ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГАОУ ВО НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Образовательная программа «Прикладная математика и информатика»

УДК XXXXX

Отчет об исследовательском проекте на тему:

Авторегрессионные языковые модели для задачи исправления опечаток в текстах на русском языке

Выполнил:

студентка группы MMOBC21_3
Гаврилова Анастасия Алексеевна _____ _____
(подпись) (дата)

Принял руководитель проекта:

Феногенова Алена Сергеевна
Team Lead команды прототипировния AGI NLP,
SaluteDevices.

_____ _____
(подпись) (дата)

Москва 2023

Содержание

Аннотация

Исследования в области обработки естественного языка не стоят на месте. Все больше и больше NLP-задач получается решать с помощью авторегрессионных моделей. Исправление ошибок и опечаток в тексте – одна из всегда актуальных задач, которая разбирается и исследуется в данной работе. Произведен обзор перспективных авторегрессионных моделей, таких как T5, ruGPT-3, которые неплохо себя зарекомендовали в задачах такого типа. На базе этих моделей произведен файнтюнинг с помощью синтетического датасета, собранного из различных источников. Результаты обученных моделей в различных вариациях представлены в сравнении с результатами других популярных решений задачи спеллчекинга на данных соревнования SpellRuEval.

Ключевые слова

Спеллчекинг, авторегрессионные модели, нейронные сети, обработка естественного языка, NLP, ruGPT3, ruT5

1 Введение

1.1 Описание предметной области

Автоматическая коррекция орфографии — одна из старейших и важнейших задач вычислительной техники. Она привлекала многих исследователей, особо известны работы Левенштейна [1] и Дамерау [2] в 60-е годы, Кернигана в 90-е [3]. Первые ранние контекстно-ориентированные методы появились в работе Голдинга и Рота в 1999-м году [4]. Задача автоматической проверки орфографии имеет широкое практическое применение — исправление запросов и дополнения, используемые в поисковых системах, корректоры орфографии, которые являются частью любого современного текстового редактора, комментарии в соцсетях, чатботы, где исправить ошибку бывает непросто из-за обилия разговорной лексики.

1.2 Постановка задачи

В связи с актуальностью данной проблемы для любого, в том числе и для русского языка, существует множество статей и исследований для построения системы исправления ошибок, некоторые из них описаны в данной работе. Кроме того, проводятся соревнования, например, в 2016 году проходило соревнование по спеллчекингу на русском языке SpellRuEval [5]. В настоящее время в различных задачах NLP широко используются авторегрессионные модели, есть обзорная статья о возможностях генеративных моделей в сфере NLP от A. Gillioz, J. Casas, E. Mugellini, O. A. Khaled [6], однако исследований по их применению в задаче спеллчекинга для русского языка крайне мало.

Целью данной работы является оценка возможностей генеративных моделей для решения задачи исправления орфографических ошибок в текстах на русском языке. Для корректного эксперимента были дообучены (fine-tune) модели ruT5(base) и ruGPT3 для решения данной задачи. Оценка и сравнительный анализ были проведены на открытом датасете для русского языка SpellRuEval [5].

1.3 Структура работы

Работа организована следующим образом. В разделе 2 дается обзор статей по задаче спеллчекинга. Далее, в разделе 3 представлены описания моделей T5 и GPT, взятых за основу в данной работе. Дана базовая информация о моделях, об их архитектуре и процессе обучения. Подготовка и проведение экспериментов представлены в разделе 4. В подразделе 4.1 описан процесс подготовки датасета для файнтюнинга моделей. В последующих

подразделах представлены экспериментальные результаты в разных вариациях. Проведён сравнительный анализ результатов, полученных в рамках экспериментов, с лучшими решениями соревнования SpellRuEval. Конечные выводы данного исследования можно найти в разделе 5.

2 Обзор литературы

Распространенным подходом к решению задачи исправления ошибок в текстах является использование моделей зашумленного канала и n-граммных моделей. Впервые контекстно-зависимое решение, использующее модель зашумленного канала и биграммы было описано ATT Bell Labs в 1990 [7]. Примерно в те же годы IBM представила похожую модель на основе триграмм [8]. В 2020 году разработана система Symmetric Delete Algorithm (SDA), использующая взвешенную сумму униграмм, биграмм и триграмм для ранжирования полученных SDA кандидатов слов [9]. В 2021 году вышла статья о применении модели BERT в задаче спеллчекинга для английского языка, модель показала неплохие результаты [10].

Исследований в данной области для русского языка не так много, одной из таких статей является работа с соревнования Dialogue Дерезы О. В., Каютенко Д. А. ,Маракасовой А. А., Феногеновой А. С, “Комплексный подход к автоматическому исправлению опечаток для русского языка” [11]. В работе описывается комплексный набор методов для обнаружения и исправления ошибок и опечаток в текстах на русском языке. Описанная система состоит из классификатора, который обнаруживает потенциальные ошибки, и корректора, который обрабатывает слова, помеченные классификатором. В классификаторе ошибок в качестве признака для обучения использовались результаты векторного представления слов из модели word2vec, обученной на текстах, содержащих как правильные написания, так и опечатки. Гибридную модель ошибок объединяет три подхода: традиционную модель зашумленного канала; модель, представленную Э. Бриллом и Р. Муром [12], и альтернативную версию модели зашумленного канала с расширенным контекстом.

Другой работой, представленной на этом же соревновании, является работа Сорокина А. А. и Шавриной Т. О., “Автоматическое исправление опечаток и орфографических ошибок для русскоязычных социальных медиа” [13]. Описанная в данной статье система заняла первое место в первом соревновании SpellRuEval по автоматическому исправлению опечаток для русского языка, достигнув F1-меры в 75%. Алгоритм включает несколько методов, в том числе основанных на расстоянии Левенштейна и словарном поиске, а также контекстном ранжировании гипотез с помощью алгоритмов машинного обучения.

В исследовании Паниной М. Ф., Байтина А. В., Галинской И. Е ”Автоматическое исправление опечаток в поисковых запросах без учета контекста” [14] решалась проблема повышения эффективности автоматического исправления ошибок в поисковых запросах.. Для принятия решения о возможности автоматического исправления использовался бинарный классификатор, разделяющий исправления на надежные, пригодные для автозамен, и ненадежные, пригодные только для подсказок, для определения надежности исправлений использованы наиболее распространенные лексические и статистические признаки.

В настоящее время в данной области активно ведутся разработки для получения наиболее эффективного и применимого на практике решения. Не так давно DeepPavlov выпустила две модели для задачи спеллчекинга на русском языке – brillmoore и levenshtein-corrector. Кроме того, недавно была представлена система исправления ошибок для русского языка, реализованная на T5, специальной многозадачной нейросетевой модели для решения задач генерации и понимания текста, представленной компанией Google в 2019 году [15].

3 Описание предлагаемого метода

3.1 T5

3.1.1 Описание модели

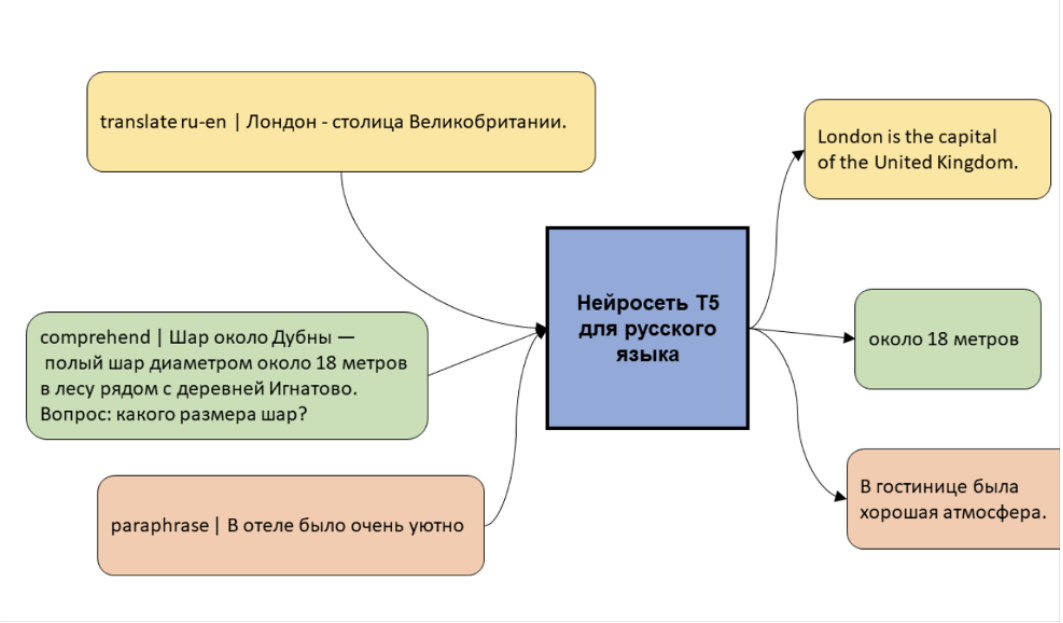

Рис. 3.1: Пример диаграммы Text-to-Text фреймворка.

Модель “Text-to-Text-Transfer-Transformer” была представлена в статье [15] от команды из Google. Модель T5 имеет энкодерно-декодерную архитектуру, которая больше всего

подходит для задач seq2seq. Тем самым авторы сделали значительный шаг в развитии архитектур для более качественного решения задач, связанных с NLP.

Для предобучения T5 использовался датасет C4 (Colossal Clean Crawled Corpus). Его размер составляет примерно 700 гигабайтов, и он представляет собой очищенную версию набора данных Common Crawl, объем которой примерно в 2 раза превышает объем Википедии. Очистка данных состояла в удалении дубликатов, предложений с пропусками, некорректных и мусорных текстов. Подобного рода фильтр приводит к получению наиболее качественных результатов в прикладных задачах.

Исследователи помимо предобучения предложили обучать модель в многозадачном режиме и представляли различные задачи (суммаризация, перевод, перафразирование и другие) в текстовом формате, то есть вход и выход модели были представлены в виде текстовых строк (рис. 3.1). Слово “transfer” нам говорит о том, что это предобученная модель, и мы можем дообучить ее на маленьком датасете, при этом знания основной модели останутся. Другими словами, произойдет перенос знаний. Для файнтюнинга модели ко входу добавляется префикс, таким образом веса модели оптимизируются под определенную задачу.

Google выпустил две версии T5: первая понимает только английский язык, зато дообучалась на 24 разных задачах, а вторая, mT5, понимает 101 язык (включая русский), но умеет только заполнять пропуски в тексте [16]. В 2021 году команда SberDevices выпустила модели ruT5-large и ruT5-base, каждая из них использовались для экспериментов в данной работе [17].

3.1.2 Архитектура

Существует три больших класса трансформерных архитектур:

1 **Encoder** (BERT)

2 **Decoder** (GPT)

3 **Encoder-Decoder** (T5)

Впервые модель трансформера была предложена в статье “Attention Is All You Need” [18] в 2017-м году. Верхнеуровнево трансформер состоит из кодирующего элемента, декодирующего и связи между ними (см. рисунок 3.2). Возможны вариации без кодирующего или без декодирующего элемента.

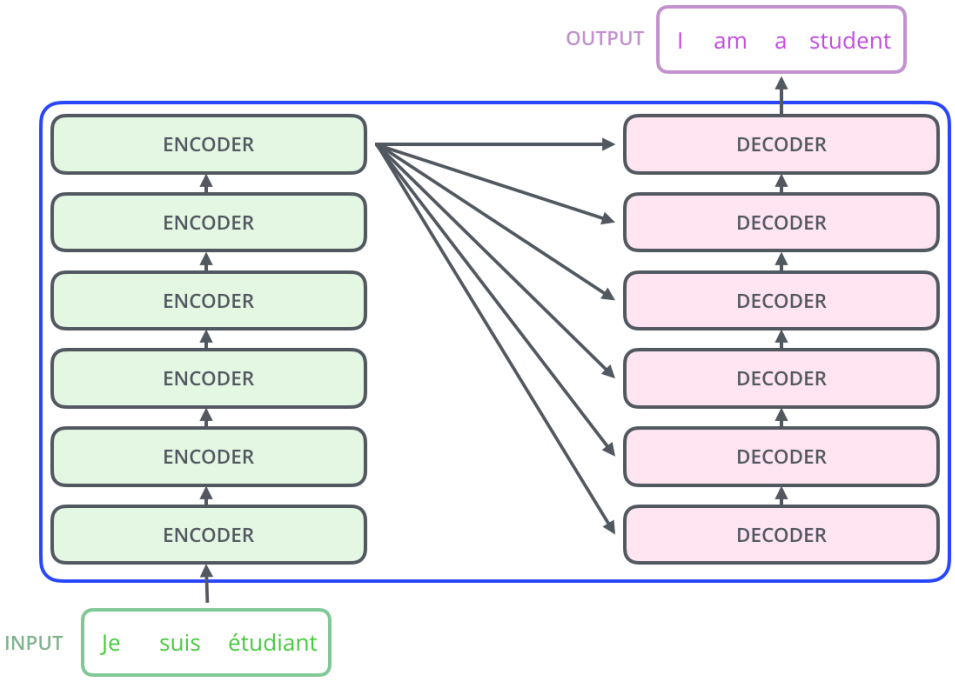

Рис. 3.2: Базовая архитектура трансформера

Кодирующий компонент – это стек энкодеров, декодирующий компонент – это стек декодеров. Все энкодеры идентичны по структуре, хотя и имеют разные веса. Каждый можно разделить на два подслоя:

1 Входная последовательность, поступающая в энкодер, сначала проходит через слой внутреннего внимания (self-attention), помогающий энкодеру посмотреть на другие слова во входящем предложении во время кодирования конкретного слова.

2 Выход слоя внутреннего внимания отправляется в нейронную сеть прямого распространения (feed-forward neural network). Точно такая же сеть независимо применяется для каждого слова в предложении.

Декодер также содержит два этих слоя, но между ними есть слой внимания, который помогает декодеру фокусироваться на релевантных частях входящего предложения (это схоже с тем, как механизм внимания организован в моделях seq2seq). На рисунке 3.3 представлено визуальное отображение слоев.

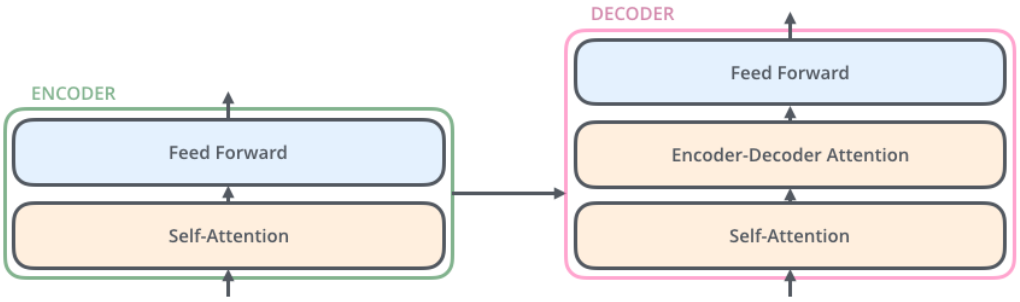

Рис. 3.3: Устройство слоев в трансформере

На базе существующих типов архитектур исследователи из Google в своей статье [15] предложили 3 архитектурных подхода, потенциально подходящих для языковых моделей. Визуально архитектуры представлены на рисунке 3.4.

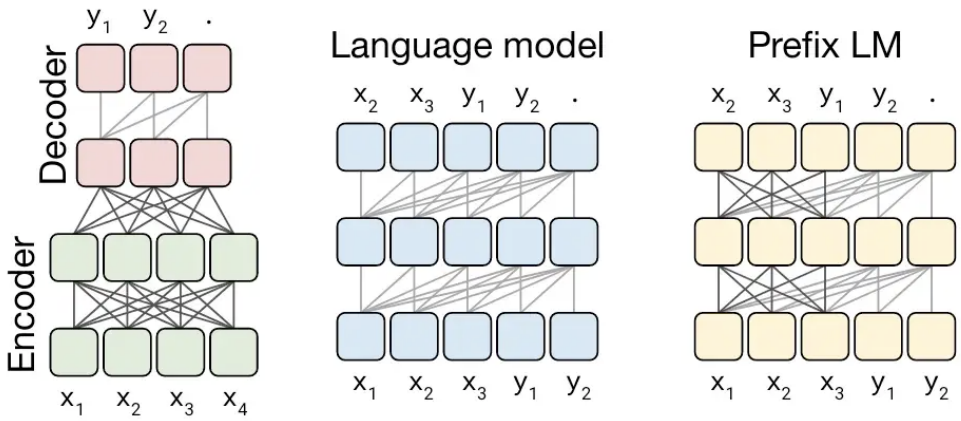

Рис. 3.4: Архитектурные подходы T5

1 **Endoder-Decoder**. Левая диаграмма на рисунке 3.4 отображает стандартную архитектуру энкодер-декодер – seq2seq архитектура, в которой энкодер обучается по аналогиии с BERT-ом, полностью видимым образом (т.е. каждый токен вносит вклад в вычисление attention каждого другого токена в последовательности), и декодер обучается каузальным способом в стиле GPT (т.е. каждый токен сопровождается всеми токенами, которые встречаются перед ним в последовательности).

2 **Language Model**. Состоит только из блока декодера. Получает конкатенацию входных и целевых данных с использованием каузальной маски, выход относится только к прошлому входу или выходу.

3 **Prefix Language Model**. Очень похожа на стандартную языковую модель, просто любой вывод будет относиться к определенной части ввода, которая содержит префикс,

может содержать информацию, специфичную для задачи, например, “Перевод с английского на немецкий: ”.

В ходе экспериментов наилучшие результаты были получены при использовании архитектуры энкодер-декодер.

3.1.3 Предварительное обучение T5

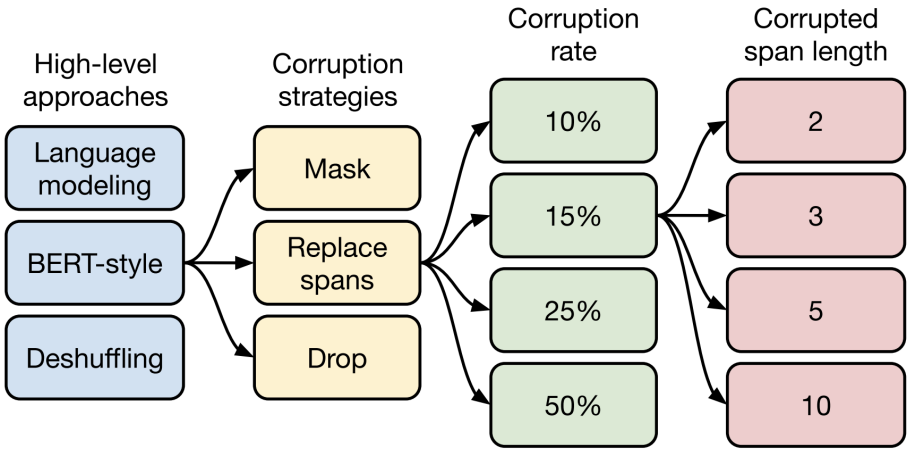

Рис. 3.5: Стратегии претрейна T5.

Что касается предварительной подготовки модели, авторы статьи [15] также изучили некоторые подходы на практике.

1 **Языковая модель (LM)**: этот подход включает в себя задачу причинно-следственного прогнозирования, то есть прогнозирование следующего слова в предложении с учетом всех слов, предшествующих этому слову (левый контекст).

2 **Deshuffling**: все слова в предложении перемешиваются, и модель обучается предсказывать исходный текст.

3 **Corrupting Spans (BERT-style)**: маскирование последовательности слов из предложения и обучение модели распознаванию маскированных токенов (или последовательности токенов), процесс маскирования отражен на рисунке 3.6.

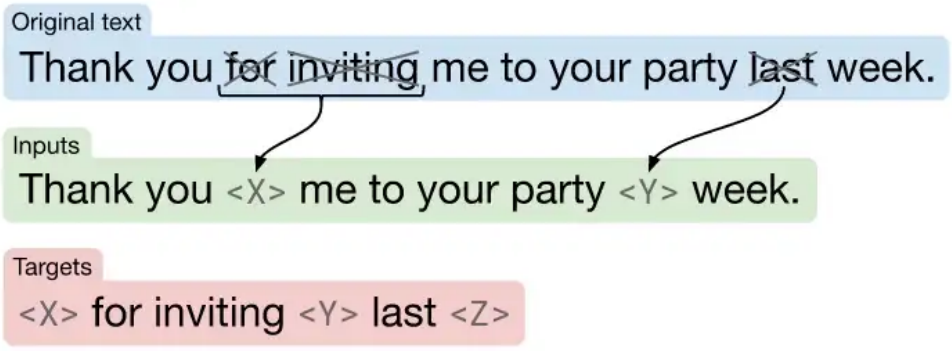

Рис. 3.6: Corrupting Spans (random)

Результаты исследования показали, что BERT-style подход на базе зашумления последовательностей продемонстрировал наилучшие результаты. При этом были протестированы разные стратегии зашумления:

- **Mask**: простая стратегия, в которой не включается этап случайной замены токенов; происходит замена 15 % токенов во входных данных токеном маски, и модель обучается восстанавливать исходный незашумленный текст.
- **Replace spans**: вместо замены каждого зашумленного токена заменяется вся последовательность зашумленных токенов уникальным токеном маски.
- **Drop**: зашумленные токены полностью отбрасываются из входной последовательности; перед моделью ставится задача восстановить отброшенные токены по порядку.

Сравнительный анализ вышеописанных трех стратегий приводит к выводу, что стратегия замены зашумленных последовательностей (Replace spans) работает лучше всего. Более того, применяются разные коэффициенты зашумления от 10% до 50%. Лучшим оказалось зашумление с коэффициентом 15%.

3.2 GPT

3.2.1 Описание

До появления трансформеров основным инструментом для решения NLP задач были реккурентные нейронные сети (RNN, LSTM). В 2016 году представлен механизм attention, который сильно улучшал качество этих моделей [19]. Годом позже исследователи в компании Google предположили, что attention может решать задачи NLP и без RNN – так появился первый трансформер энкодерно-декодерной архитектуры [18]. Затем в OpenAI решили

оставить только декодерную часть архитектуры трансформера – так появилась авторегрессионная языковая модель GPT «Generative Pre-trained Transformer» [20]. Идея этой модели аналогична любой модели генерации текста, т. е. она принимает на вход текстовые данные и генерирует текст в качестве результата. Количество параметров модели варьируется от 100 миллионов до 175 миллиардов (в зависимости от версии модели).

1 **GPT-1**: Первая версия модели содержала 12 слоев и была обучена на текстах из 7000 книг. Дообучение на некоторых задачах NLP позволило модели выбить SOTA результаты, однако с генерацией длинных текстов модель не справлялась, максимальная длина контекста у GPT-1 составила 512 токенов [20].

2 **GPT-2**: Модель следующего поколения была в разы больше предшествующей – 48 слоев и 1.5 млрд. параметров. К данным для обучения, помимо книг, добавили 8 млн. сайтов – всего объем данных составил 40 гб текста. Архитектурно исследователи переместили слои нормализации. В результаты GPT-2 обучилась генерировать длинные связные тексты, решать новые задачи. Максимальный размер контекста GPT-2 – 1024 токена [21].

3 **GPT-3**: По сравнению со своими предшественниками размеры GPT-3 значительно увеличились за счет датасета Common Crawl (веб-архив содержит порядка триллиона слов, в сумме 570 гб текста). Провели небольшую оптимизацию attention. Спектр возможностей модели стал еще шире, например, модель научилась генерировать рабочий программный код (CODEX) [22]. Максимальный размер контекста – 2048 токенов [23].

По сравнению со своими предшественниками размеры GPT-3 значительно увеличились за счет датасета Common Crawl (веб-архив содержит порядка триллиона слов). Пайплайн формирования обучающего набора данных был следующий:

1 Фильтрация Common Crawl на базе сходства с другими высококачественными наборами.

2 Нечеткая дедубликация внутри документов и между наборами.

3 Обогащение датасетами:

- WebText - набор с текстами полезных по контенту сайтов,
- Books1, Books2 - корпуса книг,
- Англоязычная Википедия.

Во время обучения наборы данных отбирались не пропорционально их размеру. Более качественные наборы выбирались чаще, так что наборы данных Common Crawl и Books2 выбирались менее одного раза во время обучения, а другие наборы данных отбирались по 2–3 раза.

3.2.2 Токенизация

В моделях машинного обучения используются числовые представления текста. Самый очевидный способ перевести текст в численный вектор – назначить каждому символу определенное число, но в этом случае текст после токенизации был бы слишком длинным, а это усложнило бы процесс обучения модели. Решить эту проблему можно с помощью BPE (Byte Pair Encoding) – способ токенизации, при котором слова разбиваются на общие части и представляются как комбинация этих последовательностей, пример такого разбиения иллюстрирует рисунок 3.7. OpenAI использовали byte-level BPE – модификацию BPE, которая работает не с текстом, а напрямую с его байтовым представлением. Такой подход позволяет сильно сократить расходы памяти – размер общего словаря составляет 50 тысяч токенов для любых слов и символов.

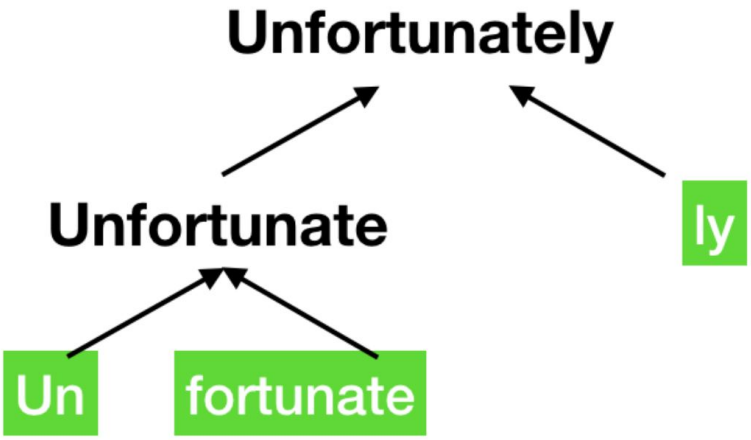

Рис. 3.7: Принцип работы BPE

3.2.3 Адаптация модели

Для адаптации предобученной языковой модели к конкретной задаче был предложен следующий метод. Модель принимает на вход краткое описание задачи, определяется условие на естественном языке и указывается ряд примеров. Ожидается, что модель выполнит поставленную перед ней задачу, так как она предобучена на огромном корпусе текстов. Такой метод схож с человеческим мозгом, который обучался много лет и сформировал множество навыков и способностей. Человеку часто бывает достаточно краткого указания, чтобы

понять, что делать. Такой способ обучения позволяет обойти ограничения, связанные с необходимостью собирать большие наборы данных, чтобы донастроить предобученную модель для выполнения конкретной задачи.

В рамках «контекстного обучения» GPT-3 для каждой задачи применялось три вида настроек:

1 **Few-Shot learning**. На вход предобученной модели подается описание задачи на естественном языке и несколько примеров ожидаемых результатов.

2 **One-Shot learning**. Очень похож на Few-Shot, но при использовании этого метода допускается лишь один пример в дополнение к описанию задачи на естественном языке.

3 **Zero-Shot learning**. Аналогичен One-Shot, но демонстрации не допускаются, а модель получает лишь инструкцию на естественном языке, без каких-либо примеров.

Эти методы являются серьёзным шагом в языковом моделировании, так как позволяют пропустить этап дообучения модели на конкретную задачу, а значит экономят значительное количество времени и ресурсов.

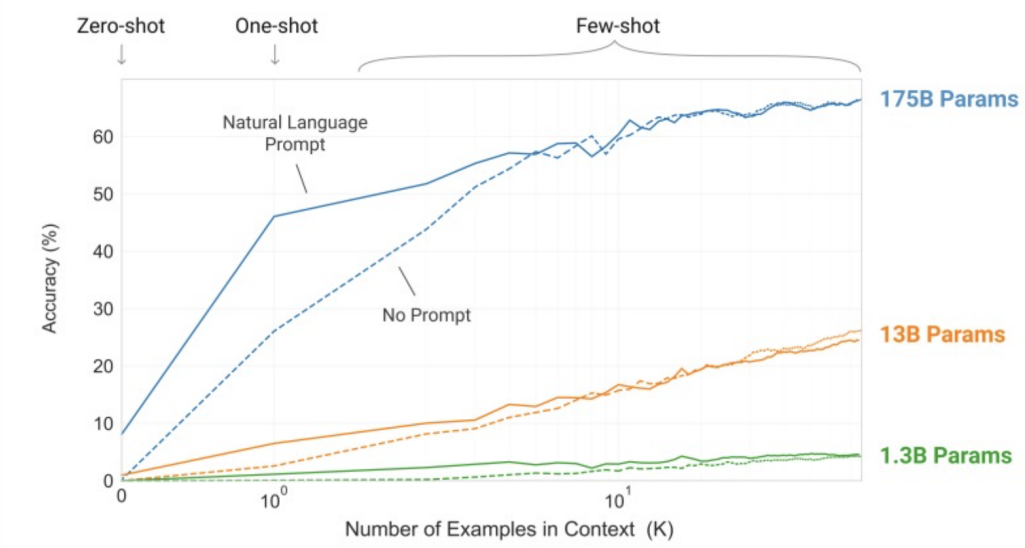

Рис. 3.8: Сравнение трех видов настроек контекстного обучения GPT-3.

На рисунке 3.8 показано сравнение применяемых методов для задачи удаления лишних символов из слова. Производительность модели улучшается с добавлением описания задачи на естественном языке, а также с увеличением количества примеров. Результаты обучения методом Few-Shot также значительно улучшаются с увеличением размера модели.

3.2.4 Архитектура

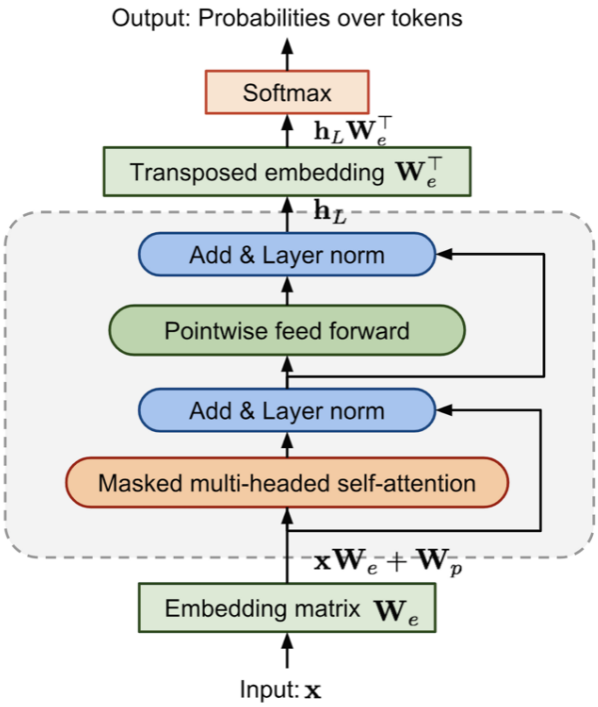

Рис. 3.9: Архитектура декодера блока трасформера

Последовательность шагов при генерации продолжения текста с помощью GPT такова (рисунок 3.9):

1 Токенизация входного текста.

2 Токены проходят через линейный слой и превращаются в набор эмбеддингов (по аналогии с word2vec).

3 Каждый эмбеддинг складывается с positional embedding.

4 Набор получившихся эмбеддингов проходит через несколько идентичных декодеров.

5 По выходу эмбеддингов из всех декодеров, эмбеддинг, который соответствует последнему токену, матрично умножается на входной транспонированный линейный слой. Применив SoftMax, мы получим распределение вероятностей для следующего токена.

6 Из получившегося распределения выбираем следующий токен (один из тривиальных подходов – использовать argmax).

7 Выбранный токен добавим к входному тексту и повторим алгоритм.

3.2.5 ruGPT-3. Примеры использования

В октябре 2020-го года на основе статьи от OpenAI [23] и кода модели GPT-2 была разработана русскоязычная версия GPT под названием ruGPT-3 в различных вариациях по количеству признаков: от 125 млн. до 13 млрд.

За счет универсальности и гибкости модели ее можно использовать не только для создания текста, но и в десятках других сложных сценариев, например:

- Анализ настроений текста, их классификация на положительные и отрицательные,
- Детоксификация тектов,
- Ранжирование картинок и подписей к ним (относим подпись к нужному изображению),
- Симплификация / суммаризация / резюмирование текстов,
- Сервисы по генерации тексты (актуально для копирайтинга),
- Переводы,
- Чат-боты,
- Генерация программного кода.

3.3 Методы генерации текста

Языковая модель возвращает распределение вероятностей следующего токена. Эти данные можно по-разному использовать для генерации текста.

1 **Greedy Search**. Способ заключается в выборе токена с максимальной вероятностью (аргмаксный подход). Метод далеко не самый лучший, так как генерация застревает в локальных минимумах и часто выдаёт повторяющиеся фрагменты.

2 **Beam search**. Чуть более сложный и качественный способ сэмплирования. Здесь мы на каждом шаге выбираем не один токен, а сразу несколько, и дальше продолжаем поиск для каждого из выбранных токенов. Таким образом мы получаем различные вариантов сгенерированного текста. Выбор в пользу того или иного варианта происходит на основе условной вероятности. Такой алгоритм позволяет выбрать не одного лучшего кандидата, а сразу N, такой параметр есть у функции generate. Генерация обладает хорошей когерентностью, но обычно вариантам не хватает “человечности”. При

это проблема повторяющихся фрагментов также остается. Пример дерева представлен на рисунке 3.10.

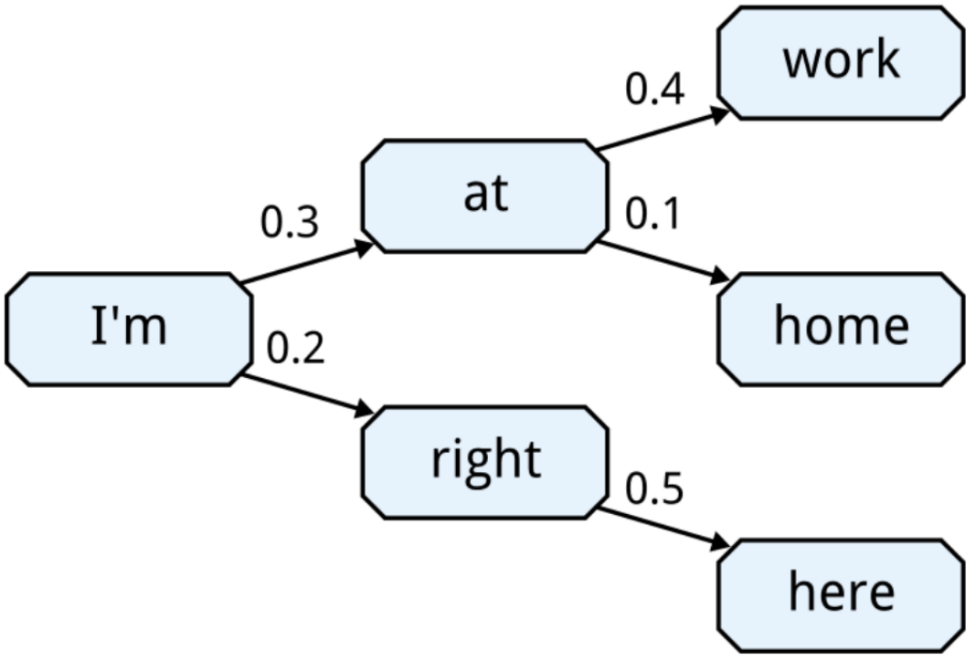

Рис. 3.10: Пример дерева beam-search. Числа — это вероятности токенов.

3 **Temperature sampling**. В рамках этого метода токен берется случайно с учетом получившегося распределения вероятностей. Параметр температуры отвечает за степень случайности метода. При нулевой температуре мы получаем greedy search. Чем выше температура, тем выше степень случайности. Как правило, хорошие результаты достигаются при температуре от 0.8 до 2. У этого метода есть и минусы, ведь случайность генерации будет иногда приводить к совсем некорректным результатам, но не очень часто.

$$p' = softmax(\frac{\ln(p)}{t}) \tag{1}$$

4 **Nucleus Sampling**. Метод ограничивает сэмплирование совсем некорректных токенов, отсекая их по вероятностям. Происходит либо зануление всех токенов, кроме самых больших, либо зануление всех, кроме тех, сумма вероятностей которых была бы не больше p.

4 Экспериментальное исследование

4.1 Подготовка датасета

В ходе работы были собраны данные из пяти источников:

- Трейн соревнования SpellRuEval (данные из социальных медиа),
- Новостной портал Lenta.ru,
- Русские субтитры,
- Русская литература,
- Сайт Gazeta.ru.

Причина выбора данных источников – отсутствие орфографических ошибок в данных, что позволяет использовать их в качестве таргета при создании будущего датасета. Наличие различных источников делает модель более гибкой и способной распознавать ошибки в разных видах документов. Недостатком таких данных является сильная несбалансированность по длине предложений (Рисунок 4.1). Чаще всего встречаются предложения длиной не превышающей 70 символов.

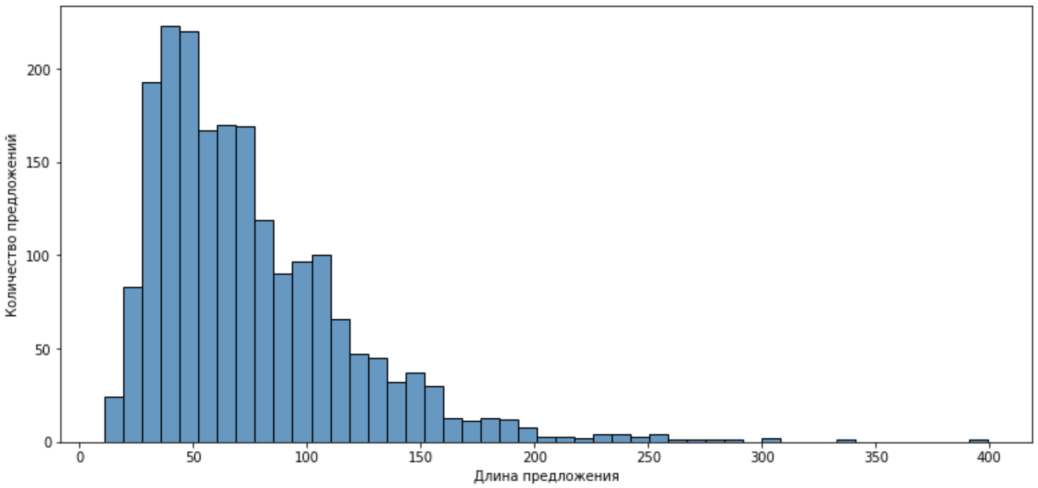

Рис. 4.1: Распределение длин слов в исходном датасете

После проведенного препроцессинга (в том числе установления порога по длинам предложений в процессе сбора данных) удалось сделать итоговый датасет более сбалансированным (Рисунок 4.2). Удалены предложения, содержащие недопустимые символы. Сделана фильтрация по длине.

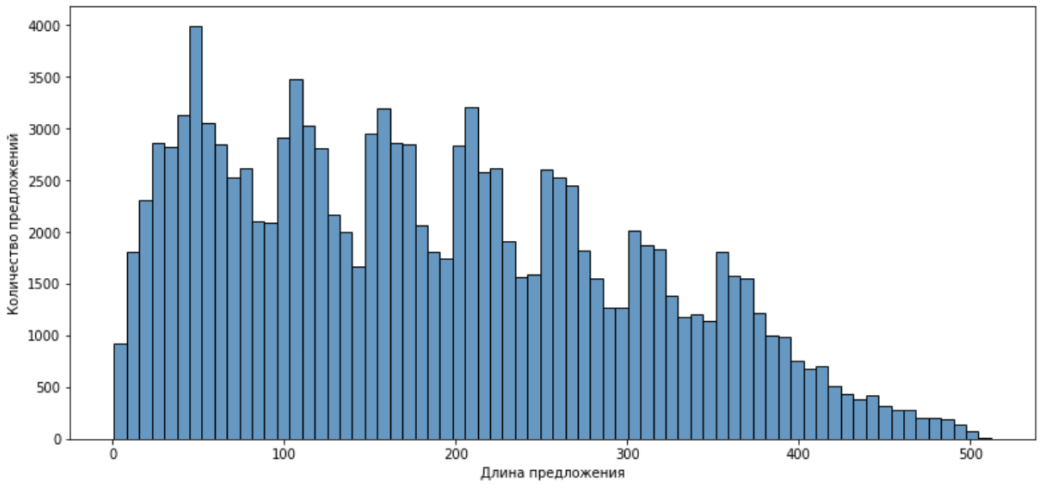

Рис. 4.2: Распределение длин слов после препроцессинга

Для данных SpellRuEval [5] предложения с опечатками уже были, так, следующей задачей была генерация предложений с ошибками для остальных данных. Для этого была реализована функция, добавляющая в исходный текст опечатки. Очевидно, что лучше всего для этой цели подошли бы часто встречающиеся в жизни ошибки людей, сгенерированные опечатки могут быть достаточно редкими на практике (например, когда клавиши букв находятся далеко друг от друга, непреднамеренно такую ошибку совершить очень сложно). Для генерации подобных реалистичных ошибок был использован датасет от КартаСлов [24], содержащий информацию о самых встречаемых опечатках в словах и их весе в зависимости от частоты их встречаемости. Таким образом, если слово есть в словаре КартаСлов и необходимо добавить в него опечатку, ошибка генерировалась случайно с учетом вероятностного распределения ее веса для данного слова, такая опечатка является естественной. Для тех слов, которые в данном словаре отсутствуют, генерировались синтетически следующие виды ошибок (стандартные перестановки по Левенштейну):

- вставка лишнего символа,
- пропуск символа,
- замена символа,
- неверный порядок символов.

Код данной функции показан на рисунке 4.3, параметр error-rate – степень зашумленности датасета, регулирующий вероятность совершения ошибки в конткретном слове. Наличие данного параметра позволило делать эксперименты на датасетах разной степени зашумления.

```python
def add_spelling_errors(token, error_rate=0.22):
    if (validate(token)): # английские опечатки не исправляем
        return token

    assert(0.0 <= error_rate < 1.0)
    if len(token) < 2:
        return token
    rand = np.random.rand()
    if(rand>error_rate):
        return token
    prob = error_rate / 4.0
    if token in err.index:
        return make_error(token) # сделать натуральную ошибку из КартаСЛОВ
    elif rand < prob: # вставка лишнего символа
        random_char_index = np.random.randint(len(token))
        token = token[:random_char_index] + np.random.choice(chars) \
                + token[random_char_index + 1:]
    elif prob < rand < prob * 2: # пропуск символа

        random_char_index = np.random.randint(len(token))
        token = token[:random_char_index] + token[random_char_index + 1:]
    elif prob * 2 < rand < prob * 3: #замена символа
        random_char_index = np.random.randint(len(token))
        token = token[:random_char_index] + np.random.choice(chars) \
                + token[random_char_index:]
    elif prob * 3 < rand < prob * 4: # буквы поменяны местами
        random_char_index = np.random.randint(len(token) - 1)
        token = token[:random_char_index]  + token[random_char_index + 1] \
                + token[random_char_index] + token[random_char_index + 2:]

    return token
```

Рис. 4.3: Код функции генерации ошибок

В итоге получены синтетический датасет размером порядка 117 тысяч предложений, который можно использовать при обучении и валидации различных моделей для решения задачи спеллчекинга.

В качестве тестового датасета использовались тестовые данные с соревнования по спеллчекингу SpellRuEval [5], содержащие 2008 предложений. Данные собирались из социальных медиа и были отредактированы вручную.

4.2 Существующие решения

Особенность задачи спеллчекинга состоит в том, что ее можно решать по-разному, но готового решения, позволяющего идеально исправлять все синтаксические, лексические и семантические ошибки в тексте не существует. Рассмотрим некоторые из них. Результаты метрик (precision, f-score, recall, accuracy) на датасете SpellRuEval представлены в Таблице 4.1.

Таблица 4.1: Сравнительный анализ разных подходов к решению задачи спеллчекинга.

Correction method	Precision	Recall	F – measare	Speed(sentence/s)
Yandex.Speller	83.09	59.86	69.59	5.0
Damerau Levenstein 1 + lm	59.38	53.44	56.25	39.3
Brill Moore + lm	51.92	53.94	52.91	0.6
Hunspell + lm	41.03	48.89	44.61	2.1
JamSpell	44.57	35.69	39.64	136.2
Brill Moore top 1	41.29	37.26	39.17	2.4
Hunspell	30.30	34.02	32.06	20.3

Один из самых точных результатов показал Yandex.Speller. Этот инструмент использует библиотеку машинного обучения CatBoost. Благодаря CatBoost он может расшифровывать искажённые до неузнаваемости слова («адникасники» → «одноклассники») и учитывать контекст при поиске опечаток («скучать музыку» → «скачать музыку»). Кроме того, Спеллер не придирается к новым словам, ещё не попавшим в словари.

От DeepPavlov представлены две модели для исправления опечаток на русском языке - Damerau Levenstein, которая с помощью расстояния Левенштейна ищет подходящих кандидатов-слов для исправления ошибок, и модель Brill Moore, использующая для этих целей статистическую модель ошибок. В обоих решениях кандидаты для замены опечаток подбираются с учетом контекста, для этого применяется языковая модель KenLM [25].

Hunspell - это спеллчекер, который используется в компаниях LibreOffice, OpenOffice.org, Mozilla Firefox & Thunderbird, Google Chrome. Инструмент предназначен для языков со сложной морфологией, основывается на использовании словарей.

4.3 Реализованные решения

4.3.1 ruT5

При дообучении модели ruT5-base для решения задачи спеллчекинга были протестированы многочисленные конфигурации, такие как скорость обучения, число эпох, размер батча, степень зашумленности, префикс, стратегии декодинга. Эмпирическим путем удалось подобрать наиболее оптимальные параметры. Промежуточные результаты рассчитаны на неполном датасете, так как запуск одной эпохи на полном датасете требует значительных ресурсов, поэтому метрики отличаются от итоговых.

Таким образом, в результате расчетов метрик, приведенных в таблице 4.2, наилучшей конфигурацией для обучения оказалось использование префикса “Исправь :” в сочетании с

Таблица 4.2: Метрики на различных конфигурациях ruT5-base.

Prefix value	Precision	Recall	F – measure
Исправь :	42.41	41.02	41.02
Корректное написание :	37.38	33.44	36.25
Перепиши без ошибок:	35.92	34.94	34.91
Error rate %	Precision	Recall	F – measure
22%	42.41	41.02	41.02
30%	37.38	33.44	36.25
10%	18.20	17.82	14.71

датасетом зашумленности 22% (процент сгенерированных ошибок, error rate в функции 4.3). Кроме того, в ходе работы пришлось отказаться от коррекции предложений длины 500+ символов (в SpellRuEval максимальная длина предложения - 400 символов), так как модель на них показывала худшие результаты, вероятно, для обучения такой модели необходимо больше данных и вычислительных ресурсов. По итогу экспериментов лучшие результаты показали две модели - одна, обученная на исходном датасет с error rate 22% 2 эпохи, другая модель - обученная последовательно на датасете с error rate 30 % (сокращенном) и одну эпоху на 22%-ом датасете. На рисунке 4.4 представлен график целевой функции. Кроме того, на датасете с error rate 22% дообучена модель ruT5-large, она продемонтрировала лучший результат по сравнению с ruT5-base на том же датасете.

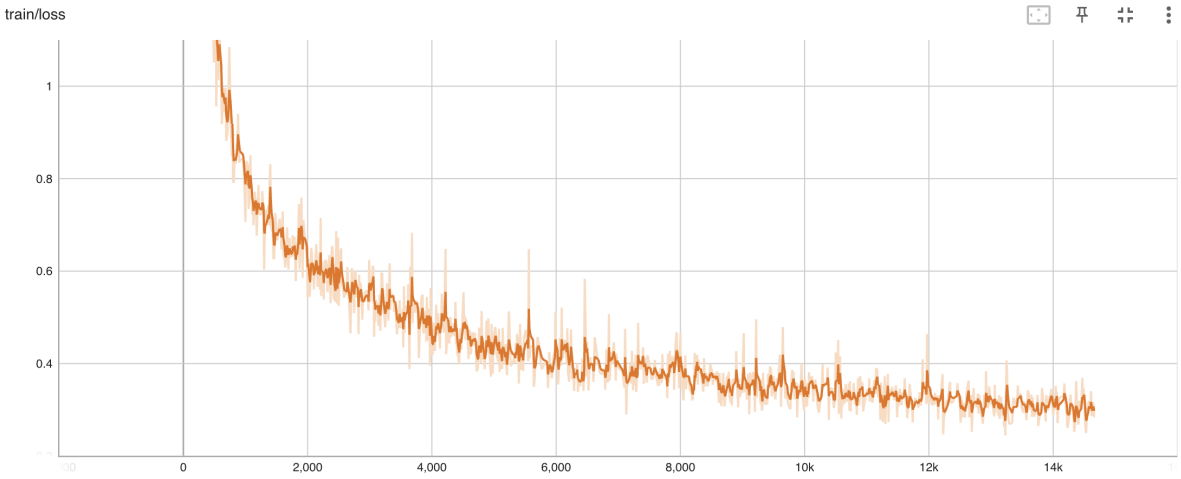

Рис. 4.4: График loss ruT5-base

4.3.2 ruGPT3

Для обучения ruGPT3 использовался тот же датасет с 22% шума. Для формирования входной последовательности применялась следующая схема:

<s>слово с ошибкой ==> исправленное слово</s> (2)

На вход сети подается часть до разделяющего токена “==>” включительно. Выход модели имеет вид, указанный в формуле (2). График целевой функции ruGPT3 представлен на рисунке 4.5. Результаты в целом получились хуже, чем у T5, часть ошибок модель исправила, однако, иногда в конце длинных предложений она генерировала случайный текст, отличающийся от входного. Возможно, такое поведение связано с тем, что модели необходимо больше данных, или проблема заключается в отсутствии энкодерной части у ruGPT3.

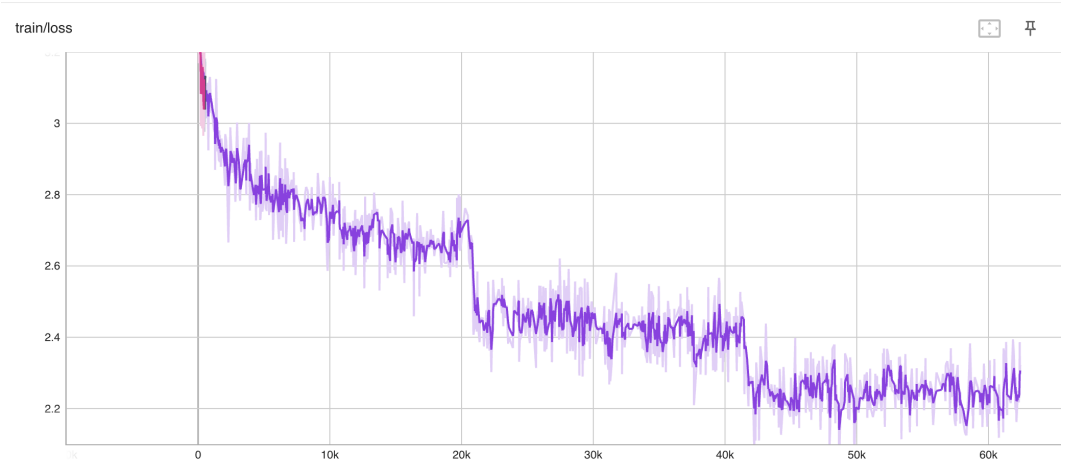

Рис. 4.5: График loss RuGPT3.

4.3.3 SymSpell

В качестве эксперимента реализован смешанный алгоритм, использующий SymSpell [26] и ruT5-base. Суть алгоритма состоит в том, что на вход программе подается датасет слов, каждое из которых он запоминает. Далее, когда на вход поступает ошибочное предложение, те слова, которые алгоритм не встречал, он считает ошибочными и начинает их исправление. Для этого к ошибочным словам применяем все возможные стандартные преобразования по Левенштейну, таким образом генерируются кандидаты для исправления, и те, которые известны алгоритму, считаются валидными. Затем валидных кандидатов сортируют по частоте встречаемости в датасете, добавляют в исходное предложение и эти итоговые предложения оценивает обученная ruT5-base на предмет семантической адекватности. По итогу выбирается кандидат с наилучшим значением метрики.

Алгоритм показал не слишком хороший результат на практике, так как в нем часто в качестве замены предлагались более частотные слова, ruT5-base же не всегда корректно выдавала результат семантической адекватности. Еще один недостаток данного подхода - долгое время работы.

4.3.4 Итоговое сравнение результатов

Таблица 4.3: Итоговое сравнение метрик.

Correction method	Precision	Recall	F − measure	Accuracy
1 место соревнования	81.98	69.25	75.07	70.32
2 место соревнования	67.54	62.3	64.82	61.35
3 место соревнования	71.92	52.31	60.59	58.42
ruT5-base 2 датасета	80.13	67.09	73.03	69.53
ruT5-Large	71.38	58.82	64.50	60.75
ruT5-base	64.38	48.12	55.08	54.8
ruGPT3	49.58	21.08	29.58	41.8
SymSpell+ruT5-base	28.24	19.64	23.17	30.5

В таблице 4.3 представлены метрики обученных в рамках данной работы моделей в сравнении с лучшими решениями соревнования SpellRuEval (первые три строки). Возглавила рейтинг модель, базирующиеся на языковых триграммных моделях с использованием расстояния Левенштейна, что является одним из классических подходов к решению задачи спеллчекинга. Модель ruT5-base, которая обучалась на двух датасетах с разной степенью зашумленности, имеет схожие метрики, незначительно уступает по каждой из них. Дообученная ruT5-Large, обученная на одном датасете демонстрирует неплохое качество, но уступает прошлой. RuT5-base показала результат хуже, чем ruT5-Large, в силу меньшего числа параметров. RuGPT3 проигрывает по метрикам всем предыдущим моделям, возможно, это связано с более простой архитектурой в сравнении с ruT5-base. SymSpell в предложенном варианте оказался неприменим в рамках поставленной задачи.

5 Выводы и результаты

В ходе проделанной работы сформирован синтетический датасет для обучения и сравнения моделей для задачи исправления ошибок на русском языке, обучены авторегрессионные модели на базе ruGPT3, ruT5-base и ruT5-Large, опробован гибридный подход (ruT5-base+SymSpell). В процессе исследования протестированы различные параметры датасета, моделей и выявлены их наиболее удачные конфигурации. Сравнивая качество работы предобученной модели ruT5-base с результатами работы топ-3 моделей соревнования SpellRuEval, каждая из которых использовала лингвистический словарь, n-граммную языковую модель и расстояние редактирования в том или ином виде, можно сказать, что авторегрессионные

модели могут быть использованы для решения задач спеллчекинга, ведь, они показывают достойные результаты. Сравнительный анализ показал, что на полученном синтетическом датасете модели ruT5-base и ruT5-Large, имеющие энкодер-декодерную архитектуру показывают лучший результат, по сравнению с декодерными ruGPT3. Результаты метрик не идеальны, но поле для дальнейших исследований широко: исследование других моделей энкодер-декордерной архитектуры (например, BART), проведение анализа ошибок, распределения перплексии по токенам, чтобы выяснить, какие конкретно ошибки модель плохо распознает и сделать соответствующую корректировку датасета и параметров моделей.

Список литературы

[1] Vladimir I. Levenshtein. “Binary codes capable of correcting deletions, insertions, and reversals”. В: *Soviet Physics Doklady* (1966).

[2] Fred J. Damerau. “A technique for computer detection and correction of spelling errors”. В: *Communications of the ACM* (1964).

[3] K. W. Church Kernighan и W. A. Gale. “A spelling correction program base on a noisy channel model.” В: *COLING* (1990).

[4] A. Golding и D. Roth. “A Winnow-Based Approach to Context-Sensitive Spelling Correction”. В: *Machine Learning 34* (1999), с. 107—130.

[5] *Датасет для исправления опечаток с соревнования Dialogue Evaluation,* 2016. URL: https://www.dialog-21.ru/evaluation/2016/spelling_correction/ (дата обр. 01.09.2022).

[6] A. Gillioz и др. “Overview of the Transformer-based Models for NLP Tasks”. В: *researchGate:0.154* (2020).

[7] Kernighan, Mark D. и Kenneth W. Church and William A. Gale. “A spelling correction program based on a noisy channel model.” В: *Proceedings of COLING 1990* (1990).

[8] Mays E. and Fred J. Damerau and Robert L. Mercer. “Context based spelling correction. Information Processing and Management.” В: *Information Processing Management* (1991).

[9] Gupta P. “A context-sensitive real-time Spell Checker with language adaptability.” В: *IEEE 14th International Conference on Semantic Computing (ICSC)* (2020).

[10] Yifei Hu и др. “Misspelling Correction with Pre-trained Contextual Language Model.” В: *arXiv:2101.03204v1 [cs.CL] 8 Jan 2021* (2021).

[11] Dereza O. V. и др. “A Complex Approach to Spellchecking and Autocorrection for Russian”. В: *Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference Dialogue* (2016).

[12] Eric Brill и Robert C. Moore. “An Improved Error Model for Noisy Channel Spelling Correction”. В: *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics* (2000), с. 286—293.

[13] Sorokin A. A. и Shavrina T. O. “Automatic spelling correction for Russian social media texts”. В: *Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference Dialogue 2016* (2016).

[14] Panina M. F., Baytin A. V. и Galinskaya I. E. “Context-independent autocorrection of query spelling errors”. В: *Computational Linguistics and Intellectual Technologies: Proceedings of the International Conference Dialogue 2016* (2016).

[15] Colin Raffel и др. “Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”. В: *Journal of Machine Learning Research 21* (2020), с. 1—67.

[16] Linting Xue и др. “mT5: A massively multilingual pre-trained text-to-text transformer”. В: *CoRR* abs/2010.11934 (2020). arXiv: 2010.11934. URL: https://arxiv.org/abs/2010.11934.

[17] *ruT5, ruRoBERTa, ruBERT: как мы обучили серию моделей для русского языка*. 2021. URL: https://habr.com/ru/company/sberbank/blog/567776/ (дата обр. 01.10.2022).

[18] Ashish Vaswani и др. “Attention Is All You Need”. В: *arXiv preprint, arXiv:1706.03762, version 5* (2017).

[19] Bahdanau D., KyungHyun Cho и Yoshua Bengio. “NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE”. В: *ICLR 2015* (2016).

[20] Radford. A и др. “Improving Language Understanding by Generative Pre-Training.” В: *openAI* (2018).

[21] Radford A. и др. “Language Models are Unsupervised Multitask Learners.” В: *openAI* (2019).

[22] Mark Chen и др. “Evaluating Large Language Models Trained on Code.” В: *arXiv preprint, arXiv:2107.03374v2* (2021).

[23] Tom B. Brown и др. “Language Models are Few-Shot Learners”. В: *NeurIPS Proceedings, 33* (2020).

[24] *Датасет: орфографические ошибки и опечатки*. URL: https://github.com/dkulagin/kartaslov/tree/master/dataset/orfo_and_typos (дата обр. 01.09.2022).

[25] Kenneth Heafield. “KenLM: Faster and Smaller Language Model Queries”. В: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)* (2011).

[26] *A quick overview of the implementation of a fast spelling correction algorithm*. URL: https://medium.com/@agusnavce/a-quick-overview-of-the-implementation-of-a-fast-spelling-correction-algorithm-39a483a81ddc (дата обр. 01.10.2022).