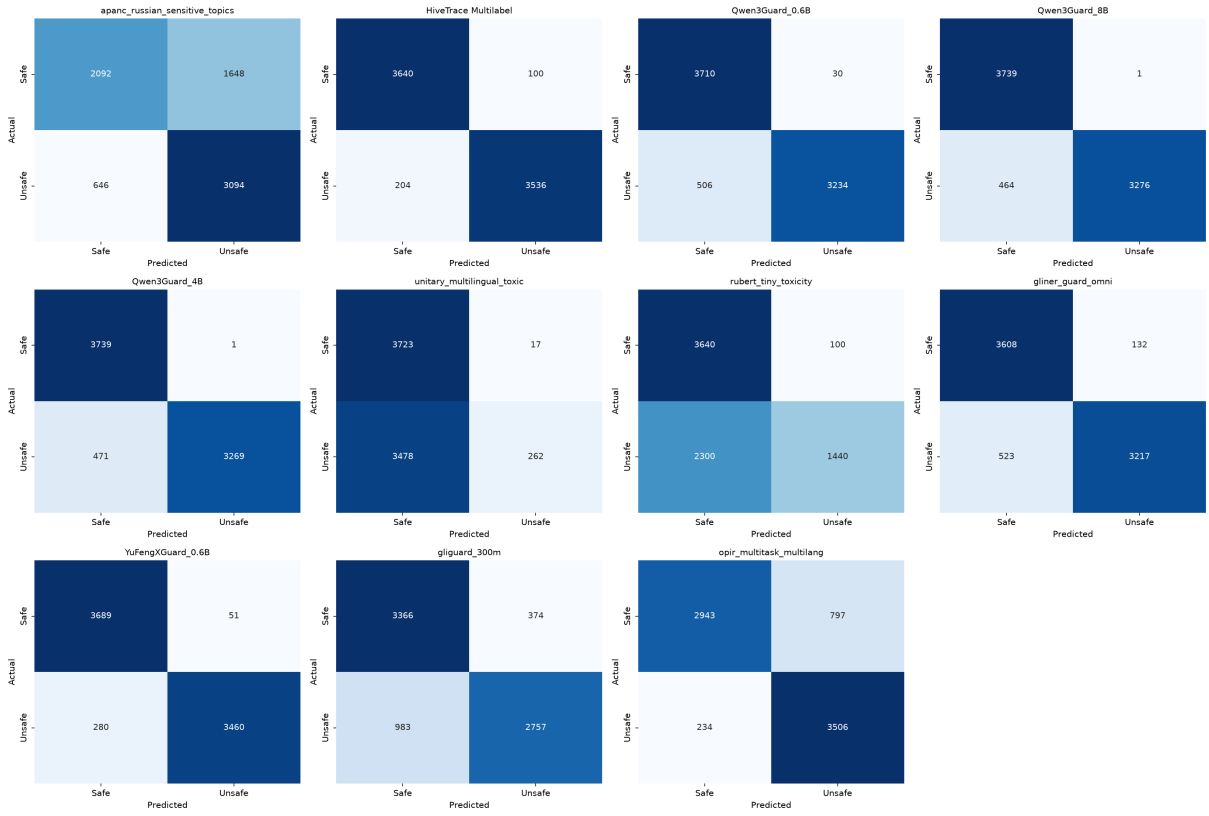

Рис. 2: Бинарные (Safe/Unsafe) матрицы ошибок для всех 11 моделей на едином тестовом наборе

Ни одна модель не доминирует в покатегорийном разрезе, а покатегорийный профиль нашей собственной модели существенно более неровный, чем следует из её агрегатов. HiveTrace Multilabel покрывает каноническую таксономию наиболее равномерно из 11 моделей, но «равномерно» здесь относительно: F1 попадает в диапазон 0.73–0.96 на семи из 14 меток бенчмарка, и одна из этих семи — `Safe`, то есть в этот диапазон попадают шесть из 13 категорий риска. Остальные семь категорий риска лежат ниже: `Weapons` 0.68, `Religion` 0.62, `Politics` 0.58, `Cybercrime` 0.56, `Hate_discrimination` 0.56, `Violence`

0.48, `Non_violent_crime` 0.34. Две из них заслуживают явного упоминания, а не строчки на рисунке. `Hate_discrimination` с 0.56 входит в список категорий высокого риска для мониторинга из раздела 6 — ровно тот случай, ради которого существуют дашборды покатегорийной полноты; а `Non_violent_crime` с 0.34 — категория бенчмарка, поглощающая `Financial_crimes` при описанном выше сведении таксономии, поэтому часть её слабости объясняется слиянием, а не самим классификатором.

Отбрасывание производной метки `Safe` меняет заголовочную цифру: macro-F1 только по 13 категориям риска равен **0.682** против значения 0.7026, приведённого по 14 меткам в таблице 11. `Safe` — самая лёгкая метка набора для любой модели, которая её вообще предсказывает, и она соответственно поднимает метрику macro у всех моделей — поэтому цифру только по категориям риска стоит читать рядом с ней.

Среди остальных моделей `YuFengXGuard_0.6B` лучшая по наркотикам (0.95), `gliner_guard_omni (наша)` — по нецензурной лексике (0.95), а семейство Qwen3Guard особенно сильно на `Safe` (0.95–0.97). Узкие модели, ориентированные только на токсичность, в основном не покрывают категории преступлений, оружия, наркотиков и политики, поэтому сильные результаты по одной оси токсичности не следует читать как полное guardrail-покрытие.

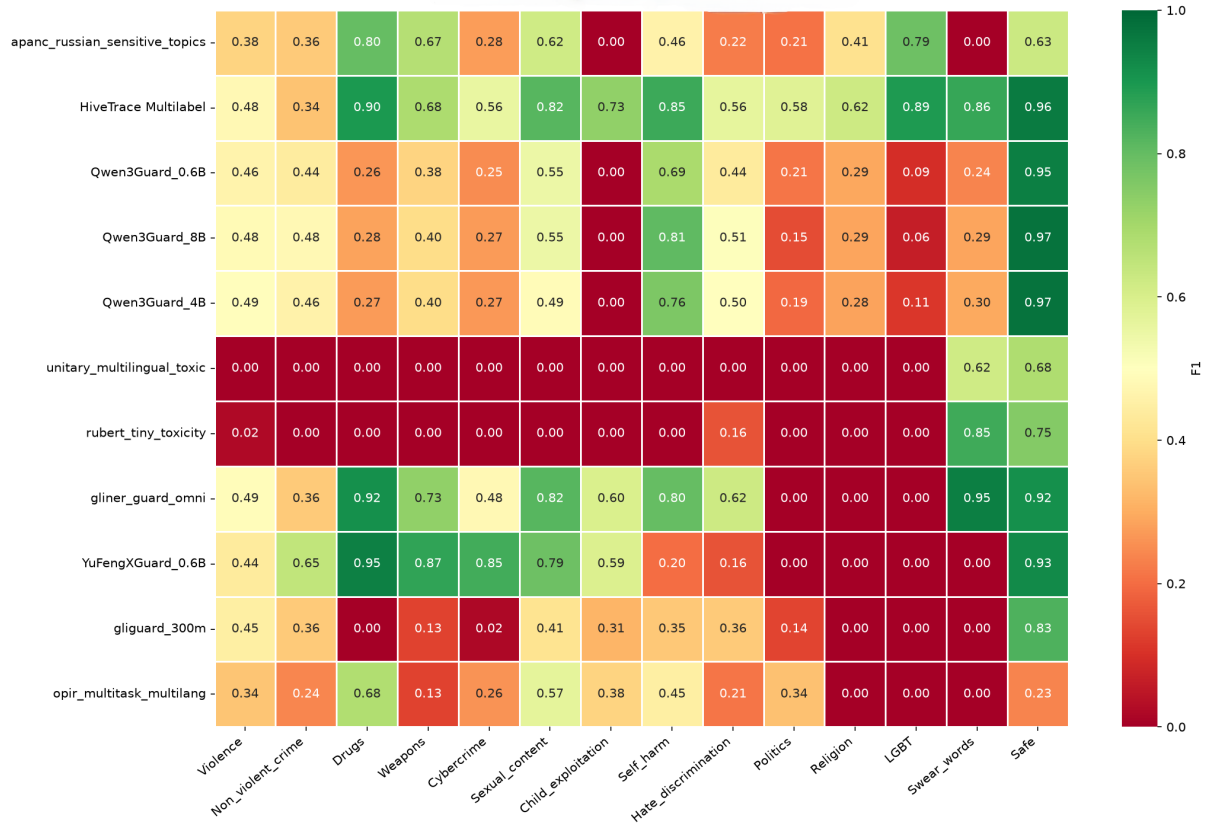

Рис. 3: F1 по 13 каноническим категориям риска и производной метке `Safe`

Разделение по типу сообщения также выявляет систематические различия. Qwen3Guard и `gliguard_300m` классифицируют ответы лучше, чем пользовательские запросы (разрывы до 0.060 и 0.091), тогда как `opir_multitask_multilang` и `rubert_tiny_toxicity` лучше работают на запросах. HiveTrace Multilabel демонстрирует наименьший разрыв (0.9598 на

запросах против 0.9578 на ответах), что указывает на устойчивость к положению текста в диалоге.

Матрица согласия (рисунок 4) показывает, что разные семейства моделей принимают существенно разные решения даже после приведения таксономий. Согласие максимально внутри семейства Qwen3Guard (попарный Jaccard 0.385–0.434). Низкое согласие между большинством прочих моделей подтверждает, что выбор guardrail нельзя основывать на одном лишь списке категорий: командам необходимо валидировать модели-кандидаты на целевом распределении данных.

Рис. 4: Согласие предсказаний по Жаккару после приведения к канонической таксономии

Бинарных метрик достаточно для грубого разделения safe/unsafe; управляемая модерация и покатегорийные политики требуют совместного использования F1-weighted, F1-macro и покатегорийной полноты. F1-weighted отражает качество при реальном распределении категорий, тогда как F1-macro не даёт частым классам маскировать деградацию на редких. Эти результаты говорят в пользу раздельной валидации входящих запросов и исходящих ответов, поскольку одна и та же модель может по-разному работать на этих двух потоках.

7. Оптимизация инференса

7.1 Архитектура serving-слоя

Реализованы три бэкенда для инференса (таблица 12) с автоматическим выбором бэкенда для каждого запроса.

Таблица 12: Бэкенды инференса

Бэкенд	**Характеристики**
PyTorch + FlashAttention2	Универсальный, работает на CPU и GPU, базовый
ONNX Runtime	Backbone на ONNX Runtime (CUDA EP) + головы на PyTorch
TensorRT	Backbone на TensorRT (многопрофильный) + головы на PyTorch; быстрейший на коротких и средних последовательностях

ONNX и TensorRT автоматически откатываются на PyTorch+FA2 на длинных последовательностях, где они не дают выигрыша (пороги: 128 токенов для ONNX, 256 токенов для TensorRT). Маршрутизация выполняется только по длине входа; размеры батча фиксированы, и serving-путь не собирает батчи динамически по длине.

7.2 Методология измерений

Все цифры этого раздела получены за один измерительный эксперимент при следующих условиях.

- **Аппаратное и программное обеспечение:** одна NVIDIA GeForce RTX 3090 (23.6 ГБ VRAM), CUDA 12.4, PyTorch 2.4.1, TensorRT и ONNX Runtime с CUDA execution provider; инференс в BF16; иной нагрузки на GPU нет.
- **Конфигурации:** три бэкенда и шесть размеров батча (1, 4, 8, 16, 32, 64). Состав батча не контролируется по длине: каждый батч получает естественный микс длин, складывающийся при данном размере, — поэтому средняя длина растёт с размером батча (18 токенов при batch=1, 26–27 при batch=4–16, 73–75 при batch=32–64; максимумы от 18 до 382 токенов, приведены построчно в таблице 13).
- **Прогрев:** каждая конфигурация прогоняется с прогревочными итерациями до измерения; эти итерации отбрасываются, а приводимая статистика вычисляется по последующим измеряемым итерациям.
- **Приводимая статистика:** среднее, медиана, P95 и P99 задержки на конфигурацию, а также пропускная способность в текстах в секунду; в отчёте приведены P95 (таблица 13) и пропускная способность (таблица 14), из которых при фиксированной цене \$0.3/GPU-час получены стоимостные цифры в таблице 15.
- **Кросс-проверка бэкендов:** логиты для одних и тех же входов численно сравниваются между бэкендами (PyTorch против ONNX Runtime, PyTorch против TensorRT); см. замечание о согласованности в конце раздела 7.3.

7.3 Задержка и пропускная способность по размерам батча

Таблица 13: Задержка P95 (мс) по бэкендам и размерам батча

Батч	Длина, ток. (сред./макс.)	PyTorch	ONNX Runtime	TensorRT
1	18 / 18	17.4	9.0	**5.9**
4	27 / 43	18.7	9.8	**7.2**
8	26 / 43	18.2	12.9	**7.9**
16	27 / 43	18.8	17.6	**12.5**
32	73 / 323	36.0	35.3	35.4
64	75 / 382	72.4	68.5	69.1

Таблица 14: Пропускная способность (текстов/с) по бэкендам и размерам батча

Батч	PyTorch	ONNX Runtime	TensorRT
1	61.6	121.0	**173.4**
4	220.6	427.1	**590.2**
8	466.1	679.7	**1045.1**
16	882.4	936.7	**1345.1**
32	912.4	929.8	936.5
64	916.5	**955.7**	954.6

Одиночный запрос (batch=1): TensorRT в 2.95× быстрее PyTorch (5.9 мс против 17.4 мс P95), ONNX Runtime — в 1.93× быстрее (9.0 мс).

Оптимум: batch=16 на TensorRT. Эти параметры дают лучшую задержку среди практичных размеров батча (12.5 мс), наивысшую пропускную способность (1345.1 текста/с) и наименьшую стоимость (см. ниже) — при этом компромисса между задержкой и пропускной способностью нет.

Бэкенды сходятся при batch ≥32: разрыв между PyTorch/ONNX/TensorRT падает до 2–6% (2.0% и 5.7% по задержке P95, 2.6% и 4.3% по пропускной способности при batch=32 и batch=64 соответственно), как только в батч естественным образом попадают более длинные тексты (средняя длина 73–75 токенов, максимум 323–382 против 18–43 при меньших батчах). Здесь одновременно происходят две вещи, и их стоит разделять. Во-первых, маршрутизация: TensorRT откатывается на PyTorch+FA2 выше 256 токенов (раздел 7.1), и это единственные строки, чьи максимумы (323 и 382 токена) пересекают этот порог, — поэтому часть того, что измеряется как «TensorRT» при батче 32–64, обслуживается путём PyTorch, что по построению сближает три бэкенда. Во-вторых, эффективность ядер: оставшаяся разница согласуется с тем, что профили оптимизации TensorRT настроены на низкую задержку при умеренных размерах батча, а не на насыщение GPU большими батчами. Измерение не разделяет эти два вклада; что оно даёт — это практический вывод: при batch ≥32 выбор бэкенда перестаёт иметь значение.

Согласованность бэкендов: измерялось численное совпадение *логитов* между бэкендами на одних и тех же входах (PyTorch против ONNX Runtime, PyTorch против TensorRT): максимальное абсолютное расхождение 0.002, среднее 0.000016. Это ограничивает численный дрейф, но не устанавливает идентичность меток. Пороги решения лежат в диапазоне 0.49–0.62 (таблица 9), поэтому оценка, отличающаяся от своего порога менее чем на 0.002, может оказаться по любую сторону в зависимости от бэкенда. Такие случаи ограничены

входами, по которым модель не определилась, и при такой величине расхождения они редки, но честная формулировка такова: смена бэкенда меняет скорость и оставляет оценки численно эквивалентными с точностью до 0.002. Развёртыванию, которому нужна побитовая воспроизводимость меток, следует зафиксировать один бэкенд.

7.4 Стоимость по размерам батча

Таблица 15: Стоимость инференса по бэкендам и размерам батча (иллюстративно: \$0.3/GPU-час, один GPU, полная утилизация)

Батч	PyTorch, \$/1 млн	ONNX, \$/1 млн	TensorRT, \$/1 млн
1	1.3528	0.6887	0.4806
4	0.3778	0.1951	0.1412
8	0.1788	0.1226	0.0797
16	0.0944	0.0890	**0.0620**
32	0.0913	0.0896	0.0890
64	0.0909	**0.0872**	0.0873

Таблица 15 переводит пропускную способность в стоимость на миллион запросов при фиксированной цене GPU. Минимум составляет **\$0.062 за 1 млн запросов** (TensorRT, batch=16). Для сравнения, фиксированный batch=8 на PyTorch (без оптимизации бэкенда и размера батча) стоит \$0.1788 за 1 млн запросов — переход на оптимальную конфигурацию снижает стоимость в **2.9**×. Относительно полного отсутствия батчинга (PyTorch, batch=1, \$1.3528 за 1 млн) снижение больше — в **21.8**×.

7.5 Эффективность батчинга

В таблице 16 приведён коэффициент параллелизма — пропускная способность(батч) / (батч × пропускная способность(batch=1)), где 1.0 соответствует идеальному линейному масштабированию:

Таблица 16: Коэффициент параллелизма по бэкендам и размерам батча

Бэкенд	batch=4	batch=8	batch=16	batch=32	batch=64
PyTorch	0.90x	0.95x	0.90x	0.46x	0.23x
ONNX Runtime	0.88x	0.70x	0.48x	0.24x	0.12x
TensorRT	0.85x	0.75x	0.48x	0.17x	0.09x

Коэффициент TensorRT падает быстрее остальных (0.17x при batch=32, 0.09x при batch=64 против 0.46x/0.23x у PyTorch). Это говорит о том, что TensorRT оставляет небольшое преимущество на одиночном запросе по мере роста батча, — а не о том, что он становится более медленным бэкендом. Коэффициент нормирован на пропускную способность каждого бэкенда при batch=1, а пропускная способность TensorRT в 2.8× выше, чем у PyTorch (173.4 против 61.6 текста/с), поэтому пропускная способность даёт заметно меньшее отношение. В абсолютных величинах на больших батчах три бэкенда расходятся на единицы процентов, причём TensorRT при batch=64 всё ещё немного впереди PyTorch (954.6 против 916.5 текста/с, таблица 14). **Практическая рекомендация:** использовать

TensorRT при батче $\leq$16, где его преимущество велико; при батче $\geq$32 выбор бэкенда почти не влияет на результат.

8. Заявление о данных и этические соображения

8.1 Заявление о данных

Происхождение. Корпус собран преимущественно из открыто лицензированных наборов данных на Hugging Face Hub и в других публичных репозиториях, отфильтрованных по совместимости лицензий с коммерческим использованием, плюс синтетические примеры, сгенерированные внутри компании для категорий, где открытых данных недостаточно (раздел 3). Языки — русский и английский, включая внутрисообщенческий code-switching. Корпус содержит только текст: никаких изображений, аудио или видео ни на одном этапе.

Разметка. В обучающем корпусе метки берутся из исходных наборов данных там, где они их предоставляют; синтетические данные проходят двухэтапную проверку — LLM-судья, затем рассмотрение экспертами-людьми — до попадания в обучение. Внешний оценочный набор раздела 6.2 размечался отдельно: каждый пример аннотировали три независимых аннотатора, итоговая метка — большинство из трёх, а межаннотаторское согласие по всему набору составляет κ Фляйсса $= 0.82$. Остаточный шум разметки при таком уровне согласия мал относительно различий между моделями, приведённых в разделе 6.2, но он не нулевой и одинаково применим ко всем оцениваемым на наборе моделям.

Персональные данные. Очистка удаляет самые частые носители идентифицирующей информации: URL заменяются плейсхолдером `[URL]`, номера карт и подобные идентификаторы вырезаются.

Публикация. Корпус не распространяется. Основная модель закрыта; вторая наша модель, `gliner_guard_omni`, опубликована открыто [17].

8.2 Чувствительный обучающий материал

Два аспекта обучающих данных следует проговорить прямо, а не оставлять подразумеваемыми.

Эксплуатация детей. Таксономия включает `Child_exploitation`, а классификатор не может выучить категорию, которой он никогда не видел. Соответствующий обучающий материал является исключительно текстовым — контент, описывающий трудоэксплуатацию несовершеннолетних, склоняющий к ней или способствующий ей, — взят из открытых наборов данных по безопасности и модерации. Изображения не задействованы ни на одном этапе, и не производится материалов, которые сами по себе могли бы функционировать как инструктивный или эксплуатирующий контент. Корпус не публикуется и не распространяется.

Синтетические данные от abliterated-моделей. Для категорий с критическим дефицитом открытых данных мы генерировали примеры с помощью abliterated-моделей с открытыми весами — моделей, у которых удалено поведение отказа, что как раз и позволяет им производить небезопасный текст, нужный детектору в качестве положительных примеров. Это несёт очевидный риск и соответственно ограничивается: генерация выполняется офлайн в изолированной среде, выход используется исключительно как обучающие данные классификатора, каждый сгенерированный пример проходит описанную выше двухэтапную верификацию, и ни сгенерированный корпус, ни промпты генерации не публикуются.

Abliterated-модели используются как инструменты генерации данных и не входят ни в обученную систему, ни в какой-либо serving-путь.

8.3 Назначение и недопустимое использование

Модель предназначена как слой фильтрации входов в генеративные системы и выходов из них: она выдаёт покатегорийные вероятности, которые внедряющая система направляет в логирование, рассмотрение человеком, эскалацию или отказ (раздел 1.3). Она спроектирована для этой роли на русском и английском языках, на длинах и в регистре чат- и ассистент-трафика.

Она не подходит и не должна использоваться для: автоматических решений в отношении людей без рассмотрения человеком; слежки, профилирования или правоохранительных заключений; доказывания умысла или юридической виновности; контента на языках, отличных от русского и английского, где она не оценивалась; а также в качестве единственного механизма безопасности без мониторинга — с учётом покатегорийного качества, приведённого в разделе 6.2.

8.4 Риски и справедливость

Конструкция с приоритетом полноты (раздел 5.1) означает, что ожидаемый режим отказа — ложные срабатывания, и они распределены неравномерно. Тематический детектор срабатывает на тему независимо от позиции по данным вопросам, поэтому издержка ложного срабатывания по `LGBT`, `Religion` или `Policy` непропорционально ложится на людей, говорящих на эти темы, включая членов затрагиваемых групп; под срабатывание попадают в том числе поддерживающие, личные и информационные высказывания. Эти три категории ведут себя по-разному в разделе 6.2: `LGBT` — одна из сильнейших категорий модели (F1 0.89), то есть срабатывает надёжно; `Religion` и `Policy` — одни из слабейших (0.62 и 0.58), то есть срабатывают шумно. Ни один из профилей не успокаивает, и они требуют разных ответов. Развёртываниям, которым эти категории не нужны, следует их отключить; тем, которым они нужны, следует направлять положительные срабатывания на рассмотрение человеком и иметь возможность сообщить затронутому пользователю, что решение было принято и на каком основании.

Тот же довод с обратным знаком применим к категориям вреда высокой тяжести: `Hate_discrimination` с F1 0.56 пропускает существенную долю того, что должна ловить, поэтому развёртыванию, полагающемуся на неё для защиты пользователей, нужен покатегорийный мониторинг полноты (раздел 6), а не уверенность в агрегированной цифре.

9. Ограничения

- **Новые паттерны обфускации.** Модель устойчива к обфускациям, включённым в обучение (замена символов, транслитерация, переключение раскладки; раздел 3.4). Техники обхода вне этих семейств требуют дополнительного дообучения.

- **Окнам длинных входов не хватает документного контекста.** Инференс не отбрасывает содержимое запросов длиннее 1024 токенов: он разбивает их на перекрывающиеся окна по 1024 токена (перекрытие 128 токенов) и оценивает каждое окно независимо, беря максимум вероятности по каждой категории среди окон как вердикт по запросу (раздел 4) — небезопасный фрагмент в любом отдельном окне не размывается окружающими безопасными окнами. Тем не менее каждое окно оценивается без контекста остального документа, а обучение никогда не показывало модели окно из середины документа такого рода, поскольку длинные обучающие примеры были исключены целиком, а не разбиты на окна; это риск сдвига распределения, специфичный для длинных входов, и он не смягчается маршрутизацией по длине из раздела 7, которая влияет только на скорость.

- **Тематические детекторы шире любого правила, которому они служат.** `Religion`, `LGBT` и `Policy` срабатывают на предметную область, деяние или позицию, поэтому положительная оценка охватывает существенно более широкий набор текстов, чем любая политика или норма, к которой её может применять развёртывание (раздел 1.3). Решение о том, какие из этих текстов действительно значимы, модель не принимает и принять не может; оно зависит от того, куда развёртывание направляет положительные срабатывания. Там, где размыта сама тематическая граница — мимолётные упоминания, цитирование, метаобсуждение, — решения дополнительно неустойчивы на пограничных случаях.

- **Редкие сочетания категорий.** Совстречаемость четырёх и более категорий была редка в обучающих данных, и поведение модели в таких случаях менее предсказуемо.

- **Правка порогов — единственный быстрый рычаг.** Между циклами переобучения единственная доступная в продакшене коррекция — изменение порога (раздел 5.3). Оно двигает порог категории вдоль её существующей кривой точность/полнота; оно не может починить категорию, которую модель плохо классифицирует при любом пороге, — а это как раз ситуация слабейших категорий из раздела 6.2.

10. Извлечённые уроки

Несколько результатов этого проекта выходят за пределы разработки конкретной модели. Таблица 17 — справочный указатель к тому же набору, сопоставляющий каждое изменение с измеренным эффектом.

Большую часть итогового качества определили покрытие датасета и качество разметки, а не архитектурные изменения: наибольший прирост в рамках одной модели дал не новый компонент, а добавление верифицированных данных для редких категорий (раздел 3). Единый глобальный порог 0.5 занижает достижимое качество на редких категориях; покатегорийный перебор по сетке на eval-части поднял F1 на 10–30 пунктов на отдельных редких категориях относительно этого глобального порога (раздел 5). Усреднённые по EMA веса ($\alpha = 0.999$) стабильно превосходили сырые веса последней эпохи — на 0.5–2 пункта F1-macro на eval — и сохраняются в чекпоинты и загружаются преимущественно на инференсе.

Что касается качества данных, синтетические примеры, сгенерированные без проверки человеком, ухудшали метрики на реальных примерах: низкокачественная синтетика порождала систематические ошибки, которые постфактум было трудно диагностировать; именно поэтому синтетические данные теперь требуют и прохода AI-судьи, и верификации экспертом-человеком перед включением. Стратегия разбиения имела значение по той же причине — случайное разбиение позволяло `Child_exploitation` и `Nazism` периодически полностью исчезать из eval-части, что делало мониторинг обучения для этих классов ненадёжным и мотивировало переход на `MultilabelStratifiedKFold` (раздел 3).

Что касается оптимизации, единая скорость обучения для всей модели дестабилизировала предобученный backbone при высоких значениях и одновременно оставляла обучаемые с нуля экспертные головы сходящимися слишком медленно при низких; дифференцированные скорости обучения (backbone 3e-5, эксперты 5e-4) устранили оба режима отказа одновременно (раздел 5). Что касается serving, узкий профиль `min_seq_len=16` в конфигурации оптимизации TensorRT направлял каждый запрос короче 16 токенов на PyTorch-fallback, полностью маскируя ускорение от TensorRT на коротких текстах; расширение профиля до `min=1` проявило ожидаемое ускорение $\sim$2.95$\times$ при batch=1 (раздел 7).

Таблица 17: Ключевые результаты проекта: что изменилось и какой эффект измерен

#	**Область**	**Из → в**	**Измеренный эффект**
1	Данные	Архитектурные итерации → покрытие датасета и качество разметки	Крупнейший единичный источник прироста качества (не выделен как абляция)
2	Пороги	Глобальный 0.5 → покатегорийный перебор по сетке	+10–30 пунктов F1 на отдельных редких категориях, на eval
3	Веса	Последняя эпоха → EMA ($\alpha = 0.999$)	+0.5–2 пункта F1-macro на eval
4	Синтетические данные	Неверифицированная генерация → LLM-судья + проверка экспертом	Устранён источник уверенных систематических ошибок на реальных примерах
5	Разбиение	Случайное → `MultilabelStratifiedKFold`	`Child_exploitation` и `Nazism` перестают исчезать из eval
6	Оптимизация	Единый LR → backbone 3e-5, эксперты 5e-4	Одновременно устранены нестабильность backbone и медленная сходимость экспертов
7	Serving	TensorRT `min_seq_len=16` → `min=1`	Проявлено ускорение $\sim$2.95$\times$ при batch=1, замаскированное профилем

11. Выводы

Мы описали ориентированный на промышленную эксплуатацию multi-label guardrail-классификатор для двуязычного (русский/английский) детектирования небезопасного контента по 15 категориям. Архитектура объединяет многоязычный энкодер с параллельными экспертами по категориям и модулем взаимодействия категорий; обучение использует Asymmetric Loss, дифференцированные скорости обучения, усреднение EMA и покатегорийный подбор порогов, чтобы явно закодировать асимметричную цену пропусков; калибровка вероятностей в конвейер не входит. Независимый бенчмарк против 9 открытых guardrail-моделей сторонних разработчиков и второй нашей собственной модели (всего 11 участников) на внешнем сбалансированном наборе из 7,480 примеров показывает, что одна лишь бинарная точность safe/unsafe — плохой заменитель multi-label guardrail-качества и что согласие моделей разных семейств низко даже после приведения таксономий; выбор guardrail-модели требует валидации на целевом распределении. Агрегированные цифры также скрывают широкий покатегорийный разброс — F1 от 0.34 до 0.96 по меткам бенчмарка, включая 0.56 на `Hate_discrimination`, — поэтому развёртыванию, полагающемуся на какую-либо отдельную категорию, нужен покатегорийный мониторинг. Какие категории включает развёртывание и что мониторить/блокировать — это конфигурационные решения пользователя, и именно так они сформулированы (разделы 1.3 и 8). Исследование serving на трёх бэкендах показывает, что TensorRT при batch=16, без каких-либо изменений весов модели, является оптимумом по задержке (12.5 мс P95), пропускной способности (1345.1 текста/с) и стоимости ($0.062 за 1 млн запросов, снижение в 2.9$\times$ относительно фиксированного batch=8); при batch $\geq$32 три бэкенда сходятся и выбор бэкенда перестаёт быть содержательным решением.

Список литературы

[1] Кодекс Российской Федерации об административных правонарушениях, статья 6.21 («Пропаганда нетрадиционных сексуальных отношений и (или) предпочтений, смены пола, отказа от деторождения»). https://www.consultant.ru/document/cons_doc_LAW_34661/5b103e878283a04f8ea2130f38e19f10b516b626/

[2] Уголовный кодекс Российской Федерации, статья 148 («Нарушение права на свободу совести и вероисповеданий»). https://www.consultant.ru/document/cons_doc_LAW_10699/3f061fb01a04145dc7e07fe39a97509bd2da705f/

[3] M. Marone, O. Weller, W. Fleshman, E. Yang, D. Lawrie, B. Van Durme. mmBERT: A Modern Multilingual Encoder with Annealed Language Learning. arXiv:2509.06888, 2025. https://arxiv.org/abs/2509.06888. Beca: https://huggingface.co/jhu-clsp/mmBERT-base

[4] H. Inan et al. Llama Guard: LLM-based Input-Output Safeguard for Human-AI Conversations. arXiv:2312.06674, 2023. https://arxiv.org/abs/2312.06674

[5] W. Zeng et al. ShieldGemma: Generative AI Content Moderation Based on Gemma. arXiv:2407.21772, 2024. https://arxiv.org/abs/2407.21772

[6] S. Han et al. WildGuard: Open One-Stop Moderation Tools for Safety Risks, Jailbreaks, and Refusals of LLMs. arXiv:2406.18495, 2024. https://arxiv.org/abs/2406.18495

[7] T. Markov et al. A Holistic Approach to Undesired Content Detection in the Real World. AAAI 2023. arXiv:2208.03274, 2022. https://arxiv.org/abs/2208.03274

[8] A. Lees et al. A New Generation of Perspective API: Efficient Multilingual Character-level Transformers. KDD 2022. arXiv:2202.11176. https://arxiv.org/abs/2202.11176

[9] T. Ridnik, E. Ben-Baruch, N. Zamir, A. Noy, I. Friedman, M. Protter, L. Zelnik-Manor. Asymmetric Loss for Multi-Label Classification. ICCV 2021, pp. 82–91. Препринт: arXiv:2009.14119, 2020. https://arxiv.org/abs/2009.14119

[10] K. Sechidis, G. Tsoumakas, I. Vlahavas. On the Stratification of Multi-label Data. ECML PKDD 2011, LNCS vol. 6913, pp. 145–158, Springer. https://doi.org/10.1007/978-3-642-23808-6_10

[11] apanc/russian-sensitive-topics. https://huggingface.co/apanc/russian-sensitive-topics

[12] Qwen/Qwen3Guard-Gen-0.6B. https://huggingface.co/Qwen/Qwen3Guard-Gen-0.6B

[13] Qwen/Qwen3Guard-Gen-4B. https://huggingface.co/Qwen/Qwen3Guard-Gen-4B

[14] Qwen/Qwen3Guard-Gen-8B. https://huggingface.co/Qwen/Qwen3Guard-Gen-8B

[15] unitary/multilingual-toxic-xlm-roberta. https://huggingface.co/unitary/multilingual-toxic-xlm-roberta

[16] cointegrated/rubert-tiny-toxicity. https://huggingface.co/cointegrated/rubert-tiny-toxicity

[17] hivetrace/gliner-guard-omni. https://huggingface.co/hivetrace/gliner-guard-omni

[18] Alibaba-AAIG/YuFeng-XGuard-Reason-0.6B. https://huggingface.co/Alibaba-AAIG/YuFeng-XGuard-Reason-0.6B

[19] fastino/gliguard-LLMGuardrails-300M. https://huggingface.co/fastino/gliguard-LLMGuardrails-300M

[20] knowledgator/opir-multitask-multilang-v1.0. https://huggingface.co/knowledgator/opir-multitask-multilang-v1.0