Санкт-Петербургский государственный университет

Выродов Михаил Владимирович

Выпускная квалификационная работа

Разработка системы для универсальной классификации текстов

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2021 «Программная инженерия»*

Научный руководитель:
старший преподаватель каф. СП Ю. В. Литвинов

Рецензент:
разработчик АО “Точка” А.Д. Земеров

Консультант:
старший программист-разработчик ООО “ВК” Е. С. Спирин

Санкт-Петербург
2025

Saint Petersburg State University

Mikhail Vyrodov

Bachelor's Thesis

Development of a system for universal text classification

Education level: bachelor

Speciality *09.03.04 "Software Engineering"*

Programme *CB.5080.2017 "Software Engineering"*

Profile: *Software Engineering*

Scientific supervisor:
C.Sc., prof. Y. V. Litvinov

Reviewer:
Developer at JSC "Tochka" A. D. Zemerov

Consultant:
Senior Software Engineer at LLC "VK" E. S. Spirin

Saint Petersburg
2025

Оглавление

Введение

В современном мире социальные сети играют ключевую роль в обмене информацией между людьми: они стали неотъемлемой частью повседневной жизни миллионов людей, предоставляя платформу для общения, обмена идеями и обсуждения актуальных событий. Однако с ростом числа активных пользователей, а также объёмов информации и возникает проблема её качественной модерации. Множество постов могут содержать токсичный контент, спам или распространять дезинформацию, что может нанести вред как отдельным пользователям, так и репутации самой платформы. В таких условиях классификация контента становится одной из важнейших задач для поддержания порядка и безопасности в социальных сетях.

В последние годы задача классификации контента в значительной степени решается с помощью методов машинного обучения и, в частности, современных нейронных сетей. Самой популярной архитектурой моделей для классификации является BERT (Bidirectional Encoder Representations from Transformers) [3]. Однако, обучение классификатора для контента в социальной сети имеет ряд серьезных вызовов. Одним из главных является отсутствие специфичных размеченных данных для новых сценариев классификации. В условиях быстро меняющегося домена классификации часто невозможно оперативно собрать и разметить новый набор данных для дообучения модели. Это особенно критично, когда требуется срочно внедрить новый сценарий модерации, например, выявление ботов, которые общаются с пользователями, генерируя сообщения с помощью современных нейронных сетей.

Учитывая эти проблемы, становится очевидной необходимость использования универсального подхода к классификации контента. Такой подход позволяет системе адаптироваться к новым задачам и сценариям без необходимости предварительного дообучения на специализированных данных. В качестве универсального классификатора можно использовать большие языковые модели (БЯМ), такие как GPT-4o [7], которые могут выполнять задачи классификации текстов без необходи-

мости дополнительного обучения. Эти модели обладают высокой универсальностью и могут быть использованы для различных задач без необходимости адаптации к новым доменам. Однако применение БЯМ требует высоких вычислительных затрат и наличия дорогой инфраструктуры, что делает их использование экономически нецелесообразным для ряда компаний, имеющих ограничение по ресурсам.

Вместо этого, более эффективным решением является обучение классификатора, размер которого существенно меньше, чем размер популярных БЯМ, на большом и разнообразном наборе данных с множеством классов из различных доменов. В качестве успешного примера универсальной модели можно привести GLiNER (Generalist Named Entity Recognition Model) [6] — модель, разработанную для решения задачи распознавания именованных сущностей в тексте (NER, Named Entity Recognition). Данная модель имеет около 300 миллионов параметров и для её запуска достаточно иметь несколько гигабайт видеопамяти, в то время как популярные БЯМ имеют в среднем больше семи миллиардов параметров и для их запуска часто требуется несколько современных видеокарт. Задача NER заключается в обнаружении и классификации сущностей в тексте, таких как имена людей, названия организаций или каких-либо других сущностей. Например, в предложении «Павел Дуров посетил штаб-квартиру компании Telegram в ОАЭ» NER-модель может выделить и классифицировать такие сущности: «Павел Дуров» — человек, «Telegram» — компания, а «ОАЭ» — локация. GLiNER была обучена на большом и разнообразном искусственно сгенерированном наборе данных, созданном с использованием модели серии GPT. Такой подход позволил модели эффективно обобщать знания и работать с широким спектром доменов без необходимости дополнительного обучения под каждый новый тип сущности. Результаты, продемонстрированные GLiNER, показывают, что даже относительно компактная модель на базе BERT может эффективно работать в универсальном формате.

Этот пример подтверждает, что задача создания модели для решения задач классификации, аналогичной GLiNER, является реали-

зуемой, но требует изменений в архитектуре модели GLiNER, так как GLiNER умеет решать только задачу именованного распознавания сущностей. Используя аналогичный подход к обучению на большом и разнообразном наборе данных, можно создать эффективный и универсальный классификатор, способный адаптироваться к разным сценариям и задачам без необходимости дообучения. Такой подход позволит объединить высокую эффективность с экономичностью и доступностью для практического использования в компаниях, обрабатывающих большие объемы информации.

В условиях масштабов компании ВК, где разные отделы могут сталкиваться с разнообразными сценариями классификации контента, необходимо обеспечить простой и доступный способ взаимодействия с моделью. Разумным решением является создание внутреннего сервиса, с помощью которого будет использоваться разработанная модель классификации. Такой сервис позволит сотрудникам компании взаимодействовать с моделью через простой и интуитивно понятный API, отправляя запросы на классификацию контента. Это упростит интеграцию модели в существующие рабочие процессы и позволит оперативно решать задачи модерации, анализа тональности и темы контента, выявления спама и других сценариев классификации.

Данная работа направлена на разработку универсальной системы классификации текстов и внутреннего сервиса с удобным доступом к ней, который позволит эффективно классифицировать различные данные без необходимости разметки специфичных данных и дообучения модели под каждый новый сценарий классификации.

1. Постановка задачи

Целью работы является создание универсального решения для автоматической классификации текстов из различных доменов. Для её выполнения были поставлены следующие задачи.

1. Изучить существующие подходы к универсальной классификации текстов, провести их сравнительный анализ.

2. Спроектировать решение по универсальной классификации текстов, включая алгоритм работы классификатора и систему для его эффективного использования.

3. Подготовить обучающую выборку для классификатора, провести обучение классификатора на подготовленной выборке и оценить его производительность на тестовых данных.

4. Разработать программную систему, реализующую предложенное решение.

5. Выполнить экспериментальное исследование созданного решения на популярных задачах классификации, оценить его практическую эффективность.

2. Обзор

В данной главе представлен обзор и сравнительный анализ аналогов разрабатываемой системы, анализ модели GLiNER для универсального решения задач NER и обзор русскоязычных наборов данных для последующего определения архитектуры классификатора в разрабатываемой системе и набора данных для его обучения.

2.1. Нужные понятия из машинного обучения

Языковая модель — это распределение вероятностей по последовательностям слов. Любой последовательности слов длины m присваивается вероятность $P(w_1, w_2, ..., w_m)$. Обычно ЯМ представляет собой рекуррентную нейронную сеть.

Эта нейронная сеть учится предсказывать вероятности $P(w, context)$ $\forall w \in V$, где V — множество всех возможных слов, а $context$ — какой-то набор слов. $P(w, context)$ — вероятность того, что при добавлении к тексту $context$ следующего слова w, получившийся текст больше всего похож на тексты, на которых обучалась ЯМ. То есть модель, получив какой-то текстовый запрос $context$, считает вероятности $P(w, context)$ $\forall w \in V$, после этого выбирает наиболее вероятное следующее слово w_i для текста $context$ и теперь уже считает вероятность $P(w, context + w_i)$ $\forall w \in V$, где $context + w_i$ — строка $context$, сконкатенированная со словом w_i. Далее ЯМ выбирает наиболее вероятное следующее слово для строки $context + w_i$. Так ЯМ предугадывает следующие слова до того момента, пока самым вероятным следующим словом не станет специальное слово, обозначающее конец генерации текста. Накопившаяся последовательность слов является ответом ЯМ на изначальный текстовый запрос $context$.

Распределение вероятностей $P(w, context)$ нейронная сеть учит благодаря корректировке значений своих весов с помощью алгоритма градиентного спуска. Подробнее о самой популярной архитектуре языковой модели на основе нейронной сети, которая называется «трансформер», написано в статье [1].

Большие языковые модели — неофициальный термин для ЯМ на основе нейронный сетей, которые имеют более миллиарда весов. БЯМ обучаются очень долго и для хорошего обучения им требуется большой текстовый набор данных, который измеряется петабайтами.

2.2. Модель BERT и её применение для классификации текстов

На основе архитектуры трансформеров была разработана модель BERT (Bidirectional Encoder Representations from Transformers). Основная идея BERT заключается в том, что каждому токену во входном тексте сопоставляется численный вектор фиксированной размерности, называемый векторным представлением. Эти векторные представления кодируют семантическую информацию о словах, учитывая контекст, в котором они находятся. BERT использует только encoder часть архитектуры трансформеров. Модели, которые каждому токену во входном тексте сопоставляют численный вектор фиксированной размерности, называются моделями-кодировщиками.

Одной из ключевых особенностей BERT является способность учитывать контекст как с левой, так и с правой стороны каждого токена, что позволяет создавать более глубокие и точные векторные представления слов. Представления, полученные на выходе BERT, могут быть использованы для различных задач обработки текста, таких как машинный перевод, извлечение информации и, в частности, классификация текстов.

В контексте задач классификации текстов, векторные представления всех токенов обычно усредняются или используется специальный токен `[CLS]` (сокращение от Classification), чьё представление кодирует смысл всего текста. Полученный итоговый вектор передаётся в классификационный слой модели, который представляет собой полносвязный нейронный слой с многопеременной логистической функцией активации. Этот слой отвечает за преобразование векторного представления текста в определённую метку класса.

Процесс адаптации предобученной модели BERT к конкретной задаче классификации называется дообучением. В ходе дообучения может происходить обновление параметров как самого классификационного слоя, так и параметров всей модели BERT. На этом этапе модель обучается на размеченных данных из целевого домена, что позволяет улучшить её способность к классификации текстов в специфических сценариях.

2.3. Проблемы дообучения моделей в контексте классификации контента в социальной сети

В соцсетях контекст и тактики злоупотреблений меняются с высокой частотой, поэтому статический датасет для дообучения быстро устаревает. Базовая дообученная модель не успевает адаптироваться к новым видам токсичности, спама и, в общем, контента.

Кроме того, спектр задач необъятен: от тональности и тематики текста до неожиданных спец-кейсов, требующих немедленного реагирования. Поддерживать отдельную дообученную модель на каждый сценарий классификации непрактично и ресурсозатратно, поэтому нужен более универсальный классификатор, способный покрывать широкий набор задач без постоянной ручной разметки данных и частого дообучения моделей.

2.4. Обзор и сравнение аналогов

Универсальная классификация — это метод, при котором модель способна присваивать входным данным определённый класс без необходимости дообучения на каждом новом наборе данных. Рассмотрим три основных подхода к построению универсальных классификаторов:

2.4.1. Модели, дообученные на NLI (Natural Language Inference) наборах данных

Один из популярных подходов к созданию универсальных классификаторов — использование моделей-кодировщиков, дообученных на наборах данных с заданиями NLI (Natural Language Inference). Кодировщиками называются такие модели, которые преобразуют текст в численный вектор. NLI представляет собой задачу, в которой модели учатся определять, следует ли одно утверждение из другого. Примеры в NLI наборах данных состоят из двух текстов (гипотезы и следствия) и правильного ответа, который представляет собой один из трёх возможных ответов:

1. «есть следствие»;
2. «есть противоречие»;
3. «нет ни того, ни другого».

Процесс классификации через такие модели включает следующие шаги:

1. Для входного текста и каждого из n классов гипотезой является входной текст, а следствие формируется так: «`Данный текст принадлежит к классу: {class}`», где `class` — очередной класс из списка.
2. Запросы подаются в модель, и для каждого из них на основе полученного численного вектора входного запроса вычисляется вероятность того, что есть логическое следствие между гипотезой и следствием. То есть измеряется, насколько класс действительно соответствует входному тексту.
3. В качестве результата выбирается класс, для которого такая вероятность максимальна.

Данный подход требует многократных запусков модели для классификации входного текста (по одному запуску на каждый класс), что может быть очень ресурсозатратным при большом числе классов.

2.4.2. Использование кодировщиков без дообучения

Второй подход к универсальной классификации заключается в использовании кодировщиков (например, BERT или DeBERTa) без этапа дообучения на специфичных наборах данных. Такие модели уже могут кодировать тексты в векторы фиксированной размерности, которые можно использовать для вычисления их семантической схожести с векторными представлениями классов.

Процесс классификации через кодировщики выглядит следующим образом:

1. Для входного текста и для каждого из n классов вычисляются векторные представления (*embeddings*) с помощью кодировщика, причем n представлений классов достаточно вычислить один раз и переиспользовать при каждом новом запросе, так как список классов остается неизменным.

2. Вычисляется косинусное сходство (*cosine similarity*) между вектором входного текста и каждым из векторов классов.

3. В качестве результата выбирается класс, чьё векторное представление имеет наибольшее сходство с представлением входного текста.

Данный подход обладает большей эффективностью, чем предыдущий, так как позволяет проводить классификацию за один запрос к модели вместо n запросов. Однако его точность может уступать моделям, дообученным на NLI наборах данных.

2.4.3. Использование БЯМ

Третий подход к универсальной классификации предполагает использование качественных БЯМ (например, GPT-4o) для классификации. Эти модели обучались на огромном количестве разной информации и поэтому могут проводить классификацию в любом домене.

Процесс классификации с использованием БЯМ выглядит следующим образом:

1. В модель подаётся запрос, содержащий входной текст и список возможных классов.

2. Также в запрос добавляется инструкция, в которой модель просят выбрать один из предложенных классов, который наиболее точно соответствует входному тексту.

3. Модель генерирует ответ, указывая наиболее подходящий класс.

БЯМ обладают высоким качеством генерации ответов. Однако их использование требует больших вычислительных и временных затрат и доступа к мощным аппаратным ресурсам. Это делает их применение экономически нецелесообразным в среде с высокими требованиями к производительности.

2.4.4. Выбор моделей для сравнения

Для сравнения вышеописанных подходов было выбрано восемь популярных моделей для универсальной классификации:

- Из них следующие четыре кодировщика дообучены на NLI наборах данных:
  - facebook/bart-large-mnli [2];
  - MoritzLaurer/DeBERTa-v3-base-mnli-fever-anli [4];
  - MoritzLaurer/deberta-v3-large-zeroshot-v2.0 [4];
  - MoritzLaurer/mDeBERTa-v3-base-mnli-xnli [4].

  Их количество параметров находится между 184 и 435 миллионами. Данные модели были выбраны, так как они имеют большое количество ежемесячных скачиваний на HuggingFace[1] (модель facebook/bart-large-mnli имеет ≈3.5 миллиона ежемесячных скачиваний).

[1]Портал HuggingFace https://huggingface.co/ (дата обращения 27.05.2025)

- Для второго подхода была выбрана модель dunzhang/stella_en_400M_v5[2]. Это модель, имеющая максимальные оценки качества на таблице лидеров MTEB[3] из всех моделей, имеющих меньше миллиарда параметров. MTEB (Massive Text Embedding Benchmark) [10] — бенчмарк для оценки кодировщиков. Он состоит из 56 наборов данных, включающих в себя семь типов заданий, на которых и проверяются модели. Таким образом, эта модель имеет схожую с БЯМ эффективность, при этом занимая кратно меньше памяти.

- Также было выбрано четыре различных БЯМ, которые отличаются по количеству параметров и по качеству работы. Из них три модели имеют открытый код и одна, GPT-4o [7], с доступом по API. С открытым исходным кодом были выбраны следующие модели:

  – Qwen/Qwen2.5-1.5B-Instruct [13];

  – google/gemma-2-2b-it [8];

  – meta-llama/Llama-3.1-8B-Instruct [9].

  Их количество параметров разнится от полутора до восьми миллиардов.

2.4.5. Сравнение выбранных моделей

Для объективного сравнения моделей в задаче универсальной классификации было выбрано три набора данных для классификации текстов из MTEB (Massive Text Embedding Benchmark) [10]:

- mteb/tweet_sentiment_extraction[4];

[2]Репозиторий модели stella_en_400M_v5 https://huggingface.co/dunzhang/stella_en_400M_v5 (дата обращения 27.05.2025)

[3]Таблица лидеров https://huggingface.co/spaces/mteb/leaderboard (дата обращения 27.05.2025)

[4]Набор данных tweet_sentiment_extraction https://huggingface.co/datasets/mteb/tweet_sentiment_extraction (дата обращения 27.05.2025)

- mteb/amazon_reviews_multi[5] (было взято подмножество этого набора данных, в котором тексты на английском языке);
- mteb/mtop_domain[6].

Первый и второй набор данных были выбраны, потому что это единственные два набора данных для классификации на MTEB, на которых первая десятка моделей из таблицы лидеров MTEB имеет менее 90 % правильно решённых заданий. Это позволяет более чётко различать производительность моделей, так как высокие значения точности (более 90–95 %) могут сглаживать разницу между моделями и затруднять объективное сравнение их эффективности. Также, тексты в них имеют аспект социального взаимодействия (первый набор данных содержит тексты из отзывов на различные товары, а второй — тексты постов в популярной социальной сети).

Последний набор данных был выбран, потому что тексты и классы в нём охватывает сразу несколько доменов. Таким образом будет проверятся универсальность модели для классификации.

На моделях каждого вида валидация проводилась по-разному:

- На моделях, дообученных на NLI наборах данных, валидация проводилась с помощью zero-shot classification pipeline[7] в библиотеке HuggingFace transformers[8]. Данный конвейер составляет запросы к модели именно так, как это было описано выше;
- Модель dunzhang/stella_en_400M_v5 была использована так же, как описано использование кодировщиков без дообучения выше;
- Процесс валидации БЯМ отличался в зависимости от модели. Для моделей со сравнительно небольшим количество параметров

[5]Набор данных amazon_reviews_multi https://huggingface.co/datasets/mteb/amazon_reviews_multi (дата обращения 27.05.2025)

[6]Набор данных mteb/mtop_domain https://huggingface.co/datasets/mteb/mtop_domain (дата обращения 27.05.2025)

[7]Документация HuggingFace https://huggingface.co/tasks/zero-shot-classification (дата обращения 27.05.2025)

[8]Библиотека transformers https://github.com/huggingface/transformers (дата обращения 27.05.2025)

(Qwen/Qwen2.5-1.5B-Instruct [13], google/gemma-2-2b-it [8]) вместо классического механизма генерации текста ответ генерировался по конкретной структуре с помощью фреймворка outlines[9] для структурной генерации. Для каждого валидационного набора данных БЯМ могла сгенерировать только те слова, которые являлись классами в этом наборе данных. Такое решение было принято, потому что БЯМ с малым количество параметров часто не следуют структуре ответа, которая была задана в запросе к ним, и могут генерировать лишние символы;

- Модели meta-llama/Llama-3.1-8B-Instruct [9] и GPT-4o [7] валидировались без использования структурной генерации, так как они достаточно качественные, чтобы генерировать только нужные классы в качестве ответа, без лишних символов. Для генерации классов через GPT-4o был использован OpenAI Batch API.

Результаты валидации выбранных моделей на этих трех наборах данных представлены в Таблице 1. Жирным текстом выделены первый и второй максимум в каждом ряду, так как первый максимум всегда достигается моделью GPT-4o [7]. Это неудивительно, так как эта модель, которая требует наибольших ресурсных затрат среди всех рассмотренных.

По результатам становится понятно, что задача универсальной классификации решаема и с помощью кодировщиков. Разница между F1 метрикой самой качественной БЯМ (GPT-4o [7]) и кодировщиками в среднем равна 11.4 процентов. При этом разница между БЯМ meta-llama/Llama-3.1-8B-Instruct и кодировщиками в среднем равна 10 процентов, учитывая что эти кодировщики имеют в 20 раз меньше параметров.

[9]Фреймворк outlines https://github.com/dottxt-ai/outlines (дата обращения 27.05.2025)

Model	mteb domain	tweet sentiment	amazon reviews	average
MoritzLaure/DeBERTA-v3-base-mnli-fever-anli	0.57	0.64	0.35	0.52
facebook/bart-large-mnli	0.73	0.47	0.35	0.52
MoritzLaure/deberta-v3-large-zeroshot-v2.0	0.85	**0.66**	0.33	0.62
MoritzLaure/mDeBERTa-v3-base-mnli-xnli	0.77	0.50	0.35	0.54
dunzhang/stella_en_400M_v5	**0.91**	0.51	0.38	0.60
meta-llama/Llama-3.1-8B-Instruct	0.86	0.62	**0.46**	**0.65**
Qwen/Qwen2.5-1.5B-Instruct	0.82	0.56	0.26	0.55
google/gemma-2b-it	0.63	0.28	0.26	0.39
GPT-4o	**0.94**	**0.69**	**0.51**	**0.71**

Таблица 1: Результаты валидации выбранных моделей в виде F1 метрик

2.5. Анализ модели GLiNER

Для определения архитектуры универсального классификатора был проведён анализ существующих универсальных моделей в смежных задачах. Из недавно выпущенных таких моделей особенно выделяется GLiNER (Generalist Named Entity Recognition Model) [6]. Она предназначена для задачи выделения сущностей и их типов из текста. Данная модель, имеющая всего 300 миллионов параметров, обошла модель UNINER-13B, которая решает ту же задачу, но имеет 13 миллиардов параметров, на большинстве валидационных наборов данных.

2.5.1. Архитектура модели GLiNER

Модель GLiNER представляет собой компактную модель для распознавания именованных сущностей (Named Entity Recognition, NER), построенную на основе двунаправленной трансформерной архитектуры (Bidirectional Transformer Language Models, BiLM). Популярными примерами моделей с такой архитектурой являются BERT [3] и DeBERTa [5]. Архитектура GLiNER представлена на Рисунке 1.

На вход подаётся объединённая последовательность: сначала типы сущностей, разделённые токеном `[ENT]` (например, `[ENT] Person [ENT] Organization [ENT] Location [SEP]`), затем сам текст. BiLM-кодировщик обрабатывает всю цепочку и выдаёт контекстуальные векторные представления каждого токена.

Для каждого типа сущности его `[ENT]`-токен проецируется через небольшой полносвязный слой в вектор той же размерности. Диапазо-

ны текста кодируются парой векторов начального токена и конечного токена. Сопоставление происходит через сигмоидную функцию активации, применённую к скалярному произведению между вектором сущности и вектором каждого диапазона в тексте. По итогам строится матрица размером (количество сущностей) × (количество диапазонов), из которой для каждого типа сущности выбирается диапазон с максимальной вероятностью.

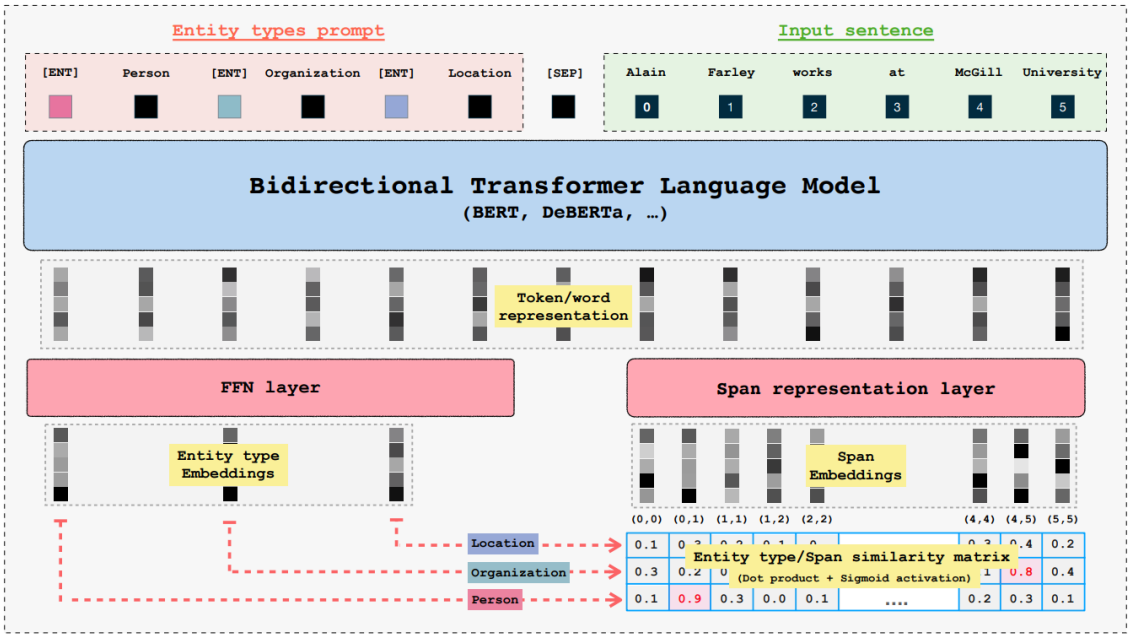

Рис. 1: Архитектура модели GLiNER

2.5.2. Обучающая выборка GLiNER

Модель обучалась на наборе данных Pile-NER[10], содержащем 50 000 текстов, собранных из разнообразного англоязычного набора данных The Pile [11]. Каждый текст в Pile-NER имеет список найденных в нём сущностей и их типов, полученный с помощью ChatGPT. Набор включает 240 000 выделенных в текстах сущностей и 13 000 уникальных типов сущностей.

Аналогично GLiNER, для обучения универсального классификатора также требуется разнообразный набор текстов.

[10]Набор данных Pile-NER https://huggingface.co/datasets/Universal-NER/Pile-NER-type (дата обращения 27.05.2025)

3. Решение по универсальной классификации текстов

Разработанное решение по универсальной классификации состоит из двух компонентов:

- Обученный на задачу универсальной классификации кодировщик. Для его обучения был разработан алгоритм универсальной классификации на основе архитектуры модели GLiNER [6], обучающая выборка, и был написан код обучения.

- Распределенная система для унифицированного использования обученного классификатора или произвольной БЯМ. Система состоит из внешнего API слоя, балансировщика нагрузки и из двух сервисов по эффективному использованию БЯМ или разработанного кодировщика.

В основе алгоритма работы универсального классификатора лежит адаптация модели GLiNER [6], изначально предназначенной для задачи распознавания именованных сущностей, к задаче универсальной классификации. Модель преобразует как анализируемый текст, так и названия возможных классов в численные векторы общего семантического пространства; затем по косинусной схожести между вектором текста и векторами классов выбирается класс с наибольшим соответствием. Такой приём позволяет обрабатывать новые сценарии классификации с помощью изменения названий классов во входной строке модели, при этом без повторного обучения нейросети.

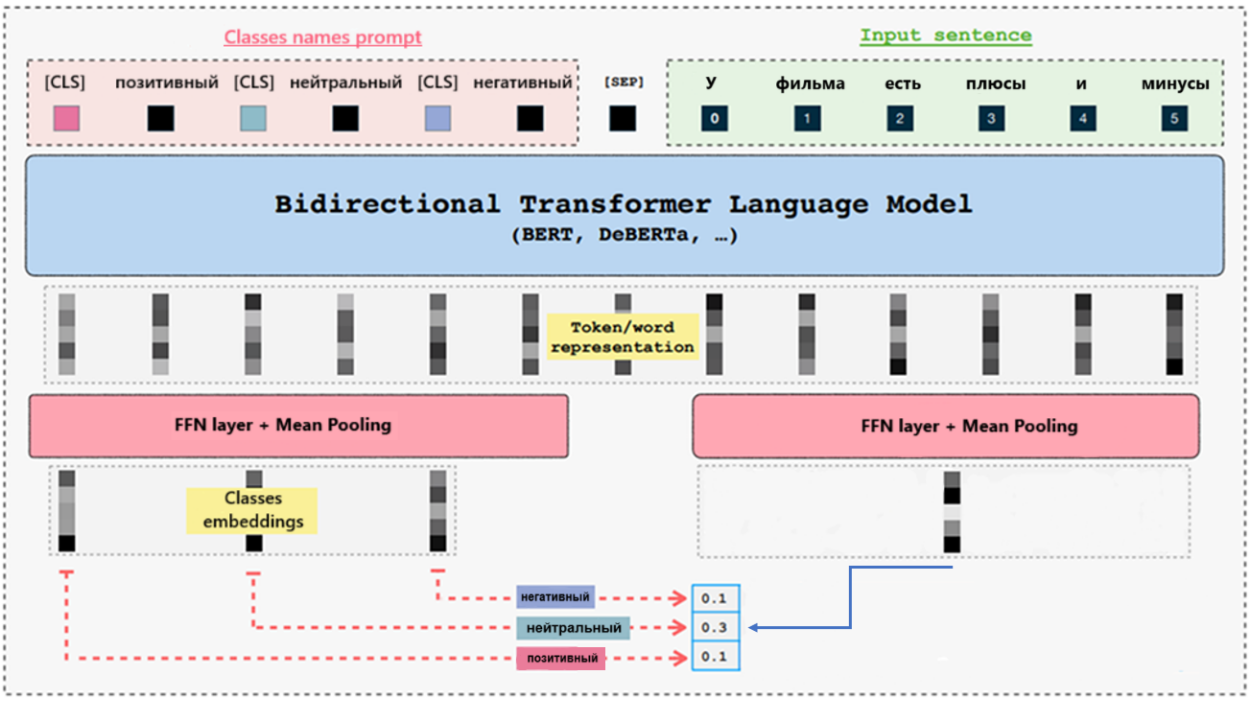

Рис. 2: Алгоритм работы классификатора

Главное отличие данного алгоритма от оригинальной модели GLiNER в том, что в ней вместо вектора схожестей строится матрица схожестей для каждого отрезка текста, так как в задаче NER необходимо рассматривать каждый отрезок текста и определять, обозначает ли он какую-либо сущность из списка. Однако, для задачи классификации достаточно иметь один текстовый вектор, что упрощает модель.

Для практического применения универсального классификатора в промышленном масштабе недостаточно обучить качественную модель — необходимо интегрировать её в инфраструктуру компании. Для этого была разработана распределенная система для универсальной классификации текстов. Ниже приводится описание предназначения каждого блока, изображенного на диаграмме развертывания системы, а также их взаимосвязи в едином конвейере классификации.

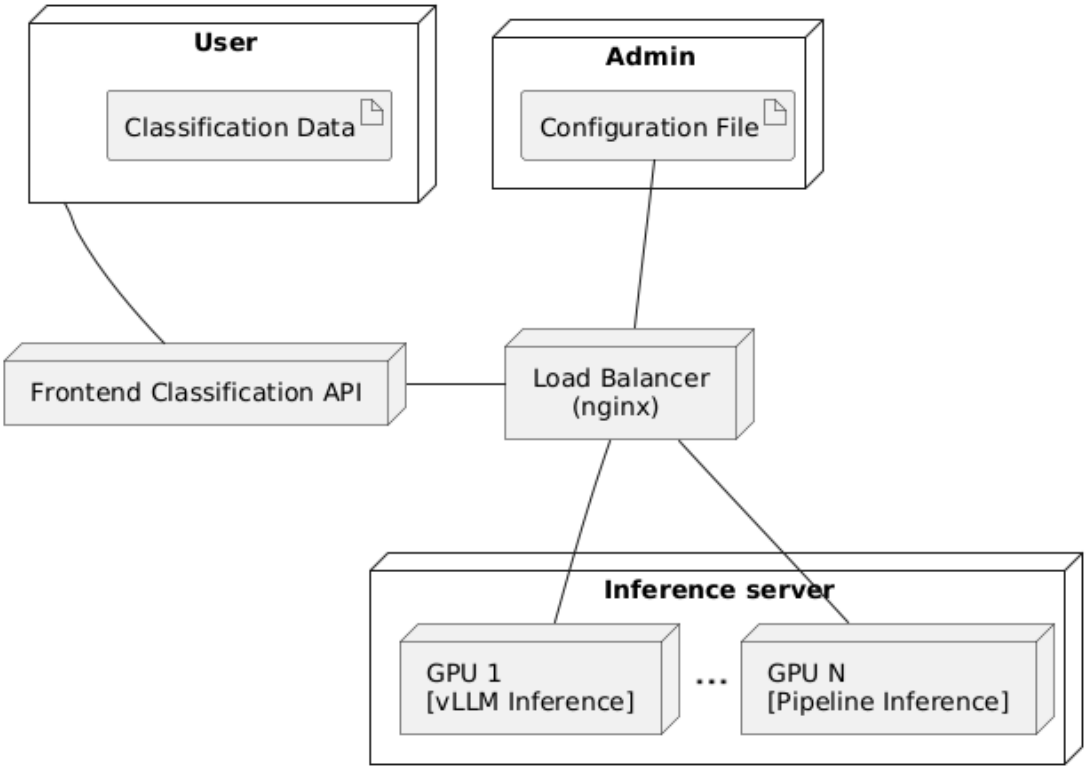

Рис. 3: Диаграмма развертывания системы

Блок пользователя моделирует внутренние сервисы или сотрудников компании, которые инициируют запросы на классификацию. Данный компонент содержит произвольные текстовые данные (сообщения, посты, комментарии и т. д.), агрегирует их в формат, совместимый с API фронтенда, и отправляет запросы на классификацию через API.

Блок администратора представляет собой интерфейс управления (CLI-утилиту), через который работники компании передают системе конфигурационные файлы. В конфигурации задаются используемые в системе модели универсальной классификации (произвольные БЯМ либо обученный в рамках данной работы универсальный классификатор) и указания, на каких серверах и GPU будут развернуты БЯМ, а на каких будет развернут разработанный универсальный классификатор.

Frontend Classification API – это слой, принимающий внешние запросы и приводящий их к унифицированной внутренней схеме. Его ключевая функция — агрегация нескольких запросов в наборы для оптимального заполнения GPU.

Модуль балансировки нагрузки реализован на базе nginx[11]. Он распределяет входящие наборы данных между несколькими экземплярами сервисов, на которых развернуты модели, по стратегии least connections. Кроме того, балансировщик выполняет:

- проверку работоспособности экземпляров сервисов;
- плавное отключение узлов перед развертыванием (graceful shutdown);
- базовую ретрансляцию запросов при временных ошибках.

Inference Server – это сервер, на котором развернуты экземпляры моделей и вспомогательные окружения. Общие задачи:

- приём наборов данных от балансировщика нагрузки и запуск модели на этих данных;
- выбор оптимальных технологий для использования модели (фреймворк vLLM[12] или разработанный конвейер классификации) на основании конфигурационных правил;
- Сбор данных о работе сервера (время обработки запроса, утилизация GPU и т.п.).

В системе есть два вида сервисов по использованию моделей, один из них предназначен для использования БЯМ, а другой — для использования разработанного классификатора. Сервисы, использующие БЯМ, работают через высокопроизводительный фреймворк vLLM, поддерживающий потоковый вывод и продвинутые техники управления памятью. Сервисы, использующие разработанный классификатор, работают через конвейер `ZeroShotClassificationPipeline`, который был разработан в рамках данной работы специально для использования разработанного универсального классификатора.

[11] Документация nginx https://nginx.org/en/docs/http/load_balancing.html (дата обращения 27.05.2025)

[12] Репозиторий vLLM https://github.com/vllm-project/vllm (дата обращения 27.05.2025)

4. Реализация обучения классификатора

В этом разделе описано, как был написан код обучения универсального классификатора и собран обучающий набор данных для него.

4.1. Детали реализации

Код обучения универсального классификатора был написан с помощью фреймворка PyTorch Lightning[13]. Данный фреймворк структурирует PyTorch код, устраняя шаблонные фрагменты. Благодаря унифицированному интерфейсу аргументы обучения легко конфигурируются через YAML-файл.

Для журналирования метрик и подбора гиперпараметров была использована платформа Weights & Biases (WandB). Она обеспечивает комплексное сопровождение экспериментов, объединяя журналирование метрик, управление артефактами и инструментарием для автоматизированного подбора гиперпараметров в единой облачной среде. Подсистема Hyperparameter Sweeps поддерживает как решётчатый, так и стохастический (Random, Bayesian) подбор гиперпараметров. Интеграция с PyTorch Lightning осуществляется через класс `WandbLogger`.

Для обучения классификатора был написан следующий код:

- Класс `GeRaClCore` – класс модели. В нем реализован алгоритм работы классификатора (он приведен на рисунке 2). Также в нем есть поддержка двух различных стратегий получения векторов текста и классов. Выбор стратегии зависит от типа кодировщика, на котором строится модель, и вынесен в отдельный конфигурационный YAML файл, в котором определяются все нужные параметры для обучения и использования классификатора.

- Класс `GeRaCl` — класс, который унаследован от `LightningModule`. В данном классе реализовано всё, что связанно с обучением классификатора: подсчет функции потерь, использование собран-

[13]Документация Pytorch Lightning https://lightning.ai/ (дата обращения 27.05.2025)

ного набора данных для обучения и логирование метрик обучения. Также `GeRaCl` использует абстрактный класс `TrainingUtils`, в котором определена фокальная функция потерь, которую можно выбрать для обучения вместо классической (бинарной кросс-энтропии), а также два планировщика темпа обучения (косинусный и линейный);

- Класс `DataModule` и абстрактный класс `DataHelper`. В них инкапсулирован процесс предобработки данных для обучения модели и реализована логика этого процесса для четырех типов обучающих выборок. В них входят такие функции, как:
  - Формирование входной строки для модели на основе входного текста, списка классов, названия сценария классификации и начального запроса к модели (последние два пункта опционально);
  - Перестановка позиций классов во входной строке модели;
  - Сбор данных в набор с настраиваемой размерностью. Данная операция нужна, чтобы модель могла обработать несколько примеров за один шаг, тем самым повышая эффективность работы графического процессора;
  - Адаптированный под обучающую выборку процесс токенизации данных.
- Скрипт `main.py` для запуска обучения классификатора. Он принимает на вход конфигурационный файл с параметрами классов `GeRaCl`, `DataModule`,а также класса `Trainer`, который инкапсулирует в себе унифицированный процесс обучения. Кроме этого, в файле указаны параметры журналирования метрик через WandB и сохранения промежуточных версий весов модели по ходу обучения.
- Скрипт `sweeps.py` для подбора гиперпараметров модели с помощью WandB Sweeps. Он принимает на вход конфигурационный

файл с перечислением названий гиперпараметров и множеств их допустимых значений.

4.2. Валидация классификатора

Валидация модели — это проверка её качества работы на данных, которые она ранее не встречала (то есть на которых модель не обучалась). Выделяют два дополняющих друг друга уровня валидации.

Первая — валидация на отдельной валидационной части исходной обучающей выборки. Её цель — оперативно контролировать переобучение и подбирать архитектурные и оптимизационные гиперпараметры. Такая валидация полезна для подбора гиперпараметров и ранней остановки обучения, однако многократное измерение метрик на одной и той же подвыборке постепенно переобучает модель под эту подвыборку, тем самым завышая метрики на валидации. Из-за этого необходима ещё одна тестовая подвыборка, которая полностью исключена из цикла разработки и используется единожды в конце, обеспечивая непредвзятую оценку реальной способности модели обобщаться на новые данные.

Вторая — валидация на реальных задачах классификации. Она проверяет, насколько модель переносит знания за пределы исходного домена и действительно полезна в реальных продуктовых сценариях. Для универсального классификатора критически важно, чтобы такая валидация включала в себя несколько различных и популярных в индустрии типов задач (классификация эмоций, темы текста и т.д.) — тогда оценка отражает устойчивость модели к разнообразным условиям эксплуатации. Дополнительно в набор следует включать задачи, максимально близкие к будущим сценариям использования модели в компании, чтобы заранее увидеть возможные проблемы при использовании модели на продуктовых задачах.

Для валидации были выбраны 5 задач из бенчмарка RU-MTEB [12] и одна продуктовая задача:

1. Kinopoisk sentiment classification — задача на классификацию отзывов из кинопоиска на три класса: позитивный, нейтральный,

негативный;

2. Headline Classification — задача классификации новостных заголовков на шесть классов: политика, экономика, происшествия, наука, спорт, культура;

3. Scibench GRNTI/OECD Classification — два набора данных для классификации темы научных статей по их заголовку и названию. В GRNTI наборе данных есть 28 классов, в OECD — 29;

4. Inappropriateness classification — набор данных на бинарную классификацию неприличных текстов;

5. Продуктовый набор данных на классификацию токсичности. Содержит пять классов. При этом одному тексту может быть присвоено несколько классов и в наборе данных есть сильный дисбаланс классов — токсичных текстов намного меньше, чем нейтральных.

Выбор данных пяти задач из RU-MTEB [12] обоснован необходимостью всесторонне протестировать универсальный классификатор на ключевых сценариях классификации текстов на русском языке. Во-первых, эти задания охватывают разные домены и стили: пользовательские рецензии, формализованные новостные заголовки, заголовки статей и контент из социальной сети. Во-вторых, они демонстрируют разнообразие сложности задач и числа классов — от бинарной классификации неприличных высказываний до 28–29 тематических категорий в научных статьях — что позволяет оценить способность модели справляться с разным объёмом и структурой текста. Наконец, все эти наборы входят в бенчмарк для кодировщиков RU-MTEB [12], что обеспечивает сопоставимость результатов с другими решениями для русского языка.

Так как в задачах из RU-MTEB [12] нет дисбаланса классов, то для оценки качества было принято решение использовать метрику точности, то есть отношение количества правильно классифицированных примеров к количеству всех примеров в задаче. Из-за наличия высо-

кого дисбаланса классов в продуктовом наборе данных было принято решение использовать macro F1-меру.

Для автоматизации процесса валидации на указанных задачах был разработан скрипт `validate.py`, который принимает на вход файл весов модели, который может находится на локальном хранилище либо на репозитории в HuggingFace. Далее скрипт валидирует модель на указанных задачах и сохраняет метрики в указанный во входных аргументах файл.

4.3. Выбор набора данных для обучения модели

Для обучения универсального классификатора на русском языке необходимо использовать разнообразный и обширный набор данных, который охватывает тексты из множества различных доменов.

В ходе обзора подходящих наборов данных был выделен deepvk/cultura_ru_edu[14]. В этом наборе данных представлены качественно отфильтрованные тексты на русском языке, отобранные из более общего набора данных mC4 (Multilingual Colossal Clean Crawled Corpus) [14]. В процессе фильтрации из набора были удалены тексты низкого качества, дублирующиеся записи и нежелательный контент, что обеспечивает высокое качество данных. Однако такая тщательная фильтрация приводит к тому, что из набора данных исключаются некоторые типы текстов, которые могут встречаться в Интернете, например, фрагменты веб-страниц, такие как заголовки сайтов или элементы пользовательского интерфейса.

Вместо этого для обучения был выбран оригинальный набор русскоязычных текстов из набора данных allenai/c4[15]. Несмотря на то, что этот набор данных содержит меньшую степень фильтрации и может включать тексты более низкого качества, его разнообразие значительно выше.

[14]Набор данных deepvk/cultura_ru_edu https://huggingface.co/datasets/deepvk/cultura_ru_edu (дата обращения 27.05.2025)

[15]Набор данных allenai/c4 https://huggingface.co/datasets/allenai/c4 (дата обращения 27.05.2025)

4.4. Сегментация набора данных allenai/c4

Перед разметкой текстов с использованием БЯМ для генерации классов было принято решение провести сегментацию текстов из набора данных allenai/c4. Этот этап предобработки данных позволяет более эффективно использовать вычислительные ресурсы за счёт уменьшения длины обрабатываемых фрагментов.

Использование текстов из allenai/c4 целиком в обучающей выборке могло бы привести к тому, что модель обучалась бы на малоинформативных фрагментах, снижая общую эффективность классификации. С другой стороны, полное исключение таких фрагментов из набора данных делает его менее разнообразным, тем самым делая модель, которая обучается на таком наборе данных, менее универсальной. Поэтому было принято решение сегментировать тексты из allenai/c4 и в обучающий набор данных для каждого текста из allenai/c4 добавлять случайно выбранный в нем сегмент.

Для этой задачи была выбрана библиотека razdel[16], разработанная для синтаксической сегментации русскоязычных текстов.

Процесс сегментации включал следующие шаги:

1. Разделение текстов на предложения: каждый текст был разбит на отдельные предложения с помощью вышеуказанной библиотеки, что позволило учесть синтаксическую структуру языка и избежать разрывов внутри предложений.

2. Формирование отрывков случайной длины: на основе полученных предложений тексты были разбиты на отрывки случайной длины в диапазоне от 20 до 150 слов. Такой подход позволил сохранить достаточный контекст для анализа, но при этом ограничил длину отрывков, чтобы они эффективно обрабатывались на этапе генерации классов.

После сегментации текстов из каждого исходного текста был случайным образом выбран один отрывок в диапазоне от 20 до 150 слов.

[16]Библиотека razdel https://github.com/natasha/razdel/ (дата обращения 27.05.2025)

4.5. Выбор моделей для генерации классов

Для обучения классификатора требуется разметка данных, которая представляет собой присвоение каждому тексту определённых классов. Для создания такой разметки было решено использовать БЯМ. Для генерации классов были выбраны две модели:

- meta-llama/Llama-3.3-70B-Instruct [9];
- GPT-4o [7].

На наборе из 50 текстов с помощью каждой модели были сгенерированы по пять списков из пяти релевантных к тексту классов, используя разные запросы.

Все результаты прошли двухэтапную проверку: сначала в ходе собственного анализа были выбраны два лучших списка классов для каждого текста, затем коллеги проголосовали за лучший из этих двух списков. По данной оценке лучшим оказался русскоязычный запрос к meta-llama/Llama-3.3-70B-Instruct [9], который был принят для дальнейшей генерации классов.

4.6. Эксперименты по разметке текстов

Обучающая выборка универсального классификатора должна включать большой набор уникальных классов, чтобы модель училась распознавать максимально широкий спектр категорий. Каждый обучающий пример при этом снабжается меткой одного «позитивного» класса (тот класс, которому пример действительно принадлежит) и нескольких «негативных» классов (другие категории, которые пример не представляет). Такой формат заставляет модель не просто учиться выделять целевую категорию, но и активно различать её на фоне множества конкурирующих классов, что повышает её устойчивость к ошибочным срабатываниям.

Было проведено четыре эксперимента по генерации обучающего набора данных с использованием БЯМ:

1. **Генерация классов к текстам**

   - С помощью БЯМ meta-llama/Llama-3.3-70B-Instruct для каждого фрагмента текста были сгенерированы до пяти релевантных позитивных классов.
   - Чтобы снабдить тексты негативными классами, берутся сгенерированные позитивные классы других текстов и рассматриваются как негативные для текущего текста. Ниже представлен пример генерации:

```
Текст: <<Интересно почитать - частные объявления...>>
Классы: [<<плавание>>, <<гражданство>>, <<реклама>>]
Индекс позитивного класса: 2
```

2. **Генерация сценариев классификации**

   - БЯМ было указано сформировать пять «сценариев» классификации, в каждом из которых должен быть один позитивный класс и несколько негативных классов, семантически близких к позитивному.
   - Кроме самого набора классов в данном методе генерировался сценарий классификации, который обозначал какой-то общий аспект сгенерированных классов.
   - Такой подход стимулирует модель лучше дифференцировать близкие по смыслу классы в контексте конкретного аспекта текста. Ниже представлен пример генерации:

```
Текст: <<Стараемся выбрасывать мусор в специальные урны...>>
Классы: [<<правительственное учреждение>>,
         <<группа активистов>>,
         <<частное предприятие>>]
Индекс позитивного класса: 1
Сценарий: Сценарий классификации по источнику высказывания
```

3. **Классификация по RU-MTEB**

   - Для каждого сегмента текстов брался один из шести наборов классов из RU-MTEB.
   - БЯМ давался случайный набор классов и предлагалось отнести текст к одному из них.

```
Текст: <<а в реале жуткие вещи - Кидалово...>>
Классы: [<<негативный>>, <<позитивный>>, <<нейтральный>>]
Индекс позитивного класса: 0
```

4. **Расширенная классификация по RU-MTEB**

   - Классы из RU-MTEB были дополнены пять-восемь вариациями, сгенерированными моделью OpenAI o3[17], вносящими лексические и смысловые детали в задачи из RU-MTEB.
   - Затем БЯМ классифицировала текст по расширенному или оригинальному списку классов, что повысило устойчивость к изменению формулировок классов в задаче.

```
Текст: <<Пиджак с рубашкой - хорошее решение для офиса...>>
Классы: [
<<смешанный отзыв (содержит и позитивные, и негативные
элементы)>>,
<<негативный отзыв>>,
<<позитивный отзыв>>
...]
Индекс позитивного класса: 2
```

Все запросы к БЯМ, используемые в этих экспериментах, доступны в репозитории набора данных[18].

[17]Блог OpenAI https://openai.com/index/introducing-o3-and-o4-mini/ (дата обращения 27.05.2025)

[18]Набор данных deepvk/GeRaCl_synthethic_dataset https://huggingface.co/datasets/deepvk/GeRaCl_synthethic_dataset (дата обращения 27.05.2025)

Для сравнения различных версий обучающей выборки полученный после обучения классификатор был валидирован на заданиях из RU-MTEB и продуктовом наборе данных с помощью скрипта `validate.py`.

В качестве итоговой обучающей выборки оказалась наиболее эффективной смешанная стратегия — были взяты примеры из второго (без использования сценариев), третьего и четвёртого эксперимента и соединены в одну обучающую выборку.

4.7. Подбор гиперпараметров

После фиксирования наиболее качественной обучающей выборки был проведен подбор гиперпараметров обучения для модели deepvk/USER2-base[19] с помощью скрипта `sweeps.py`. Конфигурация подбора приведена ниже:

```
sweep_configuration = {
    ''method'': ''grid'',
    ''name'': ''my_sweep'',
    ''metric'': {
        ''name'': ''val/epoch_accuracy'',
        ''goal'': ''maximize'',
    },
    ''parameters'': {
        ''max_lr'': {''values'': [1e-6, 5e-6, 1e-5, 5e-5]},
        ''weight_decay'': {''values'': [0, 0.01, 0.1]}
    },
}
```

Данная конфигурация действует по принципу перебора – было обучено столько версий модели, сколько возможно различных комбинаций значений указанных гиперпараметров.

Другие фиксированные гиперпараметры:

- Размер пакета – 32;
- Планировщик темпа обучения – линейный;

[19]Модель deepvk/USER2-base https://huggingface.co/deepvk/USER2-base (дата обращения 27.05.2025)

- Максимальное количество эпох обучения – 5;
- Количество шагов прогрева (warmup steps) – 1000;
- Оптимизатор – AdamW.

Наилучшая модель без признаков переобучения имела параметры `«max_lr» = 5e-6` и `«weight_decay» = 0.01`. Подбор гиперпараметров улучшил точность работы модели на заданиях из RU-MTEB с 58 % до 64 %.

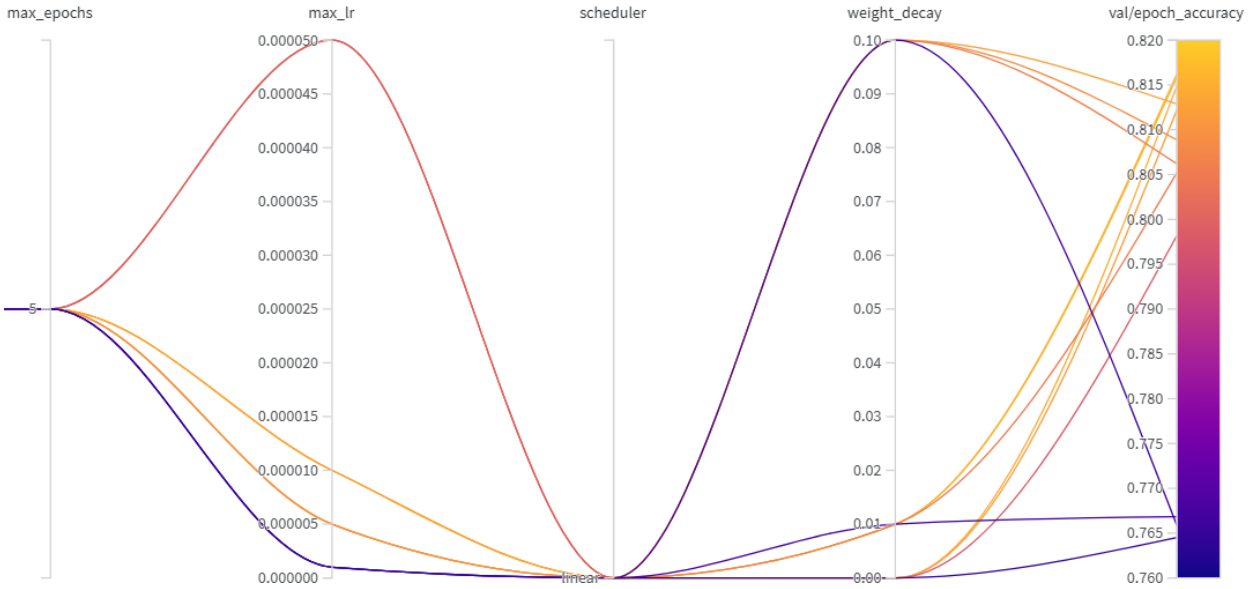

Рис. 4: Визуализация подбора гиперпараметров

5. Апробация системы

Для апробации написанного кода было проведено обучение и подбор гиперпараметров для двух кодировщиков: deepvk/USER-bge-m3[20] (359 миллионов параметров) и deepvk/USER2-base[21] (149 миллионов параметров). Данные модели были выбраны, потому что они являются разработками компании ВК и также являются одними из самых лучших кодировщиков из множества моделей со схожим количеством параметров, что отражено в таблице лидеров MTEB[22] для моделей, нацеленных на русскоязычную аудиторию (deepvk/USER-bge-m3 является наилучшим кодировщиком из моделей с количеством параметров до 500 миллионов и при этом не имеющих текстов из RU-MTEB в своих обучающих выборках). Обученные версии моделей называются GeRaCl-USER2-base и GeRaCl-USER-bge-m3

Для сравнения с другими подходами к универсальной классификации были выбраны ещё четыре популярные модели с различным количеством параметров:

- Два NLI-кодировщика:
    1. MoritzLaurer/mDeBERTa-v3-base-mnli-xnli (279 миллионов параметров, 722 тысячи ежемесячных скачиваний на Hugging Face);
    2. MoritzLaurer/bge-m3-zeroshot-v2.0 (568 миллионов параметров, 70 тысяч ежемесячных скачиваний на Hugging Face).
- Две БЯМ:
    1. Qwen/Qwen2.5-1.5B-Instruct (1.5 миллиарда параметров, два миллиона ежемесячных скачиваний на Hugging Face);

[20]Модель deepvk/USER-bge-m3 https://huggingface.co/deepvk/USER-bge-m3 (дата обращения 27.05.2025)
[21]Модель deepvk/USER2-base https://huggingface.co/deepvk/USER2-base (дата обращения 27.05.2025)
[22]Таблица лидеров RU-MTEB https://huggingface.co/spaces/mteb/leaderboard (дата обращения 27.05.2025)

2. Qwen/Qwen2.5-3B-Instruct (три миллиарда параметров, 1.4 миллиона ежемесячных скачиваний на Hugging Face);

Выбранные БЯМ и обученные GeRaCl-USER2-base и GeRaCl-USER-bge-m3 модели были использованы с помощью разработанной распределенной системы.

Результаты сравнения качества на заданиях из RU-MTEB находятся в Таблице 2. Для оценки качества использовалась метрика точности работы. Из таблицы видно, что обучение модели дает прирост метрик в среднем на 20 %, при этом увеличивая размер кодировщика в среднем на три процента. Также после обучения метрики модели GeRaCl-USER2-base стали выше, чем у модели Qwen/Qwen2.5-3B-Instruct, несмотря на то что Qwen/Qwen2.5-3B-Instruct имеет примерно в 20 раз больше параметров, чем GeRaCl-USER2-base.

Результаты сравнения качества на продуктовой задаче находятся в Таблице 3. Из таблицы видно, что самая эффективная модель для этой задачи — NLI кодировщик MoritzLaurer/bge-m3-zeroshot-v2.0. Стоит отметить, что модель deepvk/USER2-base занимает второе место по метрикам на данной задаче и имеет примерно в три раза меньше параметров, чем указанный NLI кодировщик.

Проведенные эксперименты показывают, что предложенный алгоритм обучения универсального классификатора улучшает метрики на реальных задачах классификации в индустрии в среднем на 28 процентов относительно метрик модели до обучения (средний прирост был посчитан по двум обученным моделям и на метриках из RU-MTEB и из продуктовой задачи), при этом увеличивая размер модели в среднем на четыре процента.

Model	Type	Size	Sentiment	Headlines	GRNTI	OECD	Inapp.	Avg.
mDeBERTa-v3-base-mnli-xnli	NLI Кодировщик	279M	0.54	0.53	0.34	0.23	0.62	0.45
bge-m3-zeroshot-v2.0	NLI Кодировщик	568M	**0.65**	0.72	0.53	0.41	0.67	0.60
Qwen2.5-1.5B-Instruct	БЯМ	1.5B	0.62	0.55	0.51	0.41	0.71	0.56
Qwen2.5-3B-Instruct	БЯМ	3B	0.63	0.74	0.60	0.43	**0.75**	0.63
USER-bge-m3	Кодировщик	359M	0.60	0.73	0.43	0.28	0.62	0.53
GeRaCl-USER-bge-m3	Кодировщик	367M	0.63	**0.81**	0.57	0.44	0.66	0.62
USER2-base	Кодировщик	149M	0.50	0.65	0.56	0.39	0.51	0.52
GeRaCl-USER2-base	Кодировщик	155M	0.61	0.80	**0.63**	**0.48**	0.71	**0.64**

Таблица 2: Результаты сравнения качества на RU-MTEB

Model	Type	Size	Macro F1
mDeBERTa-v3-base-mnli-xnli	NLI Кодировщик	279M	0.17
bge-m3-zeroshot-v2.0	NLI Кодировщик	568M	**0.32**
Qwen2.5-1.5B-Instruct	БЯМ	1.5B	0.13
Qwen2.5-3B-Instruct	БЯМ	3B	0.24
USER-bge-m3	Кодировщик	359M	0.15
GeRaCl-USER-bge-m3	Кодировщик	367M	0.18
USER2-base	Кодировщик	149M	0.11
GeRaCl-USER2-base	Кодировщик	155M	0.24

Таблица 3: Результаты сравнения качества на продуктовой задаче

Заключение

В рамках данной работы были выполнены следующие задачи:

- Проведён обзор трех популярных подходов к универсальной классификации — NLI кодировщики, БЯМ и кодировщики без дообучения. Был выполнен их сравнительный анализ на пяти актуальных и разнообразных наборах данных из бенчмарка RU-MTEB. В результате сравнения было доказано, что компактные кодировщики могут так же эффективно справляться с задачей универсальной классификации, как и БЯМ.

- Было спроектировано решение для универсальной классификации текстов, включающее алгоритм для универсальной классификации, основанный на модели GLiNER, а также архитектуру программной системы для эффективного использования классификатора.

- Была сформирована обучающая выборка для классификатора, основанная на наборе текстов из allenai/c4, и написан код обучения произвольного кодировщика для задачи универсальной классификации с помощью фреймворка PyTorch-Lightning. С его помощью были обучены два кодировщика: deepvk/USER2-base и deepvk/USER-bge-m3.

- Была разработана программная система, позволяющая использовать обученный классификатор и произвольные БЯМ для классификации текстов, используя FastAPI, nginx в качестве балансировщика нагрузки, а также vLLM в качестве фреймворка для оптимизированного использования БЯМ соответственно и класс ZeroShotClassificationPipiline из репозитория geracl для оптимизированного использования GeRaCl моделей.

- В результате апробации системы средний относительный прирост метрик от обучения был равен 28 процентов, а размер модели в ходе обучения увеличивался в среднем на четыре процента. После

обучения качество работы моделей стало сопоставимым с БЯМ Qwen2.5-3B-Instruct, которая имеет примерно в восемь раз больше параметров, чем GeRaCl-USER-bge-m3 и примерно в 20 раз больше параметров, чем GeRaCl-USER2-base.

Код обучения классификатора доступен по ссылке:
https://github.com/deepvk/geracl.

Код распределенной системы для классификации текстов доступен по ссылке:
https://github.com/VyrodovMikhail/zero_shot_classification.

Набор данных для обучения классификатора доступен по ссылке:
https://huggingface.co/datasets/deepvk/GeRaCl_synthethic_dataset.

Модель deepvk/GeRaCl-USER2-base, обученная на основе deepvk/USER2-base доступна по ссылке:
https://huggingface.co/deepvk/GeRaCl-USER2-base.

Список литературы

[1] Vaswani Ashish, Shazeer Noam, Parmar Niki et al. Attention Is All You Need. — 2023. — URL: https://arxiv.org/abs/1706.03762 (дата обращения: 2025-05-27).

[2] BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension / Mike Lewis, Yinhan Liu, Naman Goyal et al. // CoRR. — 2019. — Vol. abs/1910.13461. — URL: http://arxiv.org/abs/1910.13461 (дата обращения: 2025-05-27).

[3] Devlin Jacob, Chang Ming-Wei, Lee Kenton, Toutanova Kristina. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. — 2019. — URL: https://arxiv.org/abs/1810.04805 (дата обращения: 2025-05-27).

[4] Laurer Moritz, van Atteveldt Wouter, Casas Andreu, Welbers Kasper. Building Efficient Universal Classifiers with Natural Language Inference. — 2023. — . — arXiv:2312.17543 [cs]. URL: http://arxiv.org/abs/2312.17543 (дата обращения: 2024-01-05).

[5] He Pengcheng, Liu Xiaodong, Gao Jianfeng, Chen Weizhu. DeBERTa: Decoding-enhanced BERT with Disentangled Attention. — 2021. — URL: https://arxiv.org/abs/2006.03654 (дата обращения: 2025-05-27).

[6] Zaratiana Urchade, Tomeh Nadi, Holat Pierre, Charnois Thierry. GLiNER: Generalist Model for Named Entity Recognition using Bidirectional Transformer. — 2023. — URL: https://arxiv.org/abs/2311.08526 (дата обращения: 2025-05-27).

[7] OpenAI, Hurst Aaron, Lerer Adam et al. GPT-4o System Card. — 2024. — URL: https://arxiv.org/abs/2410.21276 (дата обращения: 2025-05-27).

[8] Gemma 2: Improving Open Language Models at a Practical Size / Gemma Team, Morgane Riviere, Shreya Pathak et al. — 2024. — URL: https://arxiv.org/abs/2408.00118 (дата обращения: 2025-05-27).

[9] Grattafiori Aaron, Dubey Abhimanyu, Jauhri Abhinav et al. The Llama 3 Herd of Models. — 2024. — URL: https://arxiv.org/abs/2407.21783 (дата обращения: 2025-05-27).

[10] Muennighoff Niklas, Tazi Nouamane, Magne Loïc, Reimers Nils. MTEB: Massive Text Embedding Benchmark. — 2023. — URL: https://arxiv.org/abs/2210.07316 (дата обращения: 2025-05-27).

[11] Gao Leo, Biderman Stella, Black Sid et al. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. — 2020. — URL: https://arxiv.org/abs/2101.00027 (дата обращения: 2025-05-27).

[12] Snegirev Artem, Tikhonova Maria, Maksimova Anna et al. The Russian-focused embedders' exploration: ruMTEB benchmark and Russian embedding model design. — 2025. — URL: https://arxiv.org/abs/2408.12503 (дата обращения: 2025-05-27).

[13] Team Qwen. Qwen2.5: A Party of Foundation Models. — 2024. — September. — URL: https://qwenlm.github.io/blog/qwen2.5/ (дата обращения: 2025-05-27).

[14] Xue Linting, Constant Noah, Roberts Adam et al. mT5: A massively multilingual pre-trained text-to-text transformer. — 2021. — URL: https://arxiv.org/abs/2010.11934 (дата обращения: 2025-05-27).