Санкт-Петербургский государственный университет

Мельников Олег Кириллович

Выпускная квалификационная работа

Серверная модель исправления опечаток

Уровень образования: бакалавриат
Направление: 01.03.01 “Математика”
Основная образовательная программа: СВ.5000.2018 “Математика”

Научный руководитель:
Авдюшенко А.Ю., доцент, Факультет математики и компьютерных наук СПбГУ

Рецензент:
Артемова Е.Л., доцент, Факультет компьютерных наук НИУ ВШЭ

Санкт-Петербург
2022

Содержание

1. Введение

Многие современные программные продукты в которых пользователь работает с текстом оснащены системами исправления опечаток. По мере того как пользователь набирает текст, система в режиме реального времени делает исправления, а иногда указывает на потенциальную ошибку и подсказывает возможный вариант верного написания слова. От качества системы исправления опечаток может зависеть удобство печатания для пользователя, а также различные функции конкретного программного продукта, например, качество поиска, если рассматривается текст вводимый в поисковую строку.

Качественная система исправления опечаток должна брать во внимание контекст, в котором употребляется то или иное слово, так как в зависимости от контекста корректными могут быть разные варианты написания. Кроме того, окружающий контекст может значительно помочь в детекции опечатки и поиске верного варианта исправления.

Большинство современных исследований систем исправления опечаток использует техники обработки естественного языка, главным образом применяющие нейронные сети [4] [6] [14] [5] [8], в то время как более классические подходы [1] [13] [11] основаны на редакционном расстоянии и словарях.

При разработке системы исправления опечаток важной характеристикой помимо качества исправлений является также скорость работы, потому как если не брать в расчет скорость, можно будет попробовать в качестве исправления абсолютно любое слово из словаря, и оценить какое лучше. Такой подход не применим на практике.

Существующие нейросетевые архитектуры содержащие в себе авторегрессионную часть, такие как трансформер с кодировщиком и декодировщиком, имеют потенциал справляться с исправлением опечаток лучше, чем модели без этой части, так как исправление опечаток содержит в себе этап генерации верного написания, который, в свою очередь, напрямую вписывается в авторегрессионную архитектуру.

В компании JetBrains существует система исправления опечаток, поставляемая

вместе со всеми IDE, действующая на устройстве пользователя. Система применяется для исправления опечаток в комментариях, а также при работе с Markdown и другими текстовыми файлами. В силу ограничений по памяти, на системы которые можно поставлять в комплекте IDE, данная модель имеет недостаточно ресурсов, чтобы применять нейросетевые или иные подходы использующие контекст для исправления опечаток. С другой стороны, если развертывать модель на удаленном сервере, то ограничений по памяти будет на порядок меньше, а это, в свою очередь, позволит построить более сильную модель, которая может использовать тяжеловесные нейронные сети и качественно учитывать контекст.

Цель работы:

Создание новых методов исправления опечаток, которые превзошли бы по качеству существующие подходы. Предложенные методы должны учесть недостатки существующих методов, объединив в себе нейронные сети с авторегрессионной частью для использования информации про контекст, а также способы ускорения работы нейронных сетей, чтобы модель могла использоваться в режиме реального времени в IDE JetBrains.

Задачи работы:

- Анализ существующих методов исправления опечаток
- Исследование нейросетевых моделей исправления опечаток
- Ускорение модели для работы в режиме реального времени
- Оценка качества предложенных методов на существующих датасетах и сравнение с существующими моделями

2. Обзор литературы

2.1. Классические подходы

Существует ряд стандартных открытых моделей исправления опечаток, таких как Hunspell [11], GNU Aspell [1] и Jamspell [13]. Эти модели действуют по общей схеме:

- Детекция слов с опечатками путем стемминга и анализа основы, а также анализа префиксных и суффиксных частиц. При этом для анализа используются заранее собранные словари.
- Генерация вариантов верного написания. Генерация осуществляется небольшим числом изменений символов исходного написания (замена, удаление, вставка, перестановка), затем осуществляется проверка наличия полученного слова в словаре.
- Ранжирование вариантов верного написания и выбор оптимального. Варианты ранжируются на основе признаков, например, на основе редакционного расстояния.

Важной особенностью этих моделей является то, что они не используют контекст для более точного исправления опечаток. Например, они не справляются с тем, чтобы разрешить неоднозначность и исправить thaught в *taught* или *thought* в зависимости от контекста: “Who thaught you calculus?” и “I never thaught I would be awarded the fellowship.”

Кроме того, эти модели в разных формах используют грубые допущения относительно верного варианта написания, например, в качестве способа ранжирования в одной из моделей выбирается вариант с наименьшим расстоянием по Левенштейну, а в случае равенства расстояний по Левенштейну для нескольких вариантов, в качестве ответа выбирается лексикографически минимальный вариант. Данный подход является удобной в использовании эвристикой, а не осмысленным способом дифференцировать варианты.

Другим важным аспектом подобных моделей является то, что этап сверки со словарем гарантирует, что модель не предложит ничего нецензурного. Такое свойство крайне важно для систем, используемых в программных продуктах.

2.2. Подход, применявшийся в JetBrains

В качестве модели исправления опечаток в IDE JetBrains использовалась модернизированная версия классических подходов, с применением дополнительных признаков: признаков отвечающих за созвучие и фонетику, сходство Джаро-Винклера и других.

Данный подход с одной стороны сохраняет высокую скорость работы и минимальный расход памяти, так как не использует тяжелые контекстуальные признаки, с другой стороны он максимально использует неконтекстуальную информацию о слове, что повышает его качество относительно более прямолинейных подходов.

2.3. Современные подходы, использующие контекст и глубокое обучение

Более продвинутые модели прибегают к использованию нейронных сетей для обработки последовательностей, что позволяет собирать контекстуальную информацию на основе всего предложения и применять ее для исправления опечаток.

Существуют открыто доступные, берущие во внимание контекст, модели. Например, набор из четырех моделей от NeuSpell [4]. Описание некоторых из них:

- CHAR-LSTM-LSTM: Модель строит векторные представления слов передавая отдельные символы в двунаправленную LSTM. Эти представления затем подаются на вход в другую двунаправленную LSTM, натренированную предсказывать итоговое исправление.

- BERT: Модель использует предварительно натренированную нейронную сеть архитектуры Трансформер-кодировщик для формирования векторных представлений слов, через усреднение эмбеддингов токенов входящих в слово.

Затем полученные представления передаются в классификатор для предсказания исправления.

- SC-LSTM: Модель исправляет слова используя представления непрерывных отрезков из нескольких букв, из которых состоит слово, передавая их в двунаправленную LSTM сеть. Представления для отрезков из нескольких букв строятся как one-hot эмбеддинги для префикса, суффикса и внутреннего промежутков слова.

Другим примером нейросетевых моделей исправления опечаток, использующих контекст, описание которых выложено в общий доступ, является набор решений в [6]. Описание нескольких из них:

- Word+Char Encoder: Модель использует два Трансформера-кодировщика BERT для построения эмбеддингов для слова и для каждого символа, затем конкатенирует все в один вектор и предсказывает верное слово используя линейный слой.

- Subword Encoder: Модель использует Трансформер-кодировщик BERT для построения эмбеддингов токенов и по каждому предсказывает верное написание.

Таким образом, современные методы, основнанные на нейронных сетях, действуют по общей схеме: строятся векторные представления для токенов или слов, затем на основе них, скорее всего, при помощи линейного слоя, предсказывается верное написание. С другой стороны, существуют авторегрессионные модели, которые, например, с помощью лучевого поиска, генерируют слова потокенно один за другим. Эта схема выглядит весьма естественно в качестве способа генерировать верное написание слова.

Авторегрессионый подход генерации верного написания был продемонстрирован в двух работах корпораций: Microsoft [7] и HubSpot [5]. В обоих случаях постановка была несколько специфичной - исправлять опечатки было необходимо непосредственно в поисковой строке. Специфика в том, что обычно поисковой запрос - это всего пара слов, и не факт что грамматически и синтаксически верно

написанных, в отличии от реальных текстов и предложений, для которых применяется модель в JetBrains. В обоих работах применялся Char-level Трансформер класса кодировщик-декодировщик. Однако, все датасеты, детали реализации и непосредственно веса моделей не были выложены в общий доступ, так как обучение производилось на больших объемах частных данных пользователей.

Одной из популярных моделей, имеющих авторегрессионный блок является модель BART, представленная в работе [2]. Это модель Трансформер класса кодировщик-декодировщик, которая обучалась на задаче, имеющей много общего с задачей исправления опечаток. Формально, архитектуру сети BART можно представить следующим образом:

$$h_i = x_i + p_i, H = (h_1, ..., h_n) \tag{1}$$

$$h_i^j = \sum_{k=1}^{n} SoftMax((W_q^j h_i)^T (W_k^j H))(W_v^j H) \tag{2}$$

$$h'_i = [h_i^1 ... h_i^J] \tag{3}$$

$$h''_i = LN(h'_i + h_i; \mu_1, \sigma_1) \tag{4}$$

$$h'''_i = W_2 ReLU(W_1 h''_i + b_1) + b_2 \tag{5}$$

$$z_i = LN(h'''_i + h''_i; \mu_2, \sigma_2) \tag{6}$$

$x_i \in \mathbb{R}^d$ - эмбеддинг i-ого слова входного предложения,

$p_i \in \mathbb{R}^d$ - вектор, осуществляющий позиционное кодирование i-ого слова входного предложения,

$h_i \in \mathbb{R}^d$ - скрытое состояние р i-ого слова,

$W_q^j, W_k^j, W_v^j \in \mathbb{R}^{d \times d}$ - Query, Key, Value матрицы для j-ой головы самовнимания

LN - нормировка уровня

$ReLU$ - Rectified Linear Unit кусочно-линейная функция активации

$h'_i, h''_i, h'''_i \in \mathbb{R}^d$ - промежуточные скрытые состояния

$z_i \in \mathbb{R}^d$ - скрытое представление i-ого слова после первого блока энкодера

Шаги 2-6 повторяются 6 раз подряд - это схема работы кодировщика, затем идет авторегрессионная часть декодеровщика, в котором 6 аналогичных блоков, дополненные механизмом внимания на выходные эмбеддинги последнего слоя декодера. На выходе с части декодера применяется полносвязная нейронная сеть для генерации очередного токена выходного предложения.

Основное преимущество Трансформерных моделей, в том числе и модели BART - механизм внимания, который позволяет качественно учитывать смысловые, грамматические и синтаксические связи между разными словами предложения. Это свойство может оказать существенную пользу в задаче исправления опечаток.

BART генерирует новое предложение на основе данного, причем для обучения этой модели использовалась следующая постановка: рассматривалось корректное предложение без опечаток, часть токенов переставлялась, часть удалялась, часть маскировалась, и затем модель должна была восстановить исходное предложение без искажений.

Рис. 1: Схема искажения входных данных, используемая в процессе обучения модели BART.

3. Исследованные модели

В этом разделе будут описаны различные модели и подходы, которые были исследованы в рамках этой работы.

3.1. Архитектуры и постановки обучения

Один из прямолинейных подходов к задаче, который не был применен в open-source работах - это sequence-to-sequence исправление опечаток. Иными словами, модель сначала строит контекстуальные эмбеддинги для токенов предложения, а затем, в авторегрессионном режиме переписывает предложение с опечатками в исправленную версию без опечаток. Причем модель может работать с последовательностью на разных уровнях: на уровне символов и на уровне токенов. Такой подход позитивно отличается своей простотой от всех составных методов, состоящих из частей детекции опечаток, генерации вариантов и их ранжирования, поскольку в данном случае все сокращено до одного шага.

3.1.1. Char-level seq2seq: char-level Transformer обученный с нуля

Мотивация применения char-level модели состоит в том, что при наличии типичной небольшой опечатки, с точки зрения такой модели, основная часть слова остается неизменной, то есть большинство символов остается на своих местах и также корректно обрабатывается. В то время как модели работающие на уровне токенов или даже слов могут из-за одного неверного символа полностью поменять токенизацию слова, что затруднит механизм исправления.

Для обучения этой модели был взят не предобученный Трансформер класса кодировщик-декодировщик, который посимвольно принимает на вход исходное предложение, а затем должен посимвольно генерировать предожение без ошибок. Были поставлены эксперименты с разными вариантами архитектуры в терминах числа слоев и размера внутреннего состояния.

Важно отметить, что в сравнении с token-level моделями, длина последовательности при использовании char-level сразу возрастает в несколько раз. Это может

усложнить для модели процесс извлечения контекстуальной информации, так как рассматриваемых сущностей и тем более связей между ними, становится на порядок больше.

3.1.2. Token-level seq2seq: BART fine-tune

Мотивация для этой модели состояла в том, что хоть char-level и более устойчив к изменению одного символа, тем не менее, количество важной орфографической и синтаксической информации, содержащееся в token-level предобученной модели велико, и эта информация может дать значимый прирост в качестве исправлений.

Для обучения этой модели, в качестве инициализации весов нейронной сети были взяты веса предобученной модели BART, состоящей из 6 слоев кодировщика, 6 слоев декодировщика и с размером скрытого состояния равным 768, всего в модели было 140 миллионов параметров. Таким образом, учитывая, как предобучался BART, стартовые веса модели модели были сразу приспособлены к разным видам предложений с искажениями на входе.

3.1.3. Sep-Mask-One BART fine-tune

В процессе предобучения модели BART, одна из техник которая использовалась - маскирование некоторой доли токенов предложения и попытка восстановить их моделью. Естественной идеей применения этой техники к исправлению опечаток стала попытка заменить слово написанное с опечаткой на токен $<mask>$, и затем попробовать дать модели воссоздать исходное слово, но уже без ошибки.

Если взять такой подход без модификаций, то, так как модель не принимает на вход исходное написание слова, она будет лишена важной информации, способствующей верному исправлению. Чтобы модели было проще сгенерировать именно исправленное слово, а не какое-то другое слово, просто подходящее по контексту, была предпринята попытка передать в модель информацию об исходном написании слова. Для этого на вход модели шло слово с опечаткой, затем разделяющий токен $</s>$, затем само предложение с маскированным словом с ошибкой.

Для обучения этой модели, в качестве инициализации весов нейронной сети

также были взяты веса предобученной модели BART. Важно отметить, что на этапе предобучения данная модель уже обрабатывала примеры с <*mask*> и </*s*> токенами.

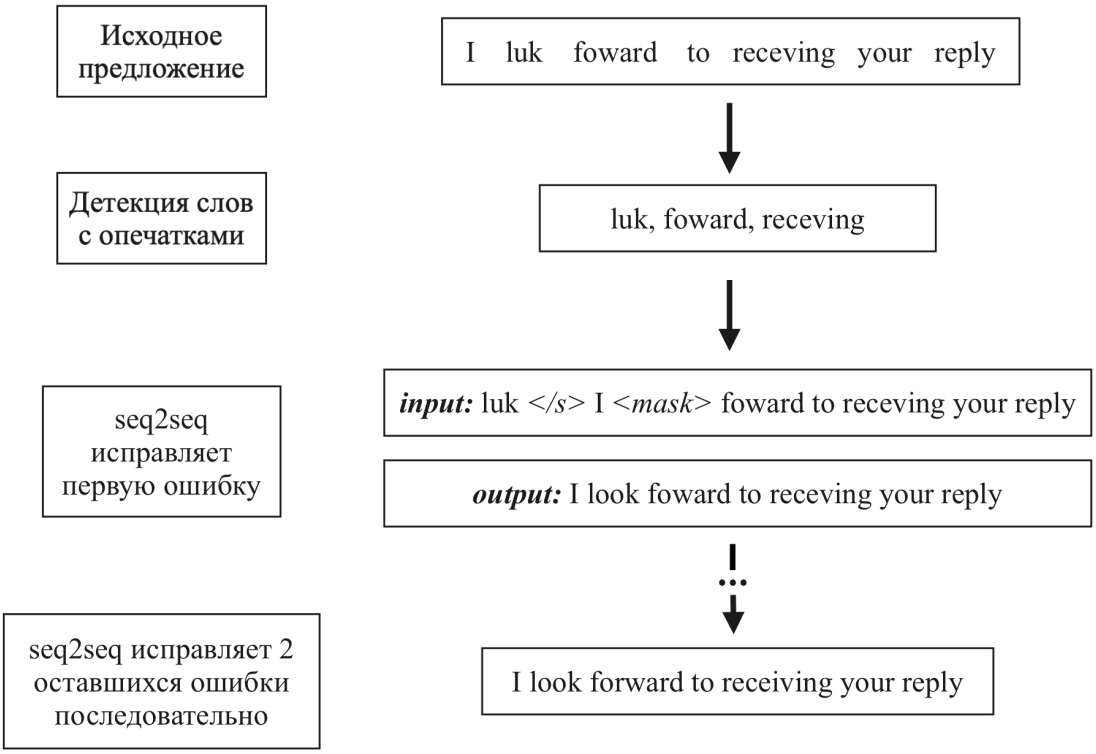

Рис. 2: Схема итеративного исправления ошибок моделью Sep-Mask-One

Заметим, что для использования такой модели необходима дополнительная модель для детекции (детектор) слов с опечатками. Однако требования по качеству предъявляемые к детектору будут уже на порядок ниже, поскольку если он, например, будет иметь ложные срабатывания, то seq2seq, который делает итоговое исправление, все равно будет вправе не исправлять и без того верное слово, а оставить его как есть. Таким образом вся внушительная контекстуальная сложность модели может заключаться именно в последней seq2seq части, а не в детекторе.

Более того, были поставлены эксперименты, сравнивающие классический детектор опечаток от Hunspell [11] и детектор опечаток на основе специально обученного тэггирующего Трансформера-кодировщика BERT, который решал задачу

бинарной классификации (класс 1: токен с ошибкой, класс 0: токен написан верно) для каждого токена. Из-за описанного выше эффекта, разница между моделями оказалась незначимой.

Другое важное свойство данной модели состоит в том, что в силу последовательных исправлений, она будет иметь накладные расходы по времени работы, в сравнении с моделями, которые исправляют все ошибки единовременно. Это автоматически делает модель менее пригодной для использования в режиме реального времени.

3.1.4. Sep-Mask-All BART fine-tune

Ориентируюсь на предыдущую модель, логичным продолжением гипотезы стала попытка маскировать не одно слово с ошибкой, а сразу все, и пытаться восстановить исправленные версии одновременно для всех слов.

Для исправления сразу всех ошибок есть 2 предпосылки:

- Зачастую предложения бывают с несколькими опечтками сразу, и если совершать исправление ошибок с помощью seq2seq модели по очереди, то получается, что когда мы учим модель исправлять первую ошибку, мы также учим модель не исправлять остальные, что противоестественно и противоречит предобучению модели. Поэтому, корректнее учить модель исправлять сразу все, не заучивая сохранение неверных написаний.

- Исправление всех ошибок одновременно делает модель более быстрой, потому что не требуется многократный запуск. Это является плюсом для использования в режиме реального времени.

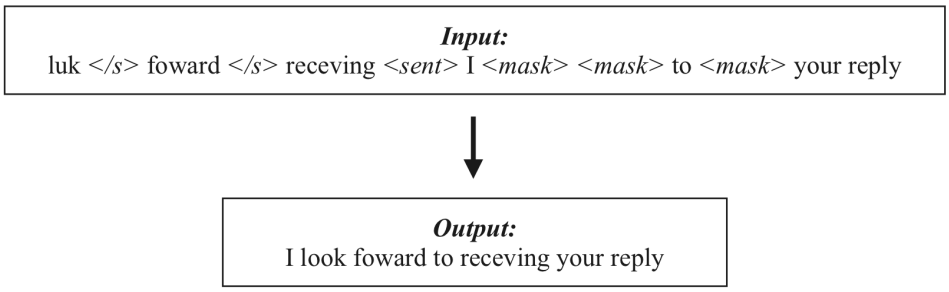

Рис. 3: Схема одновременного исправления всех ошибок моделью Sep-Mask-One

Для этой модели также необходим детектор с высокой полнотой нахождения ошибок, например, Hunspell [11].

3.1.5. Mask-Word BART fine-tune

Другой реализацией идеи использовать механизм маскирования заложенный схему предобучения модели BART, стала попытка авторегрессионно восстанавливать не все предложение целиком, а только искомое исправление слова с ошибкой. Действительно, можно не прогонять долго работающий авторегрессионый декодировщик на всем предложении, ограничившись одним словом, таким образом сократив его время работы кратно. Алгоритм работы модели выглядит так:

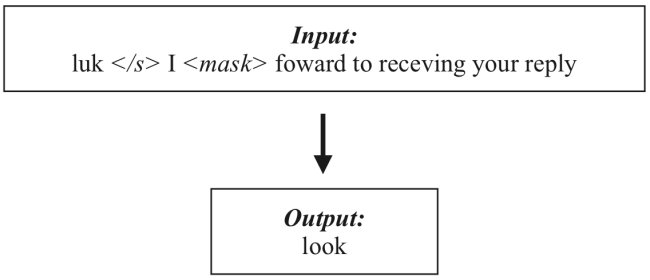

Рис. 4: Схема итеративного исправления ошибок моделью Mask-Word

Для этой модели также необходим детектор с высокой полнотой нахождения ошибок. Кроме того, эта модель также как и Sep-Mask-One должна исправлять ошибки последовательно, однако, данная модель, в отличии от *Sep-Mask-One* уже

не будет учиться при исправлении первой ошибки не исправлять остальные, выучивая таким образом неверное написание, так как остальные токены в принципе не генерируются.

3.1.6. Detection - Candidates Generation - Candidates Ranking

В начале работы над проектом, в IDE JetBrains использовалась трехступенчатая модель для исправления опечаток. Первая часть отвечает за детекцию слов с опечатками, вторая часть для каждого слова с опечаткой генерирует возможные варианты для исправления, а третья часть ранжирует кандидатов по качеству исправления и в конце концов выдает вариант, проранжированный выше всех. Было решено улучшать и использовать контекст именно в третьей части - в модели ранжирования вариантов.

Есть несколько причин, почему для промышленного использования правильнее делать отдельно модель ранжирования, а не все исправление end-to-end:

- Нейронная модель может сгенерировать что-то нецензурное. С другой стороны, в случае ранжирования прямой генерации нет - только оценка имеющихся, заранее проверенных на цензурность вариантов, что обеспечивает безопасность.

- В случае прямой end-2-end генерации верного написания предложения, для повышения качества зачастую применяется лучевой поиск, который увеличивает размер одного батча и тем самым увеличивает время на обработку одного запроса. Это может замедлить модель в несколько раз.

В качестве моделей ранжирования были рассмотрены:

- Вероятность сгенерировать данного кандитата в $<mask>$ постановке предобученным BART-ом. Иными словами, на вход seq2seq принимает предложение с замаскированным словом с опечаткой, а в декодеровщик подаются несколько версий написания исходного предложения, в которых слово с опечаткой заменено очередным вариантом. Таким образом вычисляются логиты для каждого токена слова-варианта, затем эти логиты складываются и получается

итоговая вероятностная оценка варианта - логарифм вероятности сгенерировать заданное слово на месте опечатки.

- Вероятность сгенерировать данного кандитата в Sep-Mask-All постановке BART-ом. Схема аналогична с первым вариантом, только в качестве seq2seq используется уже специально дообученный исправлять все ошибки BART из пункта 3.1.4.

В качестве моделей для детекции и генерации кадидатов использовался стандартный пакет Hunspell [11], применяемый в Google Chrome, LibreOffice и Mozilla Firefox. Эти модели справляются с тем, чтобы найти 99% ошибок (на рассмотренных датасетах), однако также имеют большой процент ложных срабатываний (>20%) и значимый процент случаев (>10%), в которых найти верного кандидата не удается.

Несмотря на перечисленные проблемы, для разработки и оценки качественной модели ранжирования, это не является существенным препятствием. Для обучения и тестирования модели ранжирования рассматривались отдельно только те случаи, где первые две модели успешно справились с детекцией и нахождением корректного варианта.

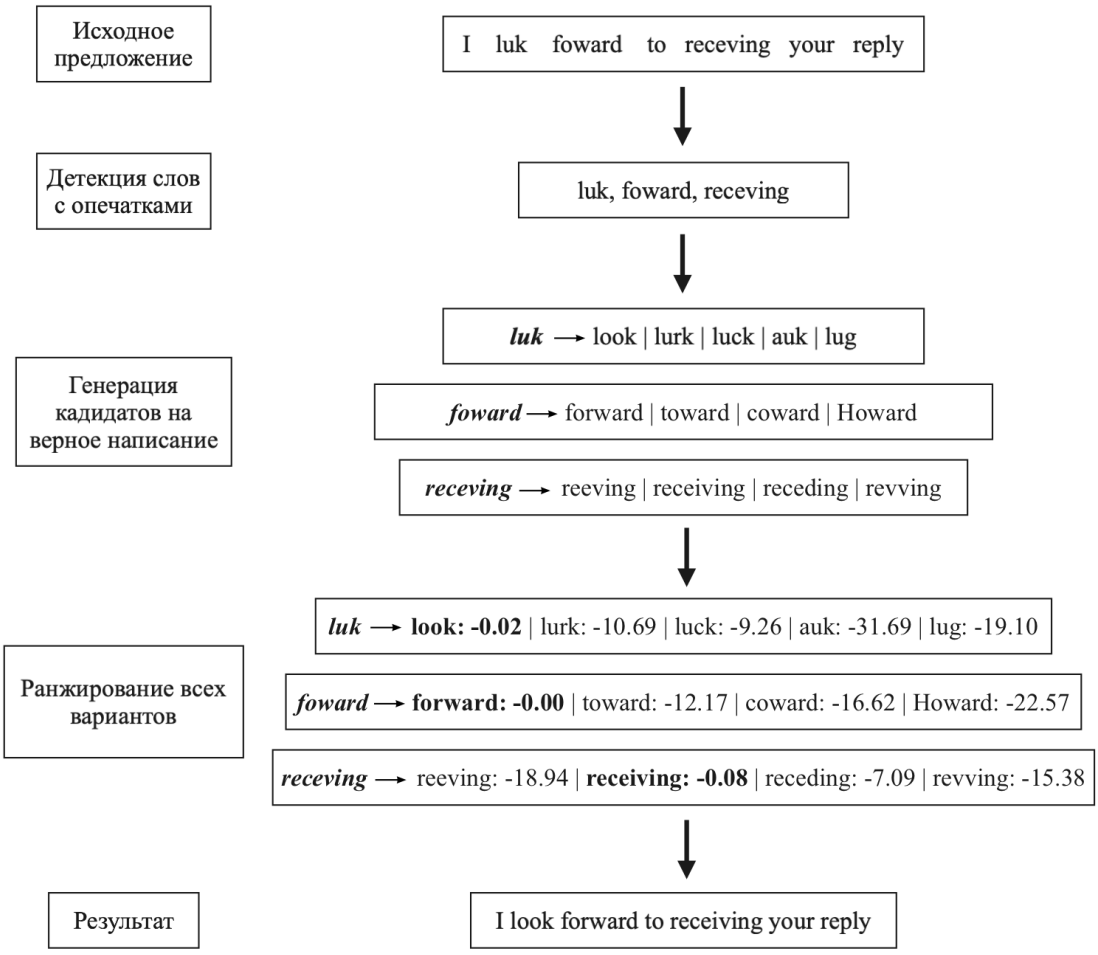

Рис. 5: Схема исправления опечток моделью типа Detection-Candidates Generation-Ranking. Отрицательные значения при ранжировании - реальные логиты, посчитанные моделью Sep-Mask-All.

3.2. Ускорение моделей

Для использования модели в режиме реального времени, необходимо, чтобы время обработки запроса было настолько незначимым, чтобы пользователь не замечал никакой задержки. На практике допустимое время зардержки - 200 миллисекунд. Внутри системы клиентское приложение посылает предложение на сервер, на сервере отрабатывает тяжелая нейросетевая модель, затем результат посыла-

ется обратно клиенту, а для пользователя исправление происходит мгновенно.

Классическим методом ускорения нейросетевых моделей является дистилляция. Существует несколько способов ее осуществлять, самый прямолинейный из них - удалить часть слоев модели, а затем дообучить полученную модель на исходном датасете.

При использовании Трансформера класса кодировщик-декодировщик основное время работы уходит именно на авторегрессионную генерацию, потому что этот процесс генерирует токен за токеном, без возможности генерировать все токены параллельно, по сравнению с тем, как это происходит в кодировщике, на этапе которого все токены известны сразу. Поэтому для ускорения seq2seq Трансформера было принято решение сократить число блоков декодера с 6 до 3, что обычно приводит к двукратному ускорению выполнения.

4. Эксперименты и результаты

В этом разделе представлены основные части постановки экспериментов: наборы данных, используемые в обучении и экспериментах, методология оценки и существующие методы, с которыми проведено сравнение.

4.1. Данные

Для обучения моделей и замера качества их работы необходимы наборы параллельных предложений с опечатками и их исправленными версиями. В ходе исследования литературы было обнаружено несколько популярных датасетов по данной тематике:

- *BEA60k* [4] содержит 63k предложений и 70k опечаток (6.8% от всех токенов). Датасет собран агрегацией множества сочинений написанных реальными людьми.
- *JFLEG* [10] содержит 1.6k предложений и 2k опечаток (6.1% от всех токенов). Также собран на основе реальных сочинений.
- *One billion word benchmark* [12] - корпус чистых предложений без опечаток, суммарно состоящий из 1 миллиарда слов. Он используется для того чтобы на его основе построить датасет параллельных предложений для исправления опечаток, применяя различные техники, целенаправленно добавив ошибки к некоторой доле токенов.

Для дообучения Трансформерной модели требуется большое количество данных, поэтому в качестве Train - Validation датасетов было решено использовать специально зашумлённую версию *One billion word benchmark*, взятую из работы [4]. Суммарно для обучения получилось 5 миллионов параллельных предложений, 20% токенов которых содержат опечатки. При этом опечатки были получены 2 способами:

- Случайные удаления, замены, вставки и перестановки символов

- Замены слов на версию с опечаткой, на основе таблицы реальных человеческих опечаток [3]

- Замены символов на основе короткого символьного контекста, в соответствии с популярными человеческими опечатками, встречавшимися в таком же символьном контексте [9]

Данный датасет был разделен на Train - Validation части в отношении 4:1.

Для оценки качества был использован набор данных *BEA60k*. В ходе работы с этим датасетом была обнаружена одна важная особенность, в нем содержатся ошибки связанные с различиями между американскими и британским диалектами, например: kilometre / kilometer, favourite / favorite, acknowledgement / acknowledgment и другие. Существенная часть ошибок, которые совершили рассмотренные модели была связанна именно с этим нюансом. Действительно, с одной стороны, когда обучается языковая модель на крупном корпусе текста, сложно удостовериться в том, что диалекты, которые в нем используются, консистентны. С другой стороны, при разработки системы исправления опечаток, которая будет работать единообразно для пользователей по всему миру, нет смысла учить модель какому-то фиксированному диалекту.

4.2. Метрики качества

В виде ключевых метрик качества системы исправления опечаток были выбраны:

- *Word-level accuracy* - доля верно написанных слов в итоговом предложении, после применения модели (исходно в датасете *BEA60k* без ошибок написаны только 94% слов).

- *Precision* - показатель насколько часто исправление модели действительно корректное.

- *Recall* - доля исправленных ошибок от общего количества ошибок.

- $F_{0.5}$ - мера, которая аггрегирует *Precision* и *Recall*, придавая большее значение именно *Precision*. Мотивация для рассмотрения этой метрики в том, что хочется учесть и *Precision*, и *Recall*, однако, с точки зрения качественного продукта, важнее меньшее число неверных исправлений, нежели попытка покрыть абсолютно все опечатки.

Таблица показывает определения *true positive (TP), false positive(FP), false negative (FN)* и *true negative (TN)* в этой работе, чтобы избежать разных интерпретаций. Метрики рассчитываются по формулам:

$$Accuracy = (TP + TN)/(TP + FP + FN + TN)$$
$$Precision = TP/(TP + FP)$$
$$Recall = TP/(TP + FN)$$
$$F_\beta = (1 + \beta^2) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall}$$

$\beta = 0.5$ в этой работе.

= Верное Слово?	Исходное слово	Предсказание
True Positive	✗	✓
False Positive	✓	✗
False Negative	✗	✗
True Negative	✓	✓

Табл. 1: Определение True Positive (TP), False Positive (FP), False Negative (FN) и True Negative (TN). ✓ означает что исходное слово или предсказание совпадает с верным слово и наоборот для ✗.

В такой трактовке *False Negative* случаев есть важный нюанс - когда в слове была ошибка, а алгоритм ее не исправил, то может быть два случая: 1) алгоритм ее не заметил и не сработал, 2) алгоритм ее заметил, сработал, но исправил неправильно (то есть это некорретно считать "Negative"случаем, модель все-таки сработала). Так что формально в данной задаче правильнее разделять не 4 слу-

чая, а 5. Тем не менее, для упрощения, чтобы было 4 случая по аналогии с задачей классификации, метрики принято считать по формулам, описанным выше.

Кроме того, для разработки качественной модели ранжирования, для интеграции в IDE JetBrains, использовалась классическая метрика ранжирования - *Precision@1*, которая отражает частоту, с которой результат ранжирования ставит верный ответ на первое место:

$$Precision@1 = \frac{1}{N} \cdot \sum_{i=1}^{N} [rank_{gt}(i) == 1]$$

Для оценки качества модели ранжирования использовались только те предложения и слова с опечатками в них, в которых детектор распознал ошибку, а генератор кандидатов предоставил, среди прочих, верный вариант исправления. Это было сделано для того, чтобы не рассматривать случаи, когда модель даже теоретически не смогла бы найти верный вариант.

4.3. Baselines

В целях сравнения в работе приведены результаты нескольких существующих нейросетевых моделей, использующих контекст для исправления опечаток:

- Jayanthi et al. (2020) [4]: BERT
- Jayanthi et al. (2020) [4]: CHAR-LSTM-LSTM
- Sakaguchi et al. (2017) [14]: SC-LSTM

Важно отметить, что кроме этих подходов, есть и другие интересные модели, описанные в обзоре литературы, однако они не были выложены в открытый доступ.

Все реализации данных моделей взяты из [4].

4.4. Результаты

В таблице приведено сравнение качества работы исследованных моделей на датасете BEA60k, состоящем из реальных человеческих ошибок, по метрикам *Word-*

level accuracy, Precision, Recall, $F_{0.5}$. Жирным выделены лучшие значения метрик.

Метрики качества моделей на датасете BEA60k				
	Precision	Recall	$F_{0.5}$	Word-level accuracy
Char-level seq2seq	0.64	0.82	0.67	0.96
Token-level seq2seq	0.69	0.85	0.72	0.96
Sep-Mask-One BART	0.7	0.89	0.73	0.97
Sep-Mask-All BART	**0.73**	0.91	**0.76**	**0.97**
Mask-Word BART	0.14	0.30	0.16	0.95
BERT	0.69	0.79	0.71	0.94
CHAR-LSTM-LSTM	0.66	0.77	0.68	0.96
SC-LSTM	0.62	0.76	0.64	0.95
D,C,R=BART prob	0.6	0.91	0.64	0.96
D,C,R=Sep-Mask-All BART prob	0.62	**0.92**	0.66	0.96

Табл. 2: Замеры предложенных и существующих моделей. Все модели обучались на синтетически сгенерированных данных из *One Billion Word Benchmark*.

Метрики качества моделей ранжирования на BEA60k	
	Precision@1
BART prob	0.96
Sep-Mask-All BART prob	**0.98**

Табл. 3: Замеры предложенных моделей ранжирования. Все модели обучались только на тех примерах из *One Billion Word Benchmark*, в которых Hunspell детектор, сработал правильно, а затем Hunspell генератор кандидатов предложил, среди прочих, верный вариант.

Выводы:

- *Авторегрессионное исправление опечаток - эффективный подход*
  Sep-Mask-All BART fine-tune модель показала оптимальное качество, что доказывает работоспособность такого подхода.

- *Char-level исправление значимо хуже по качеству, чем token-level*
  В процессе экспериментов с данной моделью были протестированы разные версии размеров Трансформерной нейронной сети: размер внутреннего представления = {128, 256, 512}, число слоев кодировщика и декодировщика = {6, 12}. Не смотря на то, что увеличение числа параметров всегда влекло за собой повышение качества, тем не менее, значимый разрыв в качестве по сравнению с token-level моделями преодолеть не удалось.

  То есть, все-таки, информация об орфорграфии лучше используется в token-level моделях, несмотря на то что они больше подвержены к неправильной токенизации в случае ошибок.

- *Явное указание seq2seq модели на ошибку повышает качество исправлений*
  Модель BART Sep-Mask-All работает существенно качественнее, чем прямолинейный end-2-end BART fine tune. Таким образом, если удается построить модель детекции с высокой полнотой, стоит воспользоваться ей, как вспомогательным инструментом, для основной модели исправлений.

- *Дообучение предобученной языковой модели исправлять только одну ошибку за раз может ухудшить качество модели*

  Анализируя причины такого низкого качества модели Sep-Mask-One BART fine tune, был сделан вывод, что проблема именно в том, что обучая модель исправлять одну ошибку за раз, модель также учится не исправлять все остальные ошибки, что противоречит предобучению BART.

- *Чтобы дообучать предобученный seq2seq генерировать вместо предложения всего одного слово, требуются специальный подход*

  На примере с Mask-Word было обнаружено, что такая идея как генерация одного слово - варианта верного написания, вместо генерации всего предложения, может потерпеть неудачу из-за большого влияния опыта, накопленного в процессе предобучения. Модели не удалось полностью перестроиться под новую постановку.

5. Заключение

В этой работе было исследовано несколько способов построения модели исправления опечаток. Было показано, что char-level модели существенно проигрывают token-level моделям на этой задаче. Кроме того, было обнаружено, что если явное указание модели на слова с опечатками, повышает качество исправлений. Лучшим из рассмотренных подходов оказался Sep-Mask-All BART fine-tune, который объединяет в себе идеи использования сильной предобученной языковой модели и указания модели на слова с ошибками через механизм маскирования. На основе лучшего подхода удалось предложить эффективную, ускоренную модель ранжирования кандидатов на исправление опечатки, которая интегрируется в IDE JetBrains.

На открытом датасете реальных опечаток BEA60k было установлено, что предложенный метод превосходит по качеству исправления ошибок открыто доступные существующие методы, тем самым доказывая, что авторегрессионная генерация исправления ошибки является эффективным подходом.

Полученные модели поддерживают использование через CLI. Модели, код обучения и датасеты доступны для публичного использования и выложены в отрытый доступ по адресу: https://github.com/melnikoff-oleg/spellchecker.

Список литературы

[1] Atkinson K. GNU Aspell. — 2019. — Access mode: http://aspell.net/.

[2] BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension / Lewis M., Liu Y., Goyal N., Ghazvininejad M., Mohamed A., Levy O., Stoyanov V., and Zettlemoyer L. // CoRR. — 2019. — Vol. abs/1910.13461. — arXiv : 1910.13461.

[3] Belinkov Y., Bisk Y. Synthetic and Natural Noise Both Break Neural Machine Translation // CoRR. — 2017. — Vol. abs/1711.02173. — arXiv : 1711.02173.

[4] Jayanthi S. M., Pruthi D., Neubig G. NeuSpell: A Neural Spelling Correction Toolkit // CoRR. — 2020. — Vol. abs/2010.11085. — arXiv : 2010.11085.

[5] Kuznetsov A., Urdiales H. Spelling Correction with Denoising Transformer // CoRR. — 2021. — Vol. abs/2105.05977. — arXiv : 2105.05977.

[6] Li X., Liu H., Huang L. Context-aware Stand-alone Neural Spelling Correction // CoRR. — 2020. — Vol. abs/2011.06642. — arXiv : 2011.06642.

[7] Lu J. Speller100: Zero-shot spelling correction at scale for 100-plus languages. — 2021. — Access mode: https://www.microsoft.com/en-us/research/blog/speller100-zero-shot-spelling-correction-at-scale-for-100-plus-languages/.

[8] Misspelling Correction with Pre-trained Contextual Language Model / Hu Y., Jing X., Ko Y., and Rayz J. T. // CoRR. — 2021. — Vol. abs/2101.03204. — arXiv : 2101.03204.

[9] Misspelling Oblivious Word Embeddings / Piktus A., Edizel N. B., Bojanowski P., Grave E., Ferreira R., and Silvestri F. // Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers). — Minneapolis, Minnesota : Association for Computational Linguistics. — 2019. — June. — P. 3226–3234. — Access mode: https://aclanthology.org/N19-1326.

[10] Napoles C., Sakaguchi K., Tetreault J. JFLEG: A Fluency Corpus and Benchmark for Grammatical Error Correction // Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers. — Valencia, Spain : Association for Computational Linguistics. — 2017. — Apr. — P. 229–234. — Access mode: https://aclanthology.org/E17-2037.

[11] Németh L. Hunspell. — 2002. — Access mode: http://hunspell.github.io.

[12] One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling / Chelba C., Mikolov T., Schuster M., Ge Q., Brants T., and Koehn P. // CoRR. — 2013. — Vol. abs/1312.3005. — arXiv : 1312.3005.

[13] Ozinov F. Jamspell. — 2019. — Access mode: https://jamspell.com.

[14] Robsut Wrod Reocginiton via semi-Character Recurrent Neural Network / Sakaguchi K., Duh K., Post M., and Durme B. V. // CoRR. — 2016. — Vol. abs/1608.02214. — arXiv : 1608.02214.