Подходы к автоматизированной оценке клиентских ответов

В задачи входит анализ ~10 000 русскоязычных комментариев (2–3 предложения) по пяти критериям, где для каждого нужно определить **тональность** (положительно/отрицательно) и **категорию** (одну из 10 возможных аспектов, каждый из которых имеет положительный или отрицательный вариант, например «Процесс рассмотрения сделки. Скорость (положительно/отрицательно)»). Такая постановка близка к задаче **аспектно-ориентированного анализа тональности** (ABSA): необходимо выделять в тексте мнения о разных аспектах и оценивать их полярность [1]. Это требует классификации по нескольким меткам (каждому критерию) и обычно предполагает обучение моделей. Ниже рассмотрены основные подходы: от классических методов до современных языковых моделей.

Подходы на основе крупных языковых моделей (LLM)

Описание: современные «большие языковые модели» (GPT-3/4, Mistral, LLaMA, RuGPT, YaLM и др.) можно использовать двумя путями: *без обучения* (zero/few-shot prompt-инженерия) или *дополнительным дообучением* (fine-tuning). В первом случае в промпте модели описывают задачу (приводят инструкции и несколько примеров разметки), и модель генерирует ответы в требуемом формате. Во втором — LLM тонко настраивают на размеченном корпусе комментариев.

Преимущества: - Не требует/минимум ручной разметки: в zero-shot режиме подходит сразу, лишь нужны хорошо составленные подсказки [2] [3]. LLM обладают огромным «пониманием языка» и широкими знаниями (GPT обучены на больших корпусах), поэтому могут адекватно обрабатывать разнообразные тексты [2]. - Гибкость и мультизадачность: один и тот же GPT-4 или Mistral можно запросить одновременно определить тональность и категорию, предложить вывод в виде таблицы или JSON. LLM хорошо генерируют связный текст, умеют подстраиваться под стиль и контекст [4], что облегчает разработку инструкций. - Наличие русскоязычных моделей: RuGPT и YaLM (от Сбера и Яндекса) обучены на отечественных данных, что важно для русского текста.

Недостатки: - Стоимость и ресурсы: большие модели (особенно GPT-4 или YaLM-100B) требуют мощного оборудования или дорогого API. Запрос каждого комментария может быть медленным и дорогостоящим в продакшене. - Сложность вывода: модель может генерировать непредсказуемые ответы и требовать пост-обработки, особенно в zero-shot режиме. Точность в строго структурированных классификационных задачах обычно ниже, чем у специализированно обученных моделей [5]. - Интерпретируемость: LLM работают как «черный ящик» – сложно понять, почему выдан тот или иной ответ. Мало информации о том, на каких признаках модель основывалась. - Ограничения на длину и формат: в режиме prompt-инференса приходится заботиться о структуре вывода (JSON, столбцы и пр.) и учитывать ограничение на длину входного/выходного текста.

Процедура оценки: при zero-shot/few-shot подходе разрабатывается промпт со структурой (например, инструкция вида “Определи тональность и категорию (из списка) данного отзыва”), в

котором может быть приведено несколько примеров. LLM выдаёт прогнозы для каждого комментария. Если используется fine-tuning (например, дообучить LLaMA/Mistral), то готовится размеченный датасет (см. ниже) и производится обучение модели подобно любому трансформеру.

Необходимость дообучения: для базового прототипа не обязательно; многие задачи классификации можно решать подсказкой. Однако эксперименты показывают, что **тонкая настройка под задачу** (промпт-инженерия или fine-tuning на размеченных примерах) существенно повышает точность. Недавнее исследование показало, что для текстовой классификации с умеренным объёмом данных тонкое дообучение специализированной модели (например, BERT-стиля) постоянно превосходит нуль-шот GPT-подход [5]. Если есть размеченные примеры, то fine-tuning GPT или LLaMA/Mistral обычно даст лучший результат, но потребует значительных ресурсов.

Требования к данным: zero-shot — без размеченных данных, но важно продумать **писемное задание**. Для few-shot нужны лишь по 5–10 примеров. Для fine-tuning LLM (несколько сотен миллионов параметров) желательно иметь минимум тысячи размеченных комментариев (чем больше, тем лучше) для предотвращения переобучения и вывода шумовой разметки.

Время тренировки: для zero-shot GPT обучение не требуется (время — на разработку промпта). Fine-tuning больших моделей (GPT-4 нельзя тонко настроить, LLaMA/Mistral можно) на ~10к примерах может занимать **несколько часов-дней** на 4–8 GPU (зависит от мощности). Инференс (оценка) каждого комментария у GPT-4 занимает доли секунды реального времени, но при большом объёме (10 000 записей) суммарно может занять десятки минут и стоить денег.

Интерпретируемость и интеграция: вывод LLM сложнее отлаживать и объяснять (нет явных весов или признаков). Однако можно интегрировать LLM через API (OpenAI) или использовать локальные кастомные модели (Mistral, LLaMA, YaLM), предоставляя сервис через HTTP. Преимущество в гибкости: модель легко заменяется, но для промышленного применения нужна мощная инфраструктура. LLM подходят больше для прототипа/помощника разметчику, чем для конечного простого классификатора.

Трансформерные классификаторы (RuBERT, XLM-R и др.)

Описание: это нейросетевые модели на основе архитектуры BERT (двунаправленный трансформер). Для классификации их *дообучают* (fine-tune) на размеченных данных. Каждый комментарий кодируется трансформером (например, RuBERT) и затем через линейный слой классифицируется по тональности и/или категории. Можно строить **несколько отдельных моделей** (отдельно тональность, отдельно категория) или одну модель, которая выдаёт мультикласс («категория+тонус», например 20 вариантов).

Преимущества: - Высокая точность на текстовых классификациях. В исследованиях RuBERT превосходит классические методы (TF-IDF+SVM и др.) и мультилингвальные аналоги, давая на 3–9% выше F1 [6]. К примеру, на задаче тематики новостей RuBERT достигал F1≈93%, существенно опережая TF-IDF+SVM [6]. - Хорошо учитывают контекст и нюансы языка: BERT «видит» весь текст целиком и сопоставляет слова с контекстом [7] [8]. Это помогает распознавать сложные конструкции, сарказм, двусмысленность и т. п. (что чисто статистические модели часто упускают) [8]. - Быстрее обучения и компактнее, чем GPT: BERT-стиль модели имеют меньше параметров и обучаются параллельно всем словам, тогда как GPT обучается автрорегресивно [7]. В практике дообучение RuBERT на ~10–20k примерах может занять от десятков минут до нескольких часов на

одной современной GPU. - Много готовых библиотек и инструментов (Hugging Face, DeepPavlov) для русского языка. Например, `DeepPavlov` содержит готовые модели RuBERT для анализа тональности русского текста [8] [9].

Недостатки: - Требуют размеченного корпуса для обучения (минимум тысячи примеров, желательно сбалансированных по классам). Дообучение «с нуля» на 10k примерах может давать переобучение, поэтому обычно используют заранее обученную модель (RuBERT, XLM-R) и тонко настраивают. - Меньшая гибкость: одна модель специализируется на классификации, она не генерирует текст. Если захотеть менять формат вывода, нужно перекодировать или переназначить модель. - Интерпретируемость: хотя в BERT можно анализировать веса внимания, в целом модель даёт «черный ящик». Для объяснения нужны дополнительные методы (например, LIME/SHAP), но они дают лишь приближенную картину. - Большой объём: хотя RuBERT меньше GPT, её базовая версия (~110M параметров) всё равно весит сотни мегабайт и требует ускорителя для быстрой работы.

Процедура оценки: нужно подготовить размеченную выборку (например, 1000–5000 комментариев) с корректными метками по 5 критериям. Существует выбор – делать одну многоклассовую модель (20 классов «категория+тон») или несколько бинарных/мультиклассовых моделей. После разметки модели обучаются классическим fine-tuning'ом: подается каждая запись и правильная метка, оптимизируется кросс-энтропия. Затем модель используется для предсказания на новых отзывах. Оценка производительности — обычно метрики Precision/Recall/F1 для каждого критерия.

Необходимость дообучения: требуется обязательно, поскольку без fine-tuning модель не знает категорий заказчика. Предварительно предобученные модели на общих корпусах обеспечат хорошую стартовую точку, но слепое их применение (zero-shot) для этого узкоспециального задания будет неточным. Уже 1000–2000 помеченных примеров могут дать приемлемое качество, далее точность растёт с объёмом разметки. Важно также учитывать баланс классов (например, позитивных/негативных отзывов и распределение по категориям).

Требуемый объём данных: для каждой цели (тональность и категории) минимум несколько сотен примеров на класс, лучше тысячи+; то есть для 10+ категорий и двух тональностей понадобятся тысячи размеченных отзывов. Если всё 5 критериев часто упоминаются вместе, то можно совместно строить разметку. Опыт показывает, что BERT-семейство может начать давать разумные результаты уже на ~1000–2000 меток, а полноценно — на 5–10 тысячах.

Время обучения: на GPU типа NVIDIA A100/B40 (или даже одном средней мощности, например RTX) fine-tuning на 10k примеров занимал бы порядка **нескольких часов**. Конкретно: 3–4 эпохи по 10k записям ≈2–5 часов. Инференс одной записи в RuBERT занимает миллисекунды (на GPU ~1–3 ms, на CPU десятки ms), то есть оценка всей базы (10k) укладывается в секунды-минуты.

Интерпретируемость и интеграция: модель можно упаковать в сервис (FastAPI, Flask и т.п.), легко встроить в существующий бекенд. Для объяснения решения можно анализировать наиболее влиятельные токены (например, через attention или методы типа SHAP). Это сложнее, чем для линейных моделей, но для BERT-систем уже существуют готовые методики. Обычно модели разворачивают локально на сервере со словарём tokenizer для русского (сохранённые через HuggingFace или DeepPavlov).

Традиционные ML-методы (SVM, XGBoost, MLP и др.)

Описание: классические подходы основываются на вручную построенных признаках: например, TF-IDF-векторы слов/грамм, эмбеддинги (word2vec, FastText) усреднённые по словам, лингвистические фичи (количество слов, восклицания и т.д.). На таких признаках обучают линейные или ансамблевые классификаторы: SVM, логистическая регрессия, XGBoost, простые нейронные сети (MLP).

Преимущества: - Простота и быстрота: обучение на 10k примерах даже с тысячами признаков занимает минуты, а предсказание – миллисекунды. Не нужны GPU. - Интерпретируемость: модели типа деревьев или линейных позволяют легко увидеть важность признаков (каких слов/выражений модель «больше» боится/любит), что можно объяснить аналитикам. - Минимальные требования: нужны только сам корпус и конструктор признаков (например, sklearn, gensim). Можно начать с малой выборки и быстро получить базовый классификатор. - Сбалансированность ресурсов: подходят, если скомпрометированно нет данных или обучать нейросеть нецелесообразно.

Недостатки: - Более низкая точность в сложных задачах NLP. Для задач тонального анализа и аспектной классификации они обычно уступают трансформерам, особенно когда тексты короткие и содержат контекстные нюансы [6]. - Сложность настроек: качественные текстовые признаки требуют экспериментов (n-граммы, морфология, стемминг). Плохо обрабатывают синонимы и контекст (все слова считаются «мешком»). - Ограниченность: лексические маркеры могут не захватывать жаргон, эмодзи, или новые формы речи, тогда как нейросети учатся этим автоматически.

Процедура оценки: строится пайплайн: например, тексты нормализуются (токенизация, лемматизация), вычисляются TF-IDF, на них обучается XGBoost (или SVM). Для multi-аскетного случая можно обучать несколько моделей: одну классифицировать тональность, другую выбирать категорию, или одну модель предсказывать сочетания (ко всему тексту). Метрики оценки те же. Возможно, стоить использовать множественные фичи (словесные + эмбеддинги). Для русского доступны FastText-векторы или тематические словари.

Необходимость дообучения: зависит от метода. Энд-то-энд «без обучения» правило (например, наивный подход по позитивным/негативным словам) нет, но обычно нужен обученный на примерах алгоритм. Однако не надо «дообучать» уже натренированную модель – обучают сразу под задачу. То есть нужно собрать размеченные данные (как минимум для теста/валлидации).

Требуемый объём данных: можно начать с более скромной выборки (~500–1000 примеров) и получить простой классификатор. Но качественной работы по всем 10 категориям и тональности требуют, конечно, больше данных (чтобы покрыть все классы). Однако классические модели хуже переносят малую выборку – перекосы в данных сразу сказываются на результатах.

Время обучения: очень быстро (от секунд до минут для SVM или XGBoost на 10k примерах и десятках тысяч признаков). Предсказание почти мгновенное на CPU. Это удобно, если нужен простой интерактивный прототип.

Интерпретируемость и интеграция: интерпретируемость здесь высшая: можно получить список ключевых слов/фраз, связанных с каждым классом. Интеграция простая: любой готовый ML-классификатор можно деплоить как REST-сервис. Такие модели можно легко «объяснить» руководителю или сделать проверочный пайплайн.

Сравнение подходов (таблица)

Подход	Преимущества	Недостатки	Требования к данным	Обучение/ инференс	Интерпретируемость и интеграция
LLM (GPT, Mistral, YaLM)	– Не требует изначальной разметки (zero-shot) – Гибкая генерация ответов, объединяет несколько задач [3] – Многоязычный контекст	– Дорогие и ресурсоёмкие при развёртывании – Могут ошибаться, сложны в отладке – Низкая интерпретируемость	Нет или минимум (для few-shot)	Нет обучения (при prompt) / дни GPU (fine-tune)	Сложно объяснить решения; работать через API или сервис
Fine-tuned transformers (RuBERT, XLM-R и др.)	– Высокая точность классификации (лучше TF-IDF/SVM) [6] – Учит контекст, улавливает нюансы [8] – Быстрое GPU-дообучение и быстрая инференс (мс) [7]	– Нужны размеченные данные для обучения – Меньше гибкости вывода (фиксированные метки) – Сложнее поддерживать, чем простые модели	Тысячи примеров на все категории	Часы на GPU (fine-tuning); инференс ~мс/запись	Потруднее объяснить (сложная сеть), но интегрируется в пайплайн
Классические ML (SVM, XGBoost, MLP)	– Просты в реализации, требуют малых ресурсов – Быстрое обучение и предсказание (секунды/минуты) – Хорошая интерпретируемость (особенно дерево/ линейные)	– Низкая точность на сложных контекстах [6] – Затраты на ручные фичи, плохо схватывают контекст – Жёстко фиксированный словарь	Необходим хоть какой-нибудь размеченный корпус (даже 500 примеров)	Мгновенное обучение/ предсказание на CPU	Легко анализировать (важность слов); легко деплоить

Примечание: **BERT/RuBERT** в таблице относятся к дообучению на наших данных; **LLM (GPT)** – это подход с подсказками без обучения. Цитаты: RuBERT превосходит TF-IDF/SVM на русском [6]; GPT хорошо сгенерирует текст и даже решит задачу анализа тональности [3], но специализированные модели лучше подходят для классификации [5] [10].

Рекомендации по архитектуре и переходу к автоматизации

- **Текущий этап (полуавтоматическая разметка):** рекомендуется использовать гибридный подход с участием человека. Например, можно применять LLM или предварительно обученные модели для **генерации подсказок разметчику**. Варианты:

- **Подсказка через LLM:** сформулировать инструкцию и дать LLM (GPT-4, Mistral, YaLM) с примерами задач помочь в разметке. Модель может предложить категорию и тональность каждого критерия; специалист проверяет и при необходимости правит. Это ускорит разметку и даст обучающие примеры для будущего.
- **Базовое ML-классификатор:** начать с простого (например, XGBoost на TF-IDF) для предварительного выделения тональности или категорий. Размечать дальше по полуавтоматической схеме: автомат предсказывает, человек правит. Постепенно собирать помеченные данные.
- **Интерактивная инструкция (LLM с few-shot):** специалист будет задавать LLM вопросы вида «Этот отзыв про какую стадию сделки? Оценка положительная или отрицательная?» и получать варианты ответа для проверки.
- **Средства разметки:** использовать инструменты (Label Studio, Yandex Toloka) и встроить туда рекомендательную модель (можно через API LLM или локальную библиотеку) – разметчик кликает «Принять/отклонить» подсказку.

Важно начать с создания первой размеченной выборки (например, 500–2000 отзывов), покрывающей все категории. На её основе обучить простой трансформер (RuBERT) и уже на его базе давать подсказки. Такой подход (active learning) сочетает интуитивную гибкость человека и силу модели.

- **Переход к полной автоматизации:** по мере роста размеченной базы нужно постепенно **доверять модельным решениям всё больше**. Стратегия:
- **Шаг 1:** Собрать качественную базу (через полуручную разметку) минимум в несколько тысяч примеров.
- **Шаг 2:** Fine-tune трансформер (RuBERT или аналог) на этих данных для задачи классификации (возможно, отдельная модель на тональность и на категории). Оценить её качество (кросс-валидация, удержать часть данных для теста).
- **Шаг 3:** Внедрить эту модель в пайплайн так, чтобы она автоматически разметила новые отзывы; оставить мониторинг/фидбек от экспертов (например, с помощью A/B-тестов: часть помечает человек, часть – модель). Постепенно, если качество будет стабильным, человек верифицирует лишь малую долю (например, редкие/двусмысленные случаи).
- **Шаг 4:** По необходимости регулярно дообучать модель: добавлять новые примеры (особенно ошибок), применять подходы **прямого доменного дообучения** (transfer learning с учётом специфики опроса).

В итоге рабочая система может стать полностью автоматической, где модель выдаёт для каждого отзыва по пять меток («тональность» и «категория»), а оператор лишь контролирует качество. При этом важно выстроить систему обратной связи: например, периодически человек просматривает часть прогнозов, чтобы вовремя подправить модель.

- **Возможная архитектура:** многозадачная нейросеть (например, одна BERT-модель с двумя заголовками: тональность и категория) или ансамбль из классификаторов (отдельные модели на разные критерии). Для начала проще обучать каждый критерий отдельно, а затем объединять в один сервис.

Модели и инструменты для русского языка

- **RuBERT и XLM-Roberta:** популярные российско-ориентированные BERT-модели. К ним доступны готовые веса (DeepPavlov, HuggingFace) и код для классификации [8] [9]. Например, `DeepPavlov/rubert-base-cased` или `Ai-forever/ruBERT-base`.

- **RuGPT (Sber):** autoregressive GPT-модели для русского (124M, 355M, 1.3B параметров) [11]. Подходят для генерации текста и few-shot задач, но их нельзя напрямую использовать без дообучения для классификации.
- **YaLM (Yandex):** языковые модели до 100B параметров. Бесплатно доступны версии RuGPT-3 (аналог 13B) и YaLM-100B. Мощные, но тяжёлые; применимы в ранних экспериментах с разметкой через API или локально (если есть ресурсы).
- **Mistral (западная):** открытые модели (7B, 8x7B) с поддержкой русского. Можно запустить локально, тонко настроить.
- **DeepPavlov:** библиотека с реализациями для русского NLP (классификация, тональность, NER). Есть готовые конвейеры классификации с RuBERT.
- **Hugging Face Transformers:** огромное количество мультилингвальных моделей (mBERT, XLM-R) и несколько русских. Удобно для fine-tuning своих моделей.
- **FastText и word2vec:** быстрые эмбеддинги от Facebook, обученные на русских текстах. Могут служить фичами для простых моделей или векторной кластеризации.
- **Sklearn и XGBoost:** для классических ML. Простые в использовании, есть русскоязычные туториалы (например, классификация отзывов на TF-IDF).
- **Label Studio, doccano:** инструменты для разметки текста, которые можно интегрировать с моделью-подсказчиком (через API).
- **Toloka (Яндекс):** краудсорс-платформа для разметки, где модель может давать подсказки экспертам.

В заключение, текущий этап целесообразно начать с полуавтоматических инструментов (LLM-помощник или базовый классификатор) для сбора разметки. Для полного автоматизированного конвейера лучше ориентироваться на fine-tuned трансформеры (RuBERT/XLM), поскольку они обеспечат наилучшее качество классификации в узкоспециализированных условиях [5] [6]. Со временем, при накоплении данных, систему можно развивать вплоть до автономной работы.

Источники: современная литература показывает эффективность BERT-стиля моделей в текстовой классификации [6] [8], а исследования 2024 г. подтверждают: при достаточных метках такие модели превосходят LLM в zero-shot режимах [5]. В нашей задаче правильная стратегия — сочетание возможностей LLM (для наработки разметки) и классического обучения нейросетей на собранных данных.

---

[1] Aspect-Based Sentiment Analysis as a Multi-Label Classification Task on the Domain of German Hotel Reviews
https://aclanthology.org/2023.konvens-main.21.pdf

[2] [3] [4] [7] [8] [10] Не GPT единым: преимущества BERT перед ChatGPT | RB.RU
https://rb.ru/opinion/bert-vs-gpt/

[5] arxiv.org
https://arxiv.org/pdf/2406.08660

[6] A9R9q8fgb_10zss8a_9n4.tmp
https://www.fruct.org/publications/volume-31/fruct31/files/Lag.pdf

[9] Classification — DeepPavlov 1.7.0 documentation
http://docs.deeppavlov.ai/en/master/features/models/classification.html

[11] ai-forever/ru-gpts: Russian GPT3 models. - GitHub
https://github.com/ai-forever/ru-gpts