Аннотация

В данной работе представлены два исследования, посвящённые задаче исправления грамматических ошибок в текстах с использованием подхода Sequence Tagging. В первом исследовании описывается адаптация модели GECToR для русского языка. С учетом недостатка размеченных данных, для обучения модели был создан синтетический набор данных. Разработанная модель показала хорошие результаты на синтетических данных $F_{0.5} = 82.5$, а также продемонстрировала способность к переносу знаний на тестовый набор данных RULEC без дополнительного обучения $F_{0.5} = 22.2$.

Во втором исследовании предлагается полностью автоматизированный, не требующий разметки подход к решению задачи исправления грамматических ошибок. Метод основан на генерации данных с использованием алгоритма Левенштейна для исправления грамматических ошибок на уровне подслов с использованием правил: keep, append, replace и delete. Подход универсален для любого языка и не требует дополнительной разметки. Применение данного метода к оригинальной модели GECToR позволило достичь конкурентных результатов на английском языке: $F_{0.5} = 62.4$ на CoNLL-2014 и $F_{0.5} = 61.9$ на BEA-2019, при этом не потребовалось ни аннотирования данных, ни составления словаря грамматических правил.

Таким образом, совместное рассмотрение обоих исследований демонстрирует возможности применения и адаптации Sequence Tagging моделей как для языков с достаточным количеством размеченных данных, так и для языков, где количество таких данных ограничено.

Ключевые слова: исправление грамматических ошибок; обработка естественного языка; алгоритм левенштейна; трансформер.

Содержание

1 Введение

В настоящее время задача автоматической коррекции грамматических ошибок приобретает всё большую значимость [1, 2, 3], что связано с ростом объёмов текстовой информации [4]. Проверка больших текстов вручную требует значительных затрат времени и усилий. Таким образом, исследования в данной области необходимы для автоматизации данных процеессов. Например, при проверке письменных работ [5] преподаватели тратят большое количество времени на оценку каждого ученика, хотя было бы менее трудоемко для первичной оценки использовать автоматический алгоритм, а далее валидировать полученный результат с помощью учителя. Сюда же относится и проверка грамотности в социальных сетях — хотя автоматическая проверка орфографии уже существует, исправление грамматики до сих пор остаётся нерешённой задачей [6]. Это может быть использовано не только для социальных сетей, но и для разработки приложения по изучению языков — редактирование текстов, созданных иностранными студентами [7] Также намеренные грамматические ошибки могут использоваться для обхода систем обнаружения плагиата [8, 9]. Это приводит к возникновению состязательных атак на модели обнаружения плагиата [10], которые изначально обучались на наборах данных, содержащих минимальное количество ошибок.

В настоящее время большинство методов автоматического исправления грамматики ориентированы на английский язык [11], где подобные системы показывают высокую эффективность. В то время как для других языков, включая русский, количество исследований остается ограниченным [12]. Основная сложность заключается в морфологическом богатстве русского языка, что создает существенные препятствия для разработки эффективных систем коррекции грамматических ошибок.

Среди наиболее эффективных подходов к автоматическому исправлению грамматических ошибок в английском языке выделяются две основные архитектуры: Sequence-to-Sequence (Seq2Seq) [13, 14] и Sequence Tagging (ST) [15, 16].

Модели типа Seq2Seq работают по принципу “чёрного ящика”: на

вход подаётся ошибочное предложение, а на выходе генерируется исправленный вариант. Несмотря на высокую точность таких систем, они имеют существенные ограничения:

1. низкая производительность из-за вычислительной сложности полной генерации текста;

2. слабая интерпретируемость результатов, требующая дополнительного анализа для определения типов ошибок.

В то время как при использовании моделей на основе архитектуры ST исправление происходит на уровне токенов, ошибки помечаются и корректируются в исходной последовательности без полной перегенерации текста. Также данное решение является более интепретируемым, поскольку каждое исправление соотносится с конкретным грамматическим правилом из словаря. Также с точки зрения эффективности обеспечивается лучшая производительность за счёт локального характера исправлений

В исследовании [12] проводится сравнительный анализ данных подходов на материале русскоязычных текстов. Результаты показывают, что ST-модели значительно превосходят Seq2Seq при ограниченном объеме размеченных данных. Эксперименты выполнены на корпусе RULEC, который был разработан авторами исследования. Набор данных состоит из предложений иностранных студентов с аннотированными грамматическими ошибками. Данный корпус был использован для оценки работы нашей модели в экспериментах, но не был использован для ее обучения.

GECToR [15] в настоящее время считается одной из наиболее эффективных ST-моделей для английского языка. Модель построена на основе архитектуры трансформер [17] и использует двухголовый механизм классификации. Первая голова отвечает за детекцию ошибок, вторая — за определение типа исправления. Если ошибка не обнаружена, применяется специальная метка “$KEEP”, сохраняющая исходный токен без изменений. В экспериментах были рассмотрены различные кодировщики: BERT [18], RoBERTa [19] и XLNet [20].

Обучение модели проводилось в три этапа: обучение на синтетических данных, дообучение на корпусе с ошибками и дообучение на объединённом корпусе с ошибками и без них.

В рамках магистерской работы были выполнены два ключевых исследования:

1. В рамках первого исследования был разработан алгоритм генерации синтетических данных для русского языка. На его основе создана русскоязычная адаптация модели GECToR — RuGECToR [21]. Хотя обучение проводилось исключительно на синтетических данных, модель была ориентирована на выявление и обобщение грамматических закономерностей русского языка, а не на простое запоминание примеров.

2. Предложен способ автоматического построения словаря корректирующих правил и последующего обучения ST-моделей с его использованием. Метод опирается на уровень SentencePiece-токенов [22] и ограничивается универсальными преобразованиями: *keep*, *append*, *replace* и *delete*. Такой подход позволяет устранить зависимость от грамматических правил, полностью автоматизируя процесс. Метод был реализован на примере модели GECToR и показал сопоставимые результаты, при этом обеспечивая лёгкую адаптацию к другим языкам без необходимости ручной аннотации или знаний грамматики.

2 Адаптация модели GECToR для русского языка

2.1 Постановка задачи

Задано множество пар, в котором каждая пара состоит из предложения x_i и соответствующего ему эталонного исправления t_i:

$$S = \{(x_i, t_i)\}_{i=1}^{N},$$

где N – количество пар.

В данной работе предполагается, что каждое предложение представлено в виде последовательности токенов $x_i = \{s_1, s_2, \ldots, s_{n_i}\}$, а каждое эталонное исправление – в виде последовательности корректирующих правил $t_i = \{t_1, t_2, \ldots, t_{n_i}\}$, где n_i – длина i-го предложения.

Под понятием «токен» подразумевается слово, таким образом разбиение предложения на токены происходит на уровне слов. Каждое корректирующее правило t_j является элементом заданного нами словаря. Составление словаря описано ниже.

Целью задачи является нахождение функции T для построения отображения из последовательности токенов x_i в последовательность корректирующих правил t_i:

$$T : x_i \mapsto t_i \in \{0, 1, \ldots, k\}^{n_i},$$

где k – размер словаря корректирующих правил.

2.2 Предлагаемый подход

На вход модели GECToR подается предложение, которое требуется исправить. Далее предложение разбивается на последовательность токенов. Таким образом, задача исправления грамматических ошибок сводится к нахождению отображения T для сопоставления каждого токена с соответствующим правилом из словаря корректирующих правил. Для русского языка построен словарь, состоящий из 5183 правил, которые будут подробно описаны ниже.

Данная постановка задачи имеет недостаток: каждый токен сопоставляется только с одним правилом из словаря, но иногда для исправления ошибки требуется большее количество изменений. Для решения данной проблемы предлагается использовать итеративное исправление: предложение, полученное в результате работы модели, снова подается модели на вход. Таким образом, модель предсказывает правила для новой последовательности токенов. Правила разработаны таким образом, что последовательность токенов с ошибками может быть преобразована в последовательность токенов без ошибок за конечное число итераций. Проделаны следующие адаптации модели GECToR для русского языка:

1. Проведено обучение модели RUGECToR в два этапа:
   (a) обучение на синтетических данных с ошибками;
   (b) дообучение на комбинации из синтетических данных с ошибками и без ошибок;
2. Изменен этап применения модели для исправления ошибок с использованием библиотеки pymorphy2 [23];
3. Составлен словарь правил для исправления ошибок;
4. В качестве кодировщика использован Multilingual BERT.

2.3 Описание набора данных

исходное предложение	предложение с ошибками	корректирующие правила
'Человеческий', 'дух', 'непостижимо', 'могуч', ',', и', 'убить', 'его', 'в', 'человеке', 'почти', 'невозможно', '.'	'Человеческий-дух', 'непостижимо', 'могуч', ',', 'и', 'убить', 'его', 'в', 'чел овеке', 'почти', 'невозможно', '.'	$TRANSFORM_SPLIT_HYPHEN, $KEEP, $KEEP, $KEEP, $KEEP, $KEEP, $KEEP, $KEEP, $MERGE_SPACE, $KEEP, $KEEP, $KEEP, $KEEP

Таблица 1: Пример генерации ошибок в предложении

Для генерации синтетических ошибок на первом этапе обучения взяты предложения из русскоязычной Википедии и школьных сочинений [24]. Для второго этапа обучения использованы школьные сочинения и литературные тексты [25]. Сгенерированы специализированные ошибки для следующих частей речи: глагол, имя прилагательное, имя существительное, местоимение, причастие и числительное. Перед началом процесса генерации ошибок пронумерованы позиции токенов для указанных частей речи во всех предложениях. Процесс генерации ошибок осуществляется следующим образом:

1. случайным образом выбирается предложение, где для каждого токена случайным образом генерируется ошибка, соответствующая части речи токена;

2. после этого каждому такому токену ставится в соответствие правило из словаря корректирующих правил, которое необходимо применить для исправления ошибки.

Распределение ошибок близко к равномерному. В табл. 1 показан пример генерации ошибок в предложении.

В табл. 2 показано, что для первого этапа обучения использовалось 10.000.000 предложений, где каждое предложение содержит произвольные типы ошибок кроме пунктуационных. Для второго этапа обучения мы использовали 1.000.000 предложений: 500.000 предложений не содержат ошибок; 250.000 предложений содержат все ошибки, кроме пунктуационных; оставшиеся 250.000 предложений

набор данных	**#предложений**	**% предложений с ошибками**	**этап обучения**
Wiki + Essays	10 000 000	$\approx 100\%$	I
Proza + Essays	1 000 000	$\approx 50\%$	II

Таблица 2: Информация об обучающем наборе данных

содержат только пунктуационные ошибки. Для проведения второго этапа обучения добавлены литературные произведения и исключена Википедия с целью сделать модель более устойчивой к различиям между наборами данных [26]. В качестве тестовых наборов данных использовались школьные сочинения и тестовое подмножество набора данных RULEC. В работе [15] авторы разделяют правила, применяемые к исходным токенам $\{x_1, x_2, \ldots, x_{n_i}\}$ для получения целевого предложения, на два типа – базовые и грамматические. В нашем исследовании сохранено данное разделение. Грамматические правила подобраны таким образом, чтобы словарь корректирующих правил покрывал множество правил русского языка. Базовые правила выполняют наиболее распространенные операции редактирования на уровне токенов: сохранение текущего токена x_i без изменений – правило $\$KEEP$, удаление текущего токена x_i – правило $\$DELETE$, добавление нового токена t_1 после текущего токена x_i – правило $\$APPEND\_t_1$ или замена текущего токена x_i на другой токен t_2 – правило $\$REPLACE\_t_2$. Для генерации правил \$APPEND и \$REPLACE использованы 2500 наиболее употребляемых русских слов с точки зрения коэффициента Жуайна [27]. Для каждого слова w_i добавлены соответствующие правила $\$APPEND\_w_i$ и $\$REPLACE\_w_i$. Грамматические правила выполняют операции, специфичные для конкретного случая. Для русского языка использованы следующие правила, которые применяются непосредственно к токену: изменение времени, падежа, рода, лица и числа. Добавлены правила для часто допускаемых ошибок, например, правописание “ться”/“тся” и “при”/“пре”. Также использованы правила, которые не зависят от языка: объединение двух слов с

помощью пробела или дефиса; разделение слова, написанного через дефис, на две части; изменение регистра первой буквы слова. Рассмотрим пример корректирующего правила для исправления “красивая” на “красивый”. Таким образом, необходимо преобразовать прилагательное из женского рода в мужской. Изначально правила для грамматических преобразований создавались следующим образом:

$TRANSFORM_ADJF_GEND_femn_masc

Такие правила содержали подробную информацию о:

1. части речи слова;
2. грамматическом признаке, который необходимо изменить;
3. граммеме как для исходного, так и для исправленного слова.

Далее было решено объединить правила для разных частей речи, исключив название части речи. Кроме того, решено отказаться от указания граммемы для исходного (неправильного) слова, поскольку для его исправления необходима информация только о новой граммеме. В данном исследовании предполагается, что модель сама научится сопоставлять правила соответственно частям речи. Например, глагол и прилагательное могут менять род, а существительное – нет. Таким образом, итоговые правила, используемые в нашей модели, выглядят следующим образом:

$TRANSFORM_GEND_masc

Данная модификация была сделана для того, чтобы уменьшить размер словаря и увеличить количество примеров для каждого правила в обучающем наборе данных. Таким образом достигается компромисс между обобщающей способностью и размером модели, так как с ростом количества правил увеличивается размер слоя классификации.

Для оценки полноты составленного набора правил было проведено сравнение с классификатором ошибок в наборе данных RULEC.

Для большинства ошибок из набора данных RULEC в нашем словаре существуют аналогичные правила. Но в наборе данных RULEC существуют и более узкие правила, например, “Местоимение”, “Сущ.: число”, означающие наличие ошибки в написании множественного/ единственного в соответствующих частях речи. Для данных ошибок в нашем словаре нет специализированных правил, но тем не менее они покрываются более широкими правилами для исправления написания множественного/единственного числа без привязывания к конкретным частям речи. Также существуют правила, такие как “Орфография”, “Союз” и “Предлог”, аналогов которым нет в нашем словаре. Исправление ошибок, связанных с данными правилами, предусмотрено правилами $\$APPEND\_w_i$ и $\$REPLACE\_w_i$.

2.4 Эксперименты

В качестве предобученного кодировщика на основе архитектуры трансформер выбран Multilingual BERT [28], имеющий хорошую языковую обобщаю способность. Эксперименты по сравнению различных кодировщиков на основе архитектуры трансформер запланированы в дальнейших работах. Обучение модели проводилось в течение двух этапов, каждый из которых длился 50 эпох. На первом этапе обучения размер батча равен 32, на втором – 16. В качестве оптимизатора параметров выбран Adam с параметром $lr = 10^{-5}$. Для работы модели на этапе применения было выявлено, что количество исправлений уменьшается с каждой успешной итерацией, и необходимое число исправлений выполняется в течение первых трех итераций.

2.5 Метрики качества

Сравнение качества работы моделей проводилось на синтетических и реальных данных. Для оценки качества использованы метрики, описанные в[11]:

$$R = \frac{\sum_{i=1}^{N} |g_i \cap e_i|}{\sum_{i=1}^{N} |g_i|}, \quad P = \frac{\sum_{i=1}^{N} |g_i \cap e_i|}{\sum_{i=1}^{N} |e_i|},$$

$$F_{0.5} = \frac{(1 + 0.5^2) \cdot R \cdot P}{R + 0.5^2 \cdot P},$$

где N – количество предложений, e_i – множество исправлений, предсказанных нашей моделью для предложения s_i, а g_i – множество эталонных исправлений. Пересечение между g_i и e_i определяется как:

$$g_i \cap e_i = \{e \in e_i \mid \exists g \in g_i \colon e = g\}.$$

2.6 Результаты

System	Training stage	P	R	$F_{0.5}$
RuGECToR	I	88.4	67.1	83.1
RuGECToR	II	88.5	65.1	82.5

Таблица 3: Качество работы модели на синтетическом тестовом наборе данных

Для синтетического тестового набора данных сгенерировано 10.000 предложений с ошибками. Результаты на синтетическом наборе данных приведены в табл. 3. Видно, что метрики R и $F_{0.5}$ на втором этапе обучения ниже, чем на первом. Это можно объяснить двумя причинами:

1. 50% предложений в данных, использованных для второго этапа обучения, не содержали ошибок. Точность стала выше, но при этом количество охватываемых токенов уменьшилось.

2. На втором этапе обучения были использованы дополнительные предложения из литературных произведений, которые не использовались во время первого этапа обучения. Таким образом, была увеличена обобщающая способность модели за счет добавления данных из другого распределения.

модель	данные для обучения data	P	R	$F_{0.5}$
Classifiers (learner)	RULEC	22.6	4.8	12.9
Classifiers (min sup.)	RULEC	38.0	7.5	21.0
MT	RULEC	30.6	2.9	10.6
RuGECToR	Synthetic (I stage)	23.6	5.6	14.3
RuGECToR	Synthetic (II stages)	**40.8**	**7.9**	**22.2**

Таблица 4: Сравнение качества работы моделей на наборе данных RULEC

В качестве реальных тестовых данных использован набор данных RULEC. Результаты приведены в табл. 4. Сравнение производилось с моделями Classifiers (learner), Classifiers (minimal sup.) и MT. Данные модели были представлены в [12] и обучены на наборе данных RULEC. Модель RuGECToR достигает значения равного 22.2 с точки зрения метрики $F_{0.5}$ на наборе данных RULEC. Это значение выше результата, демонстрируемого вышеперечисленными моделями, несмотря на то, что модель RuGECToR на нем не обучалась. Таким образом, модель обладает хорошей обобщающей способностью и не переобучается под конкретный набор данных. Следует также отметить, что на синтетических тестовых данных наша модель работает лучше, чем на реальных данных. Это вполне ожидаемо, так как синтетический обучающий и синтетический тестовый наборы данных имеют схожие распределения, в то время как RULEC сильно отличается от синтетического обучающего набора данных. Табл. 4 также показывает, что обобщающая способность модели на втором этапе обучения растет. На наборе данных RULEC качество работы модели после второго этапа значительно выше, чем после первого.

3 Универсальный подход для исправления грамматических ошибок на уровне подслов

3.1 Мотивация перехода на уровень подслов

Токенизация является важнейшим этапом вычислительной обработки текста [29, 30], поскольку позволяет сократить размер словаря и повысить эффективность работы модели. В зависимости от архитектуры модели применялись различные методы токенизации: Byte Pair Encoding (BPE) для RoBERTa, WordPiece для BERT и SentencePiece для XLNet. В данном исследовании под "подсловом" понимается минимальная единица текста, полученная в результате разбиения строки символов с помощью токенизатора соответствующей модели на основе архитектуры "Трансформер".

В подходе [15], основанном на ST-модели, словарь корректирующих правил формируется на уровне целых слов. Таким образом, во время обучения и предсказания все подслова, входящие в состав одного слова, ассоциируются с единым правилом исправления из словаря.

В работе [15] вводится разделение преобразований на базовые и грамматические (g-transformations). Базовые преобразования ограничиваются простейшими операциями редактирования — удалением, сохранением, заменой и добавлением слов. В отличие от них, грамматические преобразования требуют глубокого лингвистического анализа и включают такие операции, как изменение глагольных форм, коррекцию числа и падежа существительных, а также слияние или разделение слов.

Реализация грамматических преобразований для произвольного языка ставит три ключевые задачи: формализацию грамматических правил языка, наличие размеченных обучающих данных и разработку механизма сопоставления экспертных меток с корректирующими правилами. На уровне подслов нет необходимости разрабатывать специфические для задачи операции, поскольку каждое слово

может быть исправлено конечным числом “базовых преобразований” без полного изменения слова.

Это ключевой момент, так как задача может быть решена с помощью обучения без учителя:

1. Отсутствие необходимости в размеченных данных: разметка на уровне подслов автоматически генерируется при обратном проходе по матрице расстояний, полученных с помощью алгоритма Левенштейна. Данный алгоритм поддерживает все базовые операции редактирования — удаление, сохранение без изменения, замену и добавление. Для работы метода достаточно наличия параллельных текстов — исходные предложения с ошибками и их исправленные версии.

2. Отсутствие необходимости в ручной разработке корректирующих правил: все корректирующие правила автоматически извлекаются из базовых преобразований, применяемых на уровне подслов, что устраняет необходимость трудоемкой разработки словарей грамматических правил.

В данной работе предложен метод исправления грамматических ошибок на уровне подслов, который можно обобщить для различных языков без необходимости лингвистического анализа грамматических правил или доступа к размеченным данным.

3.2 Обновленная постановка задачи

Обозначим множество исходных последовательностей как

$$\mathcal{X} = \{s_i \mid s_i = [x_1, x_2, \ldots, x_{n_i}]\}_{i=0}^{N},$$

где каждая исходная последовательность s_i, разбитая с помощью токенизатора, представлена в виде последовательности подслов длины n_i, а N — общее количество предложений.

Множество целевых последовательностей с токенизацией длины m_i аналогично задаётся как

$$\mathcal{Y} = \{t_i \mid t_i = [y_1, y_2, \ldots, y_{m_i}]\}_{i=0}^{N}.$$

Пусть задан словарь корректирующих правил W, содержащий правила: *добавить* (append), *сохранить* (keep), *заменить* (replace) и *удалить* (delete).

Требуется найти множество всех возможных последовательностей корректирующих преобразований с минимальным числом операций вставки, удаления и замены:

$$\mathcal{F} = \{w_{ij} \mid w_{ij} \circ s_i \to t_i\}_{i=0}^{N},$$

где последовательность корректирующих преобразований из словаря W задаётся как

$$w_{ij} = \{w_1^*, w_2^*, \ldots, w_{n_i}^*\}_j, j \in \{0, \ldots, o_i\},$$

где o_i — количество последовательностей корректирующих преобразований минимальной длины, преобразующих s_i в t_i.

Операция $\circ$ обозначает покомпонентное применение правил коррекции к подсловам.

3.3 Генерация синтетических аннотированных данных

В этом разделе описан процесс генерации синтетических аннотированных данных. Исходная последовательность $s_k = \{x_1, x_2, \ldots, x_{n_k}\}$ — это токенизированное k-е предложение с ошибками, целевая последовательность — токенизированное k-е предложение без ошибок $t_k = \{y_1, y_2, \ldots, y_{m_k}\}$, где n_k и m_k — длины исходной и целевой последовательностей соответственно.

В нашей формулировке задача сводится к поиску редакционного предписания.

Используется расстояние Левенштейна, которое определяется как минимальное число операций *вставки* (append), *удаления* (delete) и *замены* (replace), необходимых для преобразования одной последовательности в другую.

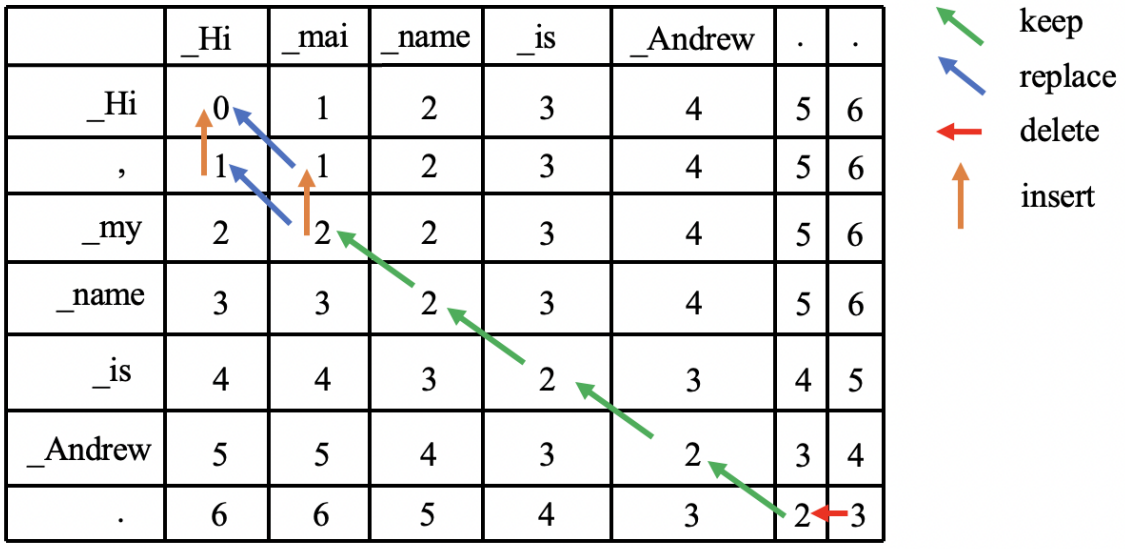

Рис. 1: Матрица Левенштейна и редакционные предписания между исходной и целевой последовательностями. Можно заметить, что существует несколько вариантов редакионных предписаний.

Расстояние Левенштейна между исходной и целевой последовательностями эффективно вычисляется с помощью двумерной матрицы, которая заполняется с использованием алгоритма динамического программирования. Пусть $D_{i,j}$ — это расстояние редактирования между префиксами $s_k[0..i]$ и $t_k[0..j]$ длины i и j, при этом $D_{0,j} = 0$ и $D_{i,0} = 0$. Остальные значения определяются следующим рекуррентным соотношением:

$$D_{i,j} = \begin{cases} D_{i-1,j-1}, & \text{если } s_k[i] = s_k[j] \\ 1 + \min \begin{cases} D_{i-1,j}, \\ D_{i,j-1}, \\ D_{i-1,j-1} \end{cases}, & \text{иначе} \end{cases}$$

На рисунке 1 показана матрица расстояний между префиксами токенизированного исходного предложения «Hi mai name is Andrew..» и токенизированного целевого предложения «Hi, my name is Andrew.». Расстояние Левенштейна между исходной и целевой последователь-

ностями отображается в правом нижнем углу матрицы и равно 3.

Редакционное предписание — это последовательность правил (*keep*, *replace*, *append*, *delete*) минимальной длины, описывающая действия, необходимые для получения целевой последовательности из исходной.

Чтобы найти эталонные корректирующие преобразования, необходимо определить редакционное предписание. Как показано на рисунке 1, предписание можно получить с помощью обратного прохода по рассчитанной матрице расстояний. Начиная с правого нижнего угла таблицы, строим пути ко всем соседним ячейкам, соответствующим меньшим подзадачам и меньшим расстояниям редактирования. Движение вверх соответствует вставке подслова в исходную последовательность (*append*), влево — удалению подслова (*delete*), по диагонали — замене (*replace*) или сохранению без изменений (*keep*), если подслова совпадают.

Утв. Количество редакционных предписаний соответствует количеству путей в графе подзадач с минимальной стоимостью.

Например, для нашего случая существует два редакционных предписания: {*delete* **.**; *append* **_my**; *replace* **_mai** на **,**} и {*delete* **.**; *append* **,**; *replace* **_mai** на **_my**}. Таким образом, нам необходимо определить оптимальное редакционное предписание. Ввиду краткости правила *keep* опущены.

Обозначим через $EP_k = \{e_1, e_2, \ldots, e_{o_k}\}$ множество редакционных предписаний для пары последовательностей (s_k, t_k), где o_k — количество предписаний для k-й пары.

Для определения оптимального редакционного предписания для k-й пары учитывается схожесть подслов при применении правила *replace*. Пусть $R_l \in e_l$, $l \in \{1, 2, \ldots, o_k\}$ — множество всех правил *replace* с мощностью p_l для произвольного предписания e_l:

$$R_l = \{replace\_t_{1i}\_t_{2i}\}_{i=0}^{p_l},$$

где $t_{1i} \in s_k$, $t_{2i} \in t_k$.

Введём метрику схожести подслов:

$$\sigma_l = \sum_{i=0}^{p_l} \text{LevenshteinDist}(t_{1i}, t_{2i}),$$

где LevenshteinDist — функция, вычисляющая расстояние Левенштейна между t_{1i} и t_{2i} на уровне символов в подсловах.

Оптимальное предписание e_l^* выбирается как:

$$l = \arg\min\{\sigma_1, \sigma_2, \ldots, \sigma_{o_k}\}.$$

Это и есть эталонная последовательность корректирующих преобразований для пары (s_k, t_k).

3.4 Обучающая система и экспериментальные настройки

В этом разделе описаны методология и возникшие в процессе построения системы трудности. Для оценки качества предлагаемого подхода использована Sequence Tagging модель GECToR, показавшая высокую эффективность в задаче исправления грамматических ошибок. Для корректного сравнения с авторами модели использованы те же данные и повторен процесс обучения с теми же параметрами.

Датасет	**Количество предложений**		**Этап**
	Уровень подслов	Уровень слов	**обучения**
PIE-synthetic	9,000,000	9,000,000	I
Lang-8	787,613	947,344	II
NUCLE	51,929	56,958	II
FCE	25,968	34,490	II
W&I+LOCNESS	21,828	34,304	II, III

Таблица 5: Наборы данных на каждом этапе обуччения с соответствующим количеством предложений: в исследовании [15] на уровне слов и в нашем исследовании на уровне подслов.

3.5 Описание данных

В таблице 5 представлены статистики данных, использованных в исследовании [15] и в нашем исследовании. Было использовано пять датасетов: PIE-synthetic [32], Lang-8 [33, 34], NUCLE [35], FCE [36] и W&I+LOCNESS [37].

3.5.1 Ограничения алгоритма

Слабым местом данного подхода является то, что обратный проход использует рекурсивные вызовы для нахождения всевозможных редакционных предписаний и выбора наилучшего. Таким образом, для длинных последовательностей с большим количеством различий возникает множество путей. Для некоторых предложений не удалось найти эталонную последовательность корректирующих преобразований, и они были исключены из обучающего набора.

3.5.2 Обучающие данные

Как показано в таблице 5, на первом этапе обучения, аналогично [15], использовались 9 миллионов предложений из PIE-synthetic. Однако ввиду ограничений только 6.7M предложений совпадают с теми, что использовались в [15]. Чтобы довести общий объём до 9M, использовались оставшиеся предложения из PIE-synthetic.

Что касается реальных данных, из таблицы 5 видно, что использовалось только 78% от общего объёма. Эти данные применялись для дообучения на этапах II и III.

3.5.3 Данные для оценки

Для сравнения качества работы моделей использовались те же тестовые наборы, что и в [15] — CoNLL-2014 и BEA-2019. В процессе оценки применялись метрики M^2 и ERRANT соответственно.

Этап обучения	**Уровень слов**		**Уровень подслов**	
	Количество эпох	Размер батча	Количество эпох	Размер батча
I	20	256	20	32
II	9	128	7	16
III	4	128	1	16

Таблица 6: Лучшие эпохи на каждом этапе обучения модели XLNet в [15] на уровне слов и в нашем исследовании на уровне подслов с соответствующими размерами батча.

Model	**CoNLL-2014**			**BEA-2019**		
	P	R	$F_{0.5}$	P	R	$F_{0.5}$
GECToR (уровень подслов + XLNet)	72.3	40.4	62.4	70.5	41.6	61.9
GECToR (уровень слов + BERT)	72.1	**42.0**	63.0	71.5	**55.7**	67.6
GECToR (уровень слов + RoBERTa)	73.9	41.5	64.0	77.2	55.1	71.5
GECToR (уровень слов + XLNet)	**77.5**	40.1	**65.3**	**79.2**	53.9	**72.4**

Таблица 7: Сравнение работы моделей на уровне слов и на уровне подслов. Приведены оценки M^2 для CoNLL-2014 (тест) и ERRANT для BEA-2019 (тест).

3.6 Обучение модели

Архитектура модели GECToR состоит из кодировщика на базе архитектуры “Трансформер” и двух линейных классификаторов. Первый классификатор предсказывает наличие ошибки в каждом подслове. Второй классификатор выбирает конкретное правило из словаря для исправления подслова или оставляет его без изменений, если наиболее вероятное правило — *keep*.

Если максимальная вероятность первого классификатора по предложению ниже определённого порога, предложение не подлежит исправлению. В [15] этот порог равен 0.66. В нашем исследовании эм-

пирическим путем было подобрано оптимальное значение 0.05.

3.6.1 Исправление на уровне слов

В [15] словарь правил исправления был построен на уровне слов. Это означает, что несколько подслов в одном слове соотносятся с одним правилом из словаря.

В исследовании использовались два типа правил: “базовые преобразования” и “g-преобразования”. Базовые преобразования — это *delete*, *keep*, *replace* и *append*. G-преобразования — это более сложные грамматические изменения: смена формы глагола, слияние/разделение слов, изменение числа существительного и др. Для построения g-трансформаций необходимо знание грамматики языка, аннотированные данные и средства сопоставления предсказанных правил с эталонными.

Весь словарь включает 5000 правил, из которых 4971 — базовые преобразования, 29 — g-преобразования.

3.6.2 Исправление на уровне подслов

В нашем подходе отсутствуют g-преобразования, что избавляет нас от необходимости вручную составлять правила. Это особенно важно для языков с богатой морфологией, для которых обобщение грамматики вручную крайне трудоёмко.

Нами был получен словарь правил на уровне подслов, используя только базовые преобразования: *delete*, *keep*, *replace* и *append*. Процесс полностью автоматический и не требует аннотированных данных, лишь параллельный корпус. Объём итогового словаря составил 25,714 правил.

3.6.3 Параметры обучения

В качестве кодировщика использован XLNet, так как он показал наилучшие результаты в [15]. Процесс обучения был повторён с теми же параметрами, за исключением размера батча.

Авторы [15] использовали размер батча 256 на первом этапе обучения и 128 на втором и третьем этапах. В нашем случае значения составили 32 и 16 соответственно. В таблице 6 представлены лучшие эпохи для каждого этапа.

3.7 Результаты

Как показано в таблице 7, результаты были получены на наборах данных CoNLL-2014 (тест) и BEA-2019 (тест) с использованием соответствующих метрик — M^2-scorer и ERRANT.

На наборе данных CoNLL-2014 наша модель на уровне подслов показывает конкурентоспособный результат $F_{0.5} = 62.4$ по сравнению с $F_{0.5} = 65.3$ для модели на уровне слов из работы [15]. Более того, наша модель на уровне подслов немного превосходит оригинальную модель по метрике полноты: $R = 40.4$ против $R = 40.1$.

На наборе данных BEA-2019 наша модель не смогла приблизиться к результатам оригинальной модели, однако всё ещё демонстрирует достаточно высокий показатель $F_{0.5} = 61.9$. Как видно, модель показывает относительно низкую полноту, что можно объяснить тем, что был использован не весь синтетический корпус данных.

Тем не менее, было показано достижение сравнимых результатов, используя обучение без учителя. Важно отметить, что данный подход является независимым и может быть адаптирован к любому языку с использованием словаря на уровне подслов.

4 Заключение

В данной работе рассматриваются два взаимодополняющих исследования, посвящённых задаче исправления грамматических ошибок, с фокусом на морфологически сложные языки с ограниченными ресурсами. В отличие от английского языка, для которого данная задача изучена достаточно хорошо, языки с богатой морфологией, такие как русский, остаются недостаточно исследованными из-за дефицита размеченных данных и сложности коррекции ошибок в подобных языковых контекстах.

В первом исследовании представлена интерпретируемая и эффективная система исправления ошибок в русских текстах, основанная на правилах и синтетическом наборе обучающих данных. Используя эффективную архитектуру GECToR, система продемонстрировала высокую точность на синтетических данных $F_{0.5} = 82{,}5$ и показала способность к обобщению на реальных текстах $F_{0.5} = 22{,}2$, превзойдя базовые модели, не обученные напрямую на таких данных. Перспективы улучшения включают расширение словаря корректирующих правил, дообучение на дополнительных датасетах и эксперименты с различными архитектурами для повышения качества исправлений.

Второе исследование предлагает полностью неконтролируемый подход к исправлению грамматических ошибок, не требующий ручной разметки данных или языково-специфичных правил. Метод основан на базовых преобразованиях на уровне подслов и использовании выравнивания по Левенштейну для генерации исправлений. Продемонстрировав эффективность для английского языка, $F_{0.5} = 62{,}4$ на CoNLL-2014 и $F_{0.5} = 61{,}9$ на BEA-2019, подход показал свою конкурентоспособность. Его применение особенно перспективно для малоресурсных языков, где ощущается острый дефицит размеченных данных. Дальнейшие исследования могут быть направлены на адаптацию метода к другим языкам, а также на изучение влияния различных архитектур “Трансформер” и стратегий токенизации на качество исправлений.

Вместе эти исследования демонстрируют перспективность как

гибридных (основанных на правилах), так и полностью неконтролируемых подходов для решения задачи исправления грамматических ошибок в условиях ограниченных языковых ресурсов. Если первая работа подтверждает эффективность синтетических данных и структурированных корректирующих правил, то вторая доказывает возможность достижения конкурентоспособных результатов без использования размеченных данных.

Список литературы

[1] Rozovskaya A., Roth D. Grammatical Error Correction: Machine Translation and Classifiers // Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). — Berlin, Germany: Association for Computational Linguistics, 2016. — P. 2205–2215.

[2] Yuan Z., Stahlberg F., Rei M., Byrne B., Yannakoudakis H. Neural and FST-based approaches to grammatical error correction // Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications. — Florence, Italy: Association for Computational Linguistics, 2019. — P. 228–239.

[3] Bryant C., Ng H.T. How Far are We from Fully Automatic High Quality Grammatical Error Correction? // Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics. — Beijing, China: Association for Computational Linguistics, 2015. — P. 697–707.

[4] Rajput D. Review on recent developments in frequent itemset based document clustering, its research trends and applications // International Journal of Data Analysis Techniques and Strategies. — 2019. — Vol. 11. — P. 176–195.

[5] Flickinger D., Yu J. Toward More Precision in Correction of Grammatical Errors // Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task. — Sofia, Bulgaria: Association for Computational Linguistics, 2013. — P. 68–73.

[6] Yuan X., Pham D., Davidson S., Yu Z. ErAConD: Error Annotated Conversational Dialog Dataset for Grammatical Error Correction // Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. — Seattle, USA: Association for Computational Linguistics, 2022. — P. 76–84.

[7] Lee J.S.Y., Seneff S. An analysis of grammatical errors in non-native speech in English // 2008 IEEE Spoken Language Technology Workshop. — 2008. — P. 89–92.

[8] Zhuravlev K., Rudakov K., Inyakin A., et al. The system of recognition of intellectual text reuse "Antiplagiat" // Mathematical methods of pattern recognition: 12th All-Russian conference: Collection of reports. — Moscow: MAKS Press, 2005. — P. 329–332.

[9] Keck C.M. How Do University Students Attempt to Avoid Plagiarism? A Grammatical Analysis of Undergraduate Paraphrasing Strategies // Writing & Pedagogy. — 2010. — Vol. 2. — P. 193–222.

[10] Zhang W.E., Sheng Q.Z., Alhazmi A., Li C. Adversarial Attacks on Deep-Learning Models in Natural Language Processing: A Survey // ACM Transactions on Intelligent Systems and Technology. — 2020. — Vol. 11, No. 3. — Art. 24. — P. 1–41.

[11] Ng H.T., Wu S.M., Briscoe T., et al. The CoNLL-2014 Shared Task on Grammatical Error Correction // Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task. — Baltimore, MD: Association for Computational Linguistics, 2014. — P. 1–14.

[12] Rozovskaya A., Roth D. Grammar Error Correction in Morphologically Rich Languages: The Case of Russian // Transactions of the Association for Computational Linguistics. — 2019. — Vol. 7. — P. 1–17.

[13] Rothe S., Mallinson J., Malmi E., Krause S., Severyn A. A Simple Recipe for Multilingual Grammatical Error Correction // Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers). — Online: Association for Computational Linguistics, 2021. — P. 702–707.

[14] Grundkiewicz R., Junczys-Dowmunt M., Heafield K. Neural Grammatical Error Correction Systems with Unsupervised Pre-training on Synthetic Data // Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications. — Florence, Italy: Association for Computational Linguistics, 2019. — P. 252–263.

[15] Omelianchuk K., Atrasevych V., Chernodub A., Skurzhanskyi O. GECToR – Grammatical Error Correction: Tag, Not Rewrite // Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications. — Seattle, WA, USA → Online: Association for Computational Linguistics, 2020. — P. 163–170.

[16] Malmi E., Krause S., Rothe S., Mirylenka D., Severyn A. Encode, Tag, Realize: High-Precision Text Editing // Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP). — Hong Kong: Association for Computational Linguistics, 2019. — P. 5054–5065.

[17] Vaswani A., Shazeer N., Parmar N., et al. Attention is All You Need // Advances in Neural Information Processing Systems. — 2017. — Vol. 30.

[18] Devlin J., Chang M.-W., Lee K., Toutanova K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding // 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. — P. 4171–4186. — DOI: 10.18653/v1/N19-1423.

[19] Liu Y., Ott M., Goyal N., Du J., Joshi M., Chen D., Levy O., Lewis M., Zettlemoyer L., Stoyanov V. RoBERTa: A Robustly Optimized BERT Pretraining Approach [Электронный ресурс]. — 2019. — Режим доступа: http://arxiv.org/abs/1907.11692 (дата обращения: 03.06.2025)

[20] Yang Z., Dai Z., Yang Y., Carbonell J., Salakhutdinov R., Le Q. V. XLNet: generalized autoregressive pretraining for language understanding // Proceedings of the 33rd International Conference on Neural Information Processing Systems. — Red Hook, NY, USA: Curran Associates Inc., 2019. — Article No. 517. — 11 p.

[21] Khabutdinov I. A., Chashchin A. V., Grabovoy A. V., Kildyakov A. S., Chekhovich U. V. RuGECToR: Rule-Based Neural Network Model for Russian Language Grammatical Error Correction // Program. Comput. Softw. — 2024. — Vol. 50, no. 4. — P. 315–321. — DOI: 10.1134/S0361768824700129.

[22] Kudo T., Richardson J. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing // In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. — Brussels, Belgium: Association for Computational Linguistics, 2018. — P. 66–71. — DOI: 10.18653/v1/D18-2012.

[23] Korobov M. Morphological Analyzer and Generator for Russian and Ukrainian Languages // Analysis of Images, Social Networks and Texts. — Springer, 2015. — Vol. 542. — P. 320–332.

[24] Open source collection of school essays [Электронный ресурс]. — Режим доступа: `https://www.kritika24.ru` (дата обращения: 07.11.2022).

[25] Open source collection of literary works [Электронный ресурс]. — Режим доступа: `https://proza.ru` (дата обращения: 07.11.2022).

[26] Trinh V.A., Rozovskaya A. New Dataset and Strong Baselines for the Grammatical Error Correction of Russian // Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021. — Online: Association for Computational Linguistics, 2021. — P. 4103–4111.

[27] Lyashevskaya O., Sharov S. Frequency Dictionary of the Modern Russian Language (based on the materials of the National Corpus of the Russian Language) [in Russian]. — Moscow: Azbukovnik, 2009. — 21 p.

[28] Kingma D., Ba J. Adam: A Method for Stochastic Optimization // International Conference on Learning Representations (ICLR). — 2015. — arXiv:1412.6980.

[29] Limisiewicz T., Balhar J., Mareček D. Tokenization Impacts Multilingual Language Modeling: Assessing Vocabulary Allocation and Overlap Across Languages // *Findings of the Association for Computational Linguistics: ACL 2023*. — Toronto, Canada: Association for Computational Linguistics, 2023. — P. 5661–5681. — DOI: 10.18653/v1/2023.findings-acl.350.

[30] Asvarov A., Grabovoy A. The Impact of Multilinguality and Tokenization on Statistical Machine Translation // *Proceedings of the 2024 Conference*. — 2024. — P. 149–157. — DOI: 10.23919/FRUCT61870.2024.10516416.

[31] Dahlmeier D., Ng H.T. Better Evaluation for Grammatical Error Correction // *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. — Montréal, Canada: Association for Computational Linguistics, 2012. — P. 568–572.

[32] Awasthi A., Sarawagi S., Goyal R., Ghosh S., Piratla V. Parallel Iterative Edit Models for Local Sequence Transduction // *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. — Hong Kong, China: Association for Computational Linguistics, 2019. — P. 4259–4269. — DOI: 10.18653/v1/D19-1435.

[33] Mizumoto T., Komachi M., Nagata M., Matsumoto Y. Mining revision log of language learning SNS for automated Japanese error

correction of second language learners // *Proceedings of the 5th International Joint Conference on Natural Language Processing*. — 2011. — P. 147–155.

[34] Rothe S., Mallinson J., Malmi E., Krause S., Severyn A. A Simple Recipe for Multilingual Grammatical Error Correction // *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. — 2021. — P. 702–707. — DOI: 10.18653/v1/2021.acl-short.89.

[35] Dahlmeier D., Ng H. T., Wu S. Building a Large Annotated Corpus of Learner English: The NUS Corpus of Learner English // *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*. — Atlanta, Georgia: Association for Computational Linguistics, 2013. — P. 22–31. — URL: aclanthology.org/W13-1703.

[36] Yannakoudakis H., Briscoe T., Medlock B. A New Dataset and Method for Automatically Grading ESOL Texts // *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. — Portland, Oregon, USA: Association for Computational Linguistics, 2011. — P. 180–189. — URL: aclanthology.org/P11-1019.

[37] Bryant C., Felice M., Andersen Ø.E., Briscoe T. The BEA-2019 Shared Task on Grammatical Error Correction // *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. — Florence, Italy: Association for Computational Linguistics, 2019. — P. 52–75. DOI: 10.18653/v1/W19-4406. URL: aclanthology.org/W19-4406.