КОМПЬЮТЕРНЫЕ НАУКИ И ИНФОРМАТИКА

Научная статья
УДК 004.89
DOI: 10.17072/1993-0550-2025-1-91-108
https://elibrary.ru/ccjape

Базовый алгоритм автоматической корректировки орфографии текстов на русском языке: разработка, оценка и перспективы

Екатерина Владимировна Исаева[1], Бехруз Зафарович Сафарбеков[2]
[1]Пермский государственный национальный исследовательский университет, г. Пермь, Россия
[2]Национальный исследовательский технологический университет "МИСИС", г. Москва, Россия
[1]ekaterinaisae@psu.ru
[2]behruzsafarbekov3@gmail.com

Аннотация. Автоматическая проверка орфографических ошибок и корректировка текстов на русском языке являются актуальной задачей в области обработки естественного языка. Целью исследования является разработка, оценка и описание программы ЭВМ для исправления орфографических ошибок с высокой точностью.

Предложенный метод основан на построчной обработке текста с использованием правил для выявления ошибок в орфографии и капитализации, а также вероятностной модели предложения слов-кандидатов для исправления ошибок. Данный алгоритм работает на уровне отдельных слов, что ограничивает его способность учитывать контекст. Для проверки качества модели использованы метрики: Precision, Recall и F1 Score. Для удобства использования и доработки программы в ее алгоритм были интегрированы автоматический анализ ошибок и формирование детализированного отчета, что позволяет выявлять сильные и слабые стороны алгоритма. Детализированное описание разработки обеспечивает воспроизводимость алгоритма и соответствует идеологии "Open-source" (Открытого программного обеспечения).

Результаты показали, что алгоритм обладает высокой точностью (Precision = 1.00), т.е. исправляет только те орфографические ошибки, которые были указаны в контрольном тексте. Однако полнота исправлений (Recall = 0.84) подчеркивает необходимость дальнейшей доработки, включая обработку контекстозависимых ошибок и работу с устойчивыми выражениями. Значение F1 Score = 0.91 подтверждает сбалансированность работы алгоритма и обосновывает его использование в качестве базовой модели корректировки текста на русском языке.

Выводы исследования подчеркивают потенциал алгоритма в задачах автоматической корректировки русскоязычного текста, а также предлагают перспективные направления для улучшения исходного кода, такие как использование n-грамм и языковых моделей.

Данная работа закладывает основу для дальнейших исследований в области автоматической корректировки текстов на русском языке.

Ключевые слова: *орфографические ошибки; грамматические ошибки; русский язык; автоматическая корректировка текста; обработка естественного языка; точность, полнота; F1 Score*

Для цитирования: *Исаева Е.В., Сафарбеков Б.З.* Базовый алгоритм автоматической корректировки орфографии текстов на русском языке: разработка, оценка и перспективы // Вестник Пермского университета. Математика. Механика. Информатика. 2025. Вып. 1 (68). С. 91–108. DOI: 10.17072/ 1993-0550-2025-1- 91-108. https://elibrary.ru/ccjape.

Статья поступила в редакцию 25.12.2025; одобрена после рецензирования 17.02.2025; принята к публикации 20.03.2025.

COMPUTER SCIENCE

Research article

Basic Algorithm for Automatic Spelling Correction of Russian Texts: Development, Evaluation and Prospects

Ekaterina V. Isaeva[1], Behruz Z. Safarbekov[2]
[1]Perm State University, Perm, Russia
[2]National University of Science and Technology "MISIS"
[1]ekaterinaisae@psu.ru
[2]behruzsafarbekov3@gmail.com

Abstract. Automatic spelling check and correction of texts in Russian is an urgent task in the field of natural language processing. Our research is aimed at developing, evaluating, and describing a computer programme for correcting spelling errors with high accuracy.

The proposed method is based on line-by-line text processing using rules for spelling and capitalisation accuracy and a probabilistic model for proposing candidate words for error correction. Our algorithm operates at the level of individual words, which limits its ability to take context into account. The metrics used to test the quality of the model are Precision, Recall, and F1 Score. For ease of use and program refinement, we integrated automated error analysis and detailed report generation to identify the strengths and weaknesses of the algorithm. The detailed development description ensures the reproducibility of the algorithm and is in line with the Open-source ideology.

The results showed that the algorithm has high Precision = 1.00, i.e., it corrects only those spelling errors that were specified in the reference text. However, the Recall = 0.84 emphasises the need for further refinement, including handling context-dependent errors and processing sye expressions. The F1 Score = 0.91 confirms the balanced performance of the algorithm and justifies its use as a basic model of text correction in Russian.

The conclusions of the study emphasise the potential of the algorithm in the tasks of automatic correction of Russian-language text, and suggest prospective areas for improving the source code, such as the use of n-grams and language models. This work lays the foundation for further research in the field of automatic correction of Russian-language texts.

Keywords: *spelling errors; grammatical errors; Russian language; automatic text correction; natural language processing, accuracy; completeness; F1 Score*

For citation: Isaeva, E. V. and Safarbekov, B. Z. (2025), "Basic Algorithm for Automatic Spelling Correction of Russian Texts: Development, Evaluation and Prospects", *Bulletin of Perm University. Mathematics. Mechanics. Computer Science*, no. 1(68), pp. 91-108. (In Russ.). DOI: 10.17072/1993-0550-2025-1-91-108. https://elibrary.ru/ccjape.

The article was submitted 25.12.2025; approved after reviewing 17.02.2025; accepted for publication 20.03.2025

Введение

Корректировка орфографии имеет важное значение для улучшения качества текста в различных сферах, включая научную, образовательную и коммерческую деятельность. Инструменты для корректировки орфографии повышают производительность труда, улучшают знание языка и повышают уверенность в письменной коммуникации. В связи с этим проблема автоматической корректировки текста уже несколько десятилетий не утрачивает свою актуальность и до сих пор находится в развитии. Сложность проверки орфографии варьируется в зависимости от орфографической структуры языка. В то время как программы проверки орфографии в английском языке достаточно хорошо зарекомендовали себя, языки с развитой морфологической системой, такие как русский, требуют специальных подходов для корректировки орфографических ошибок. К ним относятся статистические модели (n-граммы, байесовские методы) для оценки вероятности появления слов в контексте [1], методы логического сжатия словарей для оптимизации хранения и поиска слов (префиксные деревья, булевы функции) [2], и глубокие нейросетевые модели (Bi-LSTM, трансформеры) для контекстозависимого исправления ошибок [3]. Комплексный подход на основе этих методов позволит повысить точность обнаружения и исправления ошибок в морфологически сложных языках.

В статье представлено описание разработки базового метода проверки орфографии в русском языке, который может послужить основой для надстройки функций проверки контекстозависимой проверки грамматических, синтаксических и стилистических ошибок. Разработанный метод показал достаточную эффективность для корректировки технических текстов, что продемонстрировано в данной статье на примере текста о беспилотных авиационных системах.

1. Обзор методов орфографической корректировки текста

В данном разделе рассмотрены основные подходы к автоматической корректировке орфографии, которые применяются в различных языках.

Традиционным подходом к решению задачи корректировки орфографии является обращение к методам на основе лексикологических и грамматических правил. В одной из ранних работ (1997 г.) предлагается автоматическое обучение лингвистическим правилам для корректировки орфографии, при этом подчеркивается важность простоты и доступности изложения: приобретенные знания хранятся в виде небольшого набора простых правил, что позволяет наилучшим образом использовать человеческую интуицию для понимания и улучшения знаний, накопленных системой [4]. В другой статье предлагается метод автоматического морфемного анализа, проверки написания заданного слова по правилам языка и имеющимся словарям и корректировки неверно написанных слов [5]. В более поздних работах используется метод n-грамм для повышения точности в обнаружении и исправлении ошибок. В работе [6] описывается эффективный метод орфографической проверки, использующий лексикон Microsoft и стандартные наборы слов с орфографическими ошибками на английском языке для обнаружения и замены неправильно написанных слов. Рассмотренные методы просты в реализации, но имеют ограниченную эффективность при обработке сложных грамматических конструкций русского языка и контекстозависимых ошибок.

В основе статистических подходов к корректировке орфографии в текстах на разных языках лежат вероятностные модели, показывающие свою эффективность для языков с богатой омонимией, таких как китайский и русский. Вероятностные модели включают в себя такие компоненты, как модули подстановок и тематического языкового моделирования. Тематические языковые модели могут передавать более обширную семантическую информацию из строки слов (символов), тогда как обычные языковые n-граммные модели могут сохранять только информацию о локальных закономерностях [7].

В английском языке были предложены подходы, основанные на байесовской теории принятия решений и расстоянии Дамерау–Левенштейна [8].

Интересный подход предлагается в одной из ранних работ [9]. В основе данного подхода лежит вероятностная модель генерации искаженных слов из правильных. Программа может удалять или вставлять символы в слово, а также заменять один или несколько символов другими. Далее, рекурсивным образом программа вычисляет правильное слово из искаженного. Авторы подчеркивают высокую эффективность работы алгоритма, в т. ч. при работе с искаженными словами, полученными из относительно коротких слов длиной менее 6 символов.

Вероятностные языковые модели, обученные на больших текстовых корпусах, становятся все более популярной альтернативой традиционным подходам, основанным на грамматике. Эти модели находят применение в различных задачах обработки естественного языка, включая распознавание речи, оптическое распознавание символов и корректировку орфографии. Исследования сосредоточены на решении таких задач, как сглаживание моделей n-грамм, индукция статистических грамматик и двуязычное выравнивание предложений, с акцентом на решение проблемы разреженных данных и выявления скрытых структур [10].

В последних исследованиях описываются решения задачи орфографической корректировки текста с помощью моделей трансформеров. Например, разработчики представили объединенную модель зашумленного канала с языковыми моделями для корректировки грамматических ошибок с использованием ограниченного количества ресурсов. Полученная модель генерирует наборы данных с ошибками на основе истории правок статей Википедии. Используются предобученная модель BERT, настроенная на конкретные типы ошибок, и GPT-2, оперирующая предыдущими предложениями как контекстом. Оптимальные комбинации корректировок определяются с помощью лучевого поиска (beam search) качественного метода сэмплирования, при котором на каждом шаге корректировки текста выбирается несколько вероятных токенов, разветвляются пути генерации и получается несколько вариантов сгенерированного текста. Такой подход не лишен своих недочетов, связанных с повышенными вычислительными требованиями, ограничениями в наборе ошибок, которые встречаются в истории правок Википедии, а лучевой поиск может стать причиной усугубления ошибок [11].

Предобученная модель BERT с включением контекстной информации используется для решения дефицита аннотированных данных, необходимых для обучения больших языковых моделей. Задача решается в два этапа. На первом этапе осуществляется попытка определить, к какому участку оригинального текста будет применена правка. Для этого формулируется задача маркировки последовательности, в которой лексемы маркируются одной из следующих меток {оставить, заменить, вставить, удалить}. Для второго этапа (исправления) используется предварительно обученная модель типа BERT. Метки, полученные на этапе идентификации ошибок, определяют маскировку входов (где маскируются все лексемы, помеченные метками замены или вставки), и для каждой замаскированной лексемы определяются кандидаты правки. Сложности данного подхода заключаются в ограничениях в длине маркировок ошибок, избыточных правках, потере и искажении потере исходной информации [12].

Современные подходы к автоматической корректировке текста сосредоточены на использовании продвинутых языковых моделей и сложных архитектур нейронных сетей, таких как Sequence-to-Sequence и Sequence Tagging, как, например, модели GECToR [13] и SAGE (Spell checking via Augmentation and Generative distribution Emulation) [14], показавшие высокую эффективность в корректировке текста.

Статистические, вероятностные и современные нейросетевые модели хорошо справляются с корректировкой текста, но требуют больших объемов данных для обучения, что вызывает сложности для реализации с учетом дефицита размеченных, особенно специализированных технических, медицинских и др. данных на русском языке для данной задачи в доступных на открытых репозиториях, таких как Kaggle [15] и Hugging Face [16]. Кроме того, генеративные модели, сфокусированные на корректировке целых предложений, а не отдельных слов, могут вызывать проблемы интерпретации, генерируя исправленный текст, но не объясняя, почему было предложено то или иное исправление.

2. Результаты

В представленной разработке используется библиотека **LanguageTool**, используемая для корректировки орфографии в текстовом редакторе OpenOffice, которая сочетает в себе как правила, так и вероятностные модели, и предоставляет API для проверки текстов на наличие ошибок [17]. Библиотека поддерживает множество языков, включая русский, и использует набор грамматических правил и лексикографическую базу для выявления ошибок и предложения исправлений.

При сравнении LanguageTool с аналогичными инструментами было выявлено, что в отличие от морфологического анализатора Hunspell [18, 19], работающего на основе словарей, LanguageTool использует не только статические словари, но и систему правил.

Другой инструмент, SymSpell, скоростной алгоритм орфографической коррекции на основе n-грамм и расстояния Дамерау–Левенштейна [20, 21], работает только на уровне отдельных слов и используется в основном для английского языка.

Важными преимуществами использования библиотеки LanguageTool является ее доступность для использования и модификации, что соответствует идеологии Отрытого кода (Open-source), расширяемость, что позволяет добавлять новые правила, и обширность методов для корректировки орфографии, грамматики и стилистики текста на разных языках.

Предложенный в данной статье подход позволяет автоматически предлагать варианты исправления ошибок, что повышает точность корректировки. В работе используется метод *tool.check*, предоставляемый *language_tool_python*. Данный метод выполняет анализ текста, выявляя потенциальные ошибки, и возвращает список объектов, содержащих тип ошибки (орфографическая, грамматическая и т. д.), диапазон символов, в котором найдена ошибка, возможные исправления в порядке убывания вероятности. Данный метод выбран в качестве основного, поскольку он позволяет автоматически анализировать текст без необходимости вручную формулировать правила исправления ошибок. Это делает его удобным инструментом для базовой орфографической проверки, в частности, для исправления ошибок в отдельных словах и ошибочного употребления прописных и строчных букв.

Несмотря на свою простоту, предложенный метод имеет ряд преимуществ по сравнению с продвинутыми языковыми моделями: эффективность на уровне отдельных слов, предъявляет минимальные требования к ресурсам, прост для интерпретации, может быть адаптирован под конкретные задачи, не требует обучения на больших данных.

Таким образом, разработанный алгоритм позиционируется как базовый для корректировки орфографии в русских текстах. Функциональность алгоритма может быть расширена с помощью добавления других методов LanguageTool, которые позволяют не только исправлять орфографические ошибки, но и выявлять грамматические и синтаксические неточности, анализировать смысловую структуру текста и улучшать его стилистическое качество [22]. Разработанный алгоритм может быть использован на этапе подготовки текста для других, вычислительно более сложных методов, таких как языковые модели и логистическая регрессия [23].

Программа[1] нацелена на автоматическую корректировку текстов для устранения орфографических и ошибок капитализации предложений на русском языке. Алгоритм работы программы включает следующие блоки (рис. 1).

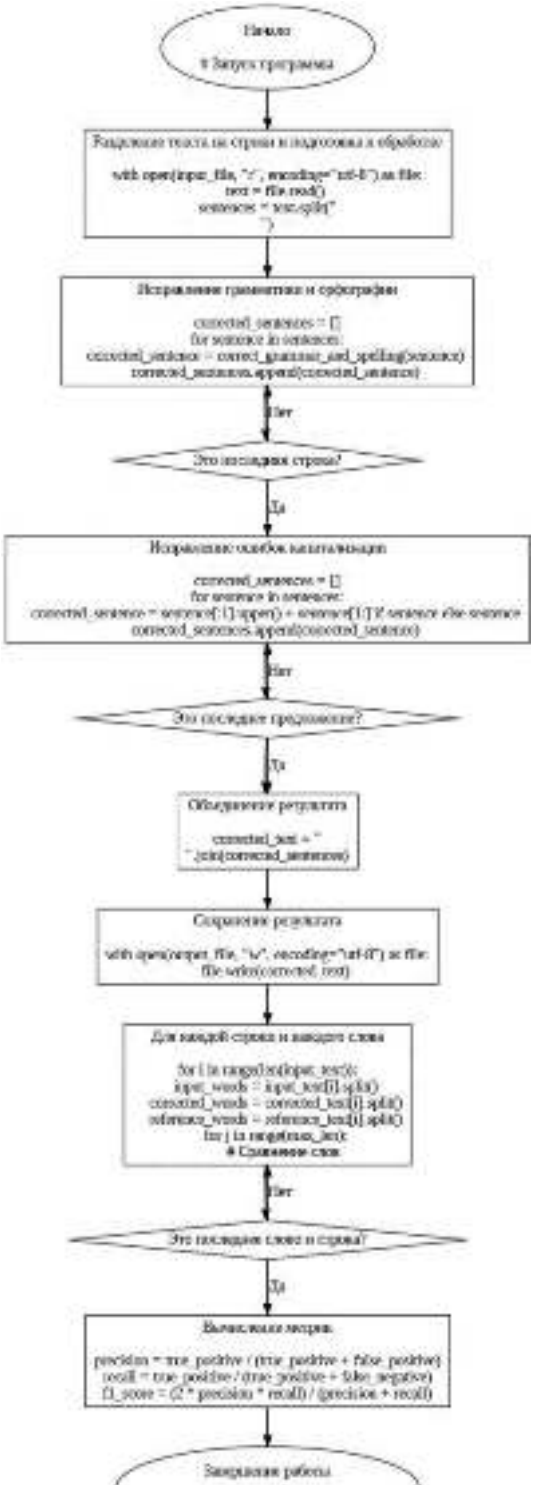

Рис. 1. *Схематическое представление алгоритма программы для корректировки орфографических ошибок на русском языке*

Блок 1. Начало: запуск программы, установка библиотек и инициализация инструмента для проверки ошибок. Для проверки орфографических ошибок используется библиотека *language_tool_python*, которая предоставляет доступ к инструменту *LanguAageTool*. В момент написания данной статьи актуальной является версия *language_tool_python-2.8.1*. Для работы с русским языком выполняется соответствующая языковая инициализация:

tool = language_tool_python.LanguageTool ('ru-RU')

Блок 2. Разделение текста на строки и подготовка к обработке: функция *process_text_file*. Следующим блоком программы является обработка текста из файла. Функция *process_text_file* предназначена для автоматической обработки текста, хранящегося в файле, с последующим сохранением результата в другой файл. Она объединяет этапы чтения текста, исправления ошибок и капитализации, обеспечивая полный цикл обработки.

В данной программе настроено считывание исходного текста из файла с расширением *.txt* (например, input.txt) для последующей обработки. Для корректного считывания текста на русском языке используется режим чтения ('r') и кодировка *UTF-8*:

with open(input_file, 'r', encoding='utf-8') as file:
text = file.read()

На данном этапе весь текст файла загружается в строковую переменную *text*.

Далее для построчной обработки разделяем считанный текст на строки. Такой подход к считыванию текста полезен по ряду причин. Во-первых, многие тексты, особенно статьи и документы, могут содержать абзацы, разделенные пустыми строками, строки, разделенные символами переноса строки (\n), например для маркировки заголовков, списков или отдельных предложений и т. д. Если такой текст обработать как одну большую строку, то есть риск нарушения его структуры: разные абзацы, списки и заголовки могут слиться в единый блок.

[1]Проект (исходный код, примеры текстового файла с ошибками и файла эталонного текста) размещен в открытом доступе на репозитории GitHub https://github.com/Kate-PSU/Spelling_cheque_RUS.

Построковая обработка текста позволяет сохранить особенности исходного форматирования. Кроме того, разделение текста на строки упрощает обработку текста, позволяя обрабатывать каждую строку отдельно, исправляя ошибки в тех строках, в которых они присутствуют, пропуская строки, не требующие изменений. Данный способ способствует повышению читаемости обработанного текста: результат можно собрать обратно в текст, сохраняя исходное форматирование. Также отметим такие плюсы построчной обработки текста, как локализация и точечная обработка ошибок, и гибкость обработки (возможность добавления дополнительных этапов обработки для отдельного прохождения по заголовкам, спискам, таблицам и др., так как к ним могут применяться специфические правила орфографии).

В данном блоке определяются переменные: *input_file*, указывающая имя входного файла, который содержит исходный текст для обработки (здесь *input.txt*), и *output_file*, указывающая имя выходного файла, в который записывается исправленный текст (здесь *corrected_output.txt*):

input_file = 'input.txt' # Имя входного файла

output_file = 'corrected_output.txt' # Имя выходного файла

Далее вызывается функция *process_text_file*, которая выполняет весь цикл обработки текста: чтение исходного файла, исправление орфографических ошибок и капитализации и сохранение обработанного текста в новый файл. Функция принимает два параметра: *input_file* и *output_file*:

process_text_file(input_file, output_file)

После успешного выполнения функции в консоль выводится сообщение: *Обработка завершена. Исправленный текст сохранен в файл: corrected_output.txt*. Это подтверждает, что файл обработан и результат успешно сохранен.

Функция *process_text_file* оборачивается в блок *try...except* для обработки возможных ошибок, таких как невозможность открыть файл и ошибок при записи в файл: *except Exception as e: print (f"Ошибка при обработке файла: {e}").*

При возникновении ошибки выводится сообщение с ее описанием. Например, если в текущую сессию не загружен файл input.txt, или загружен одноименный файл с другим расширением, то программа выдает ошибку: *Ошибка при обработке файла: [Errno 2] No such file or directory: 'input.txt'*

На рис. 2 приведен пример исходного кода функции обработки текста:

```
# Функция для обработки текста из файла
def process_text_file(input_file, output_file):
    try:
        # Открываем файл для чтения
        with open(input_file, 'r', encoding='utf-8') as file:
            text = file.read()

        # Корректируем текст
        corrected_sentences = []
        for sentence in text.split('\n'):
            corrected_sentence = correct_grammar_and_spelling(sentence)
            corrected_sentences.append(corrected_sentence)

        # Объединяем предложения в текст
        corrected_text = '\n'.join(corrected_sentences)

        # Исправляем капитализацию в тексте
        fully_corrected_text = correct_capitalization(corrected_text)

        # Сохраняем исправленный текст в новый файл
        with open(output_file, 'w', encoding='utf-8') as file:
            file.write(fully_corrected_text)

        print(f"Обработка завершена. Исправленный текст сохранен в файл: {output_file}")

    except Exception as e:
        print(f"Ошибка при обработке файла: {e}")
```

Рис. 2. *Функция обработки текста из файла (пример исходного кода)*

Блок 3. Исправление грамматики и орфографии: функция *correct_grammar_and_spelling*. После разбиения на строки каждая строка передается на вход функции *correct_grammar_and_spelling* для исправления орфографических ошибок. Каждая строка текста обрабатывается функцией, которая вызывает инструмент *LanguageTool* для проверки строки, анализирует список обнаруженных ошибок, исправляет текст, заменяя фрагменты с ошибками на предложенные кандидаты корректировок. Логика работы функции *correct_grammar_and_spelling* может быть представлена следующим образом:

1) Анализ текста на наличие орфографических ошибок. Функция *correct_grammar_and_spelling* принимает предложение (*sentence*) в качестве входного параметра. Далее, помощью метода *tool.check* анализируется текст и возвращается список ошибок (*matches*). Каждый элемент списка matches содержит информацию о найденной ошибке, а именно местоположение ошибки (начальная и конечная позиции в строке), тип ошибки (грамматическая, орфографическая и т. д.), кандидаты корректировок (список предложений).

2) Обработка ошибок. Для корректного исправления ошибок они обрабатываются в обратном порядке (с конца предложения к началу). Это предотвращает смещение индексов текста при внесении исправлений. Для каждой найденной ошибки функция проверяет наличие кандидатов для замены (if match.replacements), извлекает начальную (start) и конечную (end) позиции ошибки, заменяет часть текста с орфографической ошибкой на первый вариант корректировки из списка кандидатов (match.replacements [0]). Текст с исправлениями последовательно обновляется, создавая новую строку *corrected_sentence*.

3) Возврат результата. После обработки всех ошибок функция возвращает исправленное предложение (рис. 3).

```
# Функция для исправления грамматических и орфографических ошибок в предложении
def correct_grammar_and_spelling(sentence):
    matches = tool.check(sentence)
    corrected_sentence = sentence
    for match in reversed(matches):  # Обрабатываем ошибки в обратном порядке
        if match.replacements:  # Проверяем, есть ли замены для ошибки
            start = match.offset
            end = match.offset + match.errorLength
            corrected_sentence = corrected_sentence[:start] + match.replacements[0]\
             + corrected_sentence[end:]
    return corrected_sentence
```

Рис. 3. *Функция для исправления орфографических ошибок (пример исходного кода)*

По нашему наблюдению, использованный метод (*tool.check*) в той конфигурации, которая используется в представленной разработке, эффективен только при работе с орфографическими ошибками, но не корректирует грамматические ошибки, такие, например, как согласование числа существительного и глагола или прилагательного и существительного, управление глагола, согласование падежей прилагательного и существительного и др. В связи с этим, данный алгоритм описывается как базовый метод для корректировки орфографических ошибок.

Далее приводится пример работы функции *correct_grammar_and_spelling* для корректировки профессионально-ориентированного текста из области беспилотных авиационных систем. На вход функции подается предложение с ошибками:

Пример 1 (исходный текст): *Например, с помощю беспилотников можно бистро обнаружить нарушение экалогических норм.*

Шаги корректировки текста:

1) LanguageTool обнаруживает следующие ошибки:
   a. *помощю*
   b. бистро
   c. *екалогических*
2) Функция *correct_grammar_and_spelling* исправляет ошибки, начиная с конца предложения:
   a. екалогических → экологических
   b. бистро → быстро
   c. помощю → помощь
3) Исправленное предложение:
   a. *Например, с помощь беспилотников можно быстро обнаружить нарушения экологических норм.*

Пример иллюстрирует, что орфографическая ошибка в слове *помощу* устранена, однако, появилась новая грамматическая ошибка – ошибка управления (несогласованность падежа существительного и предлога): *с помощь* (правильный вариант – *с помощью*).

Блок 4. Исправление ошибок капитализации. Функция *correct_capitalization* разработана для исправления ошибок капитализации в тексте. Она автоматически преобразует первую букву каждого предложения в заглавную, сохраняя общую структуру текста. Алгоритм работы функции включает следующие этапы:

1) Разделение текста на предложения. Текст разбивается на отдельные предложения по разделителю ". " (точка с пробелом), что позволяет отдельно работать с каждым предложением:

sentences = text.split('. ').

Пример 2.

Исходный текст:

также бас приминяются в сельском хазяйстве для оптимизацыи расходов и повышения эфективности. однак, их использование требует учёта вапросов безопастности и конфеденцыальности даных.

Текст после разделения:

sentences = [
" *также бас приминяются в сельском хазяйстве для оптимизацыи расходов и повышения эфективности*",
" *однак, их использование требует учёта вапросов безопастности и конфеденцыальности даных* "
]

2) Корректировка первой буквы предложения. В каждом предложении первая буква преобразуется в заглавную с помощью метода *.upper():*

corrected_sentence = sentence[:1].upper() + sentence[1:] if sentence else sentence

Оставшаяся часть предложения сохраняется неизменной.

Пример 3 (исправленный текст):
Также бас приминяются в сельском хазяйстве для оптимизацыи расходов и повышения эфективности.
Однак, их использование требует учёта вапросов безопастности и конфеденцыальности даных.

В примере 3 проиллюстрировано только действие функции *correct_capitalization*, другие ошибки остались без изменения.

После этого все предложения объединяются обратно в текст, используя ". " в качестве разделителя: *return '. '.join(corrected_sentences)*

Рисунок 4 иллюстрирует исходный код функции капитализации предложений текста.

```
# Функция для исправления ошибок капитализации
def correct_capitalization(text):
    # Разбиваем текст на предложения
    sentences = text.split('. ')
    corrected_sentences = []
    for sentence in sentences:
        # Исправляем первую букву предложения
        corrected_sentence = sentence[:1].upper() + sentence[1:] if sentence else sentence
        corrected_sentences.append(corrected_sentence)
    return '. '.join(corrected_sentences)
```

Рис. 4. *Функция для исправления ошибок капитализации (пример исходного кода)*

Блок 5. Объединение результата. Исправленные строки объединяются в единый текст, разделенный символами новой строки (\n), для сохранения исходной структуры: *corrected_text = '\n'.join(corrected_sentences).*

Объединенный текст передается в функцию *correct_capitalization* для исправления ошибок капитализации. Функция обрабатывает каждое предложение, делая первую букву заглавной: *fully_corrected_text = correct_capitalization(corrected_text).*

Блок 6. Сохранение результата. Исправленный текст записывается в новый файл в режиме записи (*'w'*) с использованием кодировки *UTF-8*:

```
with open(output_file, 'w', encoding='utf-8') as file:
    file.write(fully_corrected_text)
```

Блоки 7–8. Сравнение слов и вычисление метрик: функция compare_files_and_calculate_metrics, данный раздел программы является дополнительным. Он позволяет провести автоматизированную оценку работы программы: автоматическое вычисление метрик эффективности корректировки текста и "ручную" проверку результатов по детализированному описанию ошибок.

Оценка качества работы алгоритма проводится по метрикам: *Precision* (точность), *Recall* (полнота) и *F1 Score* (средневзвешенная гармоническая метрика).

Для оценки качества алгоритма корректировки текста была разработана функция *compare_files_and_calculate_metrics*, которая сравнивает три текстовых файла:

1) Исходный файл (с ошибками) – текст до обработки (*input.txt*).

2) Файл с исправлениями – текст, полученный после работы алгоритма (*corrected_output.txt*).

3) Контрольный файл – эталонный текст без ошибок (*reference.txt*).

Примеры наполнения текстовых файлов представлены в Приложении 1.

Функция *compare_files_and_calculate_metrics* вычисляет метрики:

- Precision (доля правильно исправленных ошибок среди всех внесенных исправлений):

(1) $Precision = \frac{TP}{TP+FP}$

- Recall (доля правильно исправленных ошибок среди всех ошибок, которые необходимо было исправить):

(2) $Recall = \frac{TP}{TP+FN}$

- F1Score (гармоническое среднее между долей правильно исправленных ошибок среди всех внесенных исправлений и долей правильно исправленных ошибок среди всех ошибок, которые необходимо было исправить):

(3) $F1\ Score = 2 * \frac{\text{Precision} * Recall}{\text{Precision} + Recall}$

где:

- True Positive (TP) – исправленное слово совпадает с эталонным;
- False Positive (FP) – исправленное слово не совпадает с эталонным, при этом исходное слово было правильным;
- False Negative (FN) – исправленное слово не совпадает с эталонным, при этом исходное слово было неверным.

Алгоритм работы функции *compare_files_and_calculate_metrics* работает следующим образом:

1) чтение файлов – все три файла загружаются построчно, каждая строка текста разделяется на слова для построчного и пословного анализа;
2) сравнение слов – для каждой строки алгоритм анализирует слова из исходного, исправленного и контрольного файлов, определяются параметры для вычисления метрик.

Для возможности экспертной оценки работы программы с помощью *Pandas*, библиотеки с открытым исходным кодом, предоставляющей простые в использовании структуры данных и инструменты анализа данных [24], была сформирована таблица (датафрейм), в который записывались следующие данные:

- *Исходное слово* – слово из файла с ошибками.
- *Исправленное слово* – слово, предложенное алгоритмом.
- *Ожидаемое слово* – слово из контрольного файла.
- *Статус* – результат анализа (*True Positive, False Positive, False Negative*).

Рисунок 5 демонстрирует пример исходного кода функции сравнения двух текстовых файлов: до обработки и после исправления орфографических ошибок с помощью представленной программы и вычисления метрик качества работы программы.

```
import pandas as pd
def compare_files_and_calculate_metrics(input_file, corrected_file, reference_file):
    try:
        # Чтение файлов
        with open(input_file, 'r', encoding='utf-8') as f:
            input_text = f.read().splitlines()
        with open(corrected_file, 'r', encoding='utf-8') as f:
            corrected_text = f.read().splitlines()
        with open(reference_file, 'r', encoding='utf-8') as f:
            reference_text = f.read().splitlines()
        # Проверка длины файлов
        if not (len(input_text) == len(corrected_text) == len(reference_text)):
            raise ValueError("Файлы должны содержать одинаковое количество строк для корректного сравнения.")
        # Инициализация счетчиков
        true_positive = 0  # Правильно исправленные ошибки
        false_positive = 0  # Ложные исправления (там, где не было ошибки)
        false_negative = 0  # Пропущенные ошибки
        # Список для формирования датафрейма
        error_details = []
        # Сравнение строк на уровне слов
        for i in range(len(input_text)):
            input_words = input_text[i].split()
            corrected_words = corrected_text[i].split()
            reference_words = reference_text[i].split()
            max_len = max(len(input_words), len(corrected_words), len(reference_words))
            for j in range(max_len):
                input_word = input_words[j] if j < len(input_words) else ''
                corrected_word = corrected_words[j] if j < len(corrected_words) else ''
                reference_word = reference_words[j] if j < len(reference_words) else ''
                # Если ошибка исправлена правильно
                if input_word != reference_word and corrected_word == reference_word:
                    true_positive += 1
                    error_details.append({
                        'Исходное слово': input_word,
                        'Исправленное слово': corrected_word,
                        'Ожидаемое слово': reference_word,
                        'Статус': 'True Positive'
                    })
                # Если алгоритм внёс исправление, но ошибка не было
                elif input_word == reference_word and corrected_word != reference_word:
                    false_positive += 1
                    error_details.append({
                        'Исходное слово': input_word,
                        'Исправленное слово': corrected_word,
                        'Ожидаемое слово': reference_word,
                        'Статус': 'False Positive'
                    })
                # Если ошибка не исправлена
                elif input_word != reference_word and corrected_word != reference_word:
                    false_negative += 1
                    error_details.append({
                        'Исходное слово': input_word,
                        'Исправленное слово': corrected_word,
                        'Ожидаемое слово': reference_word,
                        'Статус': 'False Negative'
                    })
        # Вычисление метрик
        precision = true_positive / (true_positive + false_positive) if (true_positive + false_positive) > 0 else 0
        recall = true_positive / (true_positive + false_negative) if (true_positive + false_negative) > 0 else 0
        f1_score = (2 * precision * recall) / (precision + recall) if (precision + recall) > 0 else 0
        # Создание датафрейма
        error_df = pd.DataFrame(error_details)
        return precision, recall, f1_score, error_df
    except Exception as e:
        print(f"Ошибка при вычислении метрик: {e}")
        return 0, 0, 0, pd.DataFrame()
```

Рис. 5. *Функция сравнения файлов и вычисления метрик (пример исходного кода)*

Далее производится загрузка исходного, скорректированного и эталонного файлов, к которым применяется функция *compare_files_and_calculate_metrics,* выводятся показатели метрик и детализация ошибок в формате датафрейма (рис. 6).

```
# Пример использования
input_file = 'input.txt'  # Исходный файл с ошибками
corrected_file = 'corrected_output.txt'  # Файл с исправлениями
reference_file = 'reference.txt'  # Контрольный файл без ошибок

precision, recall, f1_score, error_df = compare_files_and_calculate_metrics(
    input_file, corrected_file, reference_file
    )

print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1_score:.2f}")

# Вывод датафрейма
if not error_df.empty:
    print("\nДетализация ошибок:")
    print(error_df)
```

Рис. 6. *Сравнение исходного, обработанного и эталонного файлов и вывод метрик (пример исходного кода)*

В проведенном эксперименте были полученные следующие данные (см. таблицу):

Таблица. *Детализация ошибок алгоритма исправления текста: исходные, исправленные и ожидаемые слова с указанием статуса*

Исходное слово	Исправленное слово	Исправленное слово	Ожидаемое слово
беспилотные	Беспилотные	Беспилотные	Беспилотные
(бас)	(бас)	(бас)	(БАС)
нахотят	находят	находят	находят
приминение	применение	применение	применение
используються	используются	используются	используются
например,	Например,	Например,	Например,
помощю	помощь	помощь	помощью
бистро	бистро	бистро	быстро
экалогических	экологических	экологических	экологических
также	Также	Также	Также
приминяются	применяются	применяются	применяются
хазяйстве	хозяйстве	хозяйстве	хозяйстве
оптимизацыи	оптимизации	оптимизации	оптимизации
эфективности.	эффективности.	эффективности.	эффективности.
однак,	Однако,	Однако,	Однако,
вапросов	вопросов	вопросов	вопросов
безопастности	безопасности	безопасности	безопасности
конфеденцыальности	конфиденциальности	конфиденциальности	конфиденциальности
даных.	данных.	данных.	данных.

На основе этих данных были получены следующие метрики качества:

- *Precision: 1.00*
- *Recall: 0.84*
- *F1 Score: 0.91*

Сформированная таблица ошибок позволяет детально проанализировать каждый случай корректировки ошибок, выделяя успешные исправления и упущения. Отметим успешные показатели работы программы. Высокая точность (*Precision = 1.00*) свиде-

тельствует о том, что в приведенном примере алгоритм вносит изменения только в местах, где это необходимо. Большинство ошибок исправлены правильно, что выражается в высокой *F1 Score* (*0.91*).

В качестве недочетов отметим пропущенные ошибки (*False Negative*) – выявленные слабые места алгоритма: например, ошибки в написании аббревиатур ("(бас)" → "(БАС)") и грамматические ошибки или ошибки управления ("*с помощю*" → "*с помощью*"), которые не были исправлены или были исправлены не корректно.

4. Обсуждение результатов

В ходе апробации алгоритм продемонстрировал высокую точность в исправлении орфографических ошибок в тексте, что подтверждается значением соответствующей *Precision = 1.00* (100 %). Это означает, что все исправления, внесенные алгоритмом, были корректными и соответствовали контрольному тексту. Высокое значение меры *F1 Score = 0.91* (91 %) указывает на хороший баланс между точностью и полнотой исправлений.

Кроме того, сформированная таблица ошибок позволяет детально анализировать результаты работы программы. Такая детализация особенно полезна для выявления проблемных областей и целенаправленного улучшения алгоритма.

Несмотря на высокую точность на тестовом образце текста, результаты выявили ряд ограничений, которые требуют внимания. Во-первых, отметим недостаточную полноту исправлений. Относительно невысокое значение метрики *Recall = 0.84* (84 %) говорит о том, что часть ошибок осталась не исправленной. Это связано с тем, что алгоритм не всегда распознает сложные или контекстозависимые ошибки, например: ошибка в слове *помощю* была исправлена на не корректный вариант *помощь*, который не согласуется с предлогом *с*. Ошибка в аббревиатуре (бас) осталась необработанной (не была применена капитализация). Обработка подобных ошибок требует добавления более сложных блоков контекстного анализа текста. Алгоритм работает на уровне отдельных слов, не использует n-граммы и не учитывает контекстные связи между словами. Это приводит к пропущенным ошибкам, особенно в устойчивых словосочетаниях или фразах. Например, слово "бистро" не было исправлено на "быстро" из-за отсутствия анализа зависимости между словами в предложении.

С учетом выявленных недочетов можно отметить перспективы дальнейшей оптимизации алгоритма: расширение словарей и правил исправлений (обработка аббревиатур, устойчивых выражений и n-граммов), интеграция модулей работы с морфологией (согласование падежей, числа, рода, управления и др.) и синтаксисом (разделение предложений не только по точке (.), но и другим знакам препинания, проверка корректности пунктуации).

Заключение

Разработанный алгоритм автоматической корректировки текста продемонстрировал высокую точность в исправлении базовых орфографических ошибок и способность к обработке ошибок капитализации. Значение метрики *Precision = 1.00* (100 %) свидетельствует о том, что все предложенные программой исправления были корректными, а *F1 Score = 0.91* (91 %) подтверждает высокий баланс между точностью и полнотой исправлений. Эти результаты показывают, что разработанная программа способна эффективно решать задачу проверки и корректировки орфографии текста на русском языке в условиях ограниченного набора правил.

Однако экспертный анализ работы алгоритма выявил его текущие ограничения, такие как неспособность обрабатывать контекстозависимые ошибки, например, "с по-

мощью" → "с помощью" или "(бас)" → "(БАС)". Основной причиной этих недочетов является работа алгоритма на уровне отдельных слов без учёта контекста и зависимости между словами.

Дальнейшая работа будет направлена на интеграцию более сложных методов, таких как анализ n-грамм, контексто-зависимых языковых моделей и дополнительных правил для обработки аббревиатур и устойчивых выражений. Это позволит повысить полноту исправлений (*Recall*) и адаптировать алгоритм к более сложным текстам, содержащим не только орфографические, но и грамматические, синтаксические и стилистические ошибки. Таким образом, предлагаемый алгоритм представляет собой основу для дальнейших исследований и развития методов автоматической корректировки текста на русском языке.

Приложение
Пример результатов работы программы

1) Исходный файл:

беспилотные авиационные системы (БАС) находят широкое применение в гражданских целях. Они используются для наблюдения за состоянием окружающей среды, контроля за лесными пожарами и мониторинга разливов рек. например, с помощью беспилотников можно бистро обнаружить нарушения экологических норм. также бас применяются в сельском хозяйстве для оптимизации расходов и повышения эффективности. однако, их использование требует учета вопросов безопасности и конфиденциальности данных.

2) Файл с исправлениями:

беспилотные авиационные системы (БАС) находят широкое применение в гражданских целях. Они используются для наблюдения за состоянием окружающей среды, контроля за лесными пожарами и мониторинга разливов рек. Например, с помощь беспилотников можно бистро обнаружить нарушения экологических норм. Также бас применяются в сельском хозяйстве для оптимизации расходов и повышения эффективности. Однако, их использование требует учёта вопросов безопасности и конфиденциальности данных.

3) Контрольный файл:

беспилотные авиационные системы (БАС) находят широкое применение в гражданских целях. Они используются для наблюдения за состоянием окружающей среды, контроля за лесными пожарами и мониторинга разливов рек. Например, с помощью беспилотников можно быстро обнаружить нарушения экологических норм. Также бас применяются в сельском хозяйстве для оптимизации расходов и повышения эффективности. Однако, их использование требует учёта вопросов безопасности и конфиденциальности данных.

Список источников

1. *Zukarnain N. et al.* Spelling Checker Algorithm Methods for Many Languages // 2019 International Conference on Information Management and Technology (ICIMTech). IEEE, 2019. P. 198–201. DOI: 10.1109/ICIMTech.2019.8843801.
2. *Hamrouni B.M.* Logic compression of dictionaries for multilingual spelling checkers // Proceedings of the 15th conference on Computational linguistics Morristown, NJ, USA: Association for Computational Linguistics, 1994. P. 292. DOI: 10.3115/991886.991936.
3. *Lokhande H.A. et al.* Enhancing Text Quality with Bi-LSTM: An Approach for Automated Spelling and Grammar Correction // 2024 International Conference on Advances

in Data Engineering and Intelligent Computing Systems (ADICS). IEEE, 2024. P. 01–07. DOI: 10.1109/ADICS58448.2024.10533521.
4. *Mangu L., Brill E.* Automatic Rule Acquisition for Spelling Correction // Proceedings of the Fourteenth International Conference on Machine Learning (ICML '97). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc, 1997. P. 187–194.
5. *Davrondjon G., Janowski T.* Developing a Spell-Checker for Tajik Using RAISE // Formal Methods and Software Engineering. ICFEM 2002. Lecture Notes in Computer Science / ed. George C., Miao H. Berlin, Heidelberg: Springer, 2002. Vol. 2495. P. 401–405. DOI: 10.1007/3-540-36103-0_41 EDN: XNBRMD.
6. *Atawy S.M. El, ElGhany A.A.* Automatic Spelling Correction based on n-Gram Model // Int J Comput Appl. 2018. Vol. 182, № 11. P. 5–9.
7. *Chen K.-Y., Wang H.-M., Chen H.-H.* A Probabilistic Framework for Chinese Spelling Check // ACM Transactions on Asian and Low-Resource Language Information Processing. 2015. Vol. 14, № 4. P. 1–17. DOI: 10.1145/2826234.
8. *Sasu L.* A Probabilistic Model for Spelling Correction // Bulletin of the Transilvania University of Brasov. Series III: Mathematics, Informatics, Physics. 2011. Vol. 4(53), № 2. P. 141–146.
9. *Kashyap R.L., Oommen B.J.* Spelling correction using probabilistic methods // Pattern Recognit Lett. 1984. Vol. 2, № 3. P. 147-154. DOI: 10.1016/0167-8655(84)90038-2
10. *Chen S.F.* Building Probabilistic Models for Natural Language. 1996.
11. *Flachs S., Lacroix O., Søgaard A.* Noisy Channel for Low Resource Grammatical Error Correction // Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications. Stroudsburg, PA, USA: Association for Computational Linguistics, 2019. P. 191–196. DOI: 10.18653/v1/W19-4420.
12. *Li Y., Anastasopoulos A., Black A.W.* Towards Minimal Supervision BERT-Based Grammar Error Correction (Student Abstract) // Proceedings of the AAAI Conference on Artificial Intelligence. 2020. Vol. 34, № 10. P. 13859–13860. DOI: 10.1609/aaai.v34i10.7202 EDN: QHFSCJ.
13. *Khabutdinov I.A. et al.* RuGECToR: Rule-Based Neural Network Model for Russian Language Grammatical Error Correction // Programming and Computer Software. Pleiades Publishing, 2024. Vol. 50, № 4. P. 315–321. DOI: 10.1134/ S0361768824700129 EDN: XCUPYE.
14. *Martynov N. et al.* A Methodology for Generative Spelling Correction via Natural Spelling Errors Emulation across Multiple Domains and Languages. 2023.
15. *Kaggle:* Your Machine Learning and Data Science Community [Electronic resource]. URL: https://www.kaggle.com/ (accessed: 07.02.2025).
16. *Hugging Face* – The AI community building the future. [Electronic resource]. URL: https://huggingface.co/ (accessed: 07.02.2025).
17. *Language-tool-python.PyPI* [Electronic resource]. URL: https://pypi.org/project/language-tool-python/ (accessed: 11.12.2024).
18. *Hunspell download* | SourceForge.net [Electronic resource]. URL: https://sourceforge.net/projects/hunspell/ (accessed: 02.02.2025).
19. *Goslin K., Hofmann M.* English Language Spelling Correction as an Information Retrieval Task Using Wikipedia Search Statistics // Proceedings of the 13th Conference on Language Resources and Evaluation (LREC 2022). Marseille: European Language Resources Association (ELRA), 2022. P. 458–464.
20. *SymSpell,* Github [Electronic resource]. URL: https://github.com/wolfgarbe/ SymSpell (accessed: 02.02.2025).
21. *Audah H.A., Yuliawati A., Alfina I.* A Comparison Between SymSpell and a Combination of Damerau-Levenshtein Distance with the Trie Data Structure // 2023 10th International Conference on Advanced Informatics: Concept, Theory and Application (ICAICTA). IEEE, 2023. P. 1–6. DOI: 10.1109/ICAICTA59291.2023.10390399.

22. *Проверка* орфографии, грамматики и стилистики онлайн – LanguageTool [Electronic resource]. URL: https://languagetool.org/ru (accessed: 02.02.2025).
23. *Sorokin A.A., Shavrina T.* Automatic spelling correction for Russian social media texts // Dialogue, International Conference on Computational Linguistics. Moscow: URL: https://www.researchgate.net/publication/303813582_Automatic_spelling_correction_for_ Russian_social_media_texts, 2016. EDN: XMWHDK (accessed: 02.02.2025).
24. *Pandas* 2.2.3 documentation [Electronic resource]. URL: https://pandas.pydata. org/docs/ (accessed: 02.02.2025).

References

1. Zukarnain, N., Abbas, B. S., Wayan, S., Trisetyarso, A. and Kang, C. H. (2019), "Spelling Checker Algorithm Methods for Many Languages", *in: 2019 International Conference on Information Management and Technology* (ICIMTech). IEEE, pp. 198-201. https://doi.org/ 10.1109/ICIMTech.2019.8843801.
2. Hamrouni, B. M. (1994), "Logic compression of dictionaries for multilingual spelling checkers, in: Proceedings of the 15th Conference on Computational Linguistics", *Association for Computational Linguistics*, Morristown, NJ, USA, p. 292. https://doi.org/10.3115/991886. 991936.
3. Lokhande, H. A., Kinage, L. J., Kolunkar, P. M., Salunkhe, J. M. and Kale, S. (2024), "Enhancing Text Quality with Bi-LSTM: An Approach for Automated Spelling and Grammar Correction", *in: 2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems* (ADICS), IEEE, pp. 01-07, https://doi.org/10.1109/ADICS58448. 2024.10533521.
4. Mangu, L. and Brill, E. (1997), "Automatic Rule Acquisition for Spelling Correction", *in: Proceedings of the Fourteenth International Conference on Machine Learning* (ICML '97), Morgan Kaufmann Publishers Inc, San Francisco, CA, USA, pp. 187-194.
5. Davrondjon, G. and Janowski, T. (2002), "Developing a Spell-Checker for Tajik Using RAISE", in: George, C., Miao, H. (Eds.), *Formal Methods and Software Engineering,* ICFEM 2002. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, pp. 401-405. https://doi.org/10.1007/3-540-36103-0_41.
6. Atawy, S. M. El, and ElGhany, A. A. (2018), "Automatic Spelling Correction based on n-Gram Model". Int J Comput Appl 182, pp. 5-9. https://doi.org/10.5120/ijca2018917724.
7. Chen, K.-Y., Wang, H.-M. and Chen, H.-H. (2015), "A Probabilistic Framework for Chinese Spelling Check", *ACM Transactions on Asian and Low-Resource Language Information Processing*, vol. 14, no. 4, pp. 1-17. https://doi.org/10.1145/2826234.
8. Sasu, L. (2011), "A Probabilistic Model for Spelling Correction", *Bulletin of the Transilvania University of Brasov*, *Series III: Mathematics, Informatics, Physics,* vol. 4(53), no. 2, pp. 141-146.
9. Kashyap, R. L. and Oommen, B. J. (1984), "Spelling correction using probabilistic methods", *Pattern Recognit", Lett,* vol. 2, no. 3, pp. 147-154. https:// doi.org/10.1016/0167-8655(84)90038-2.
10. Chen, S. F. (1996), "Building Probabilistic Models for Natural Language".
11. Flachs, S., Lacroix, O. and Søgaard, A. (2019), "Noisy Channel for Low Resource Grammatical Error Correction", *in: Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications. Association for Computational Linguistics,* Stroudsburg, PA, USA, pp. 191-196. https:// doi.org/10.18653/v1/W19-4420.
12. Li, Y., Anastasopoulos, A. and Black, A. W. (2020), "Towards Minimal Supervision BERT-Based Grammar Error Correction (Student Abstract)", *Proceedings of the AAAI Conference on Artificial Intelligence* vol. 34, no. 10, pp. 13859-13860. https://doi.org/10.1609/ aaai.v34i10.7202.

13. Khabutdinov, I. A., et al. (2024), "RuGECToR: Rule-Based Neural Network Model for Russian Language Grammatical Error Correction", *Programming and Computer Software. Pleiades Publishing,* vol. 50, no. 4, pp. 315-321.
14. Martynov, N. et al. (2023), "A Methodology for Generative Spelling Correction via Natural Spelling Errors Emulation across Multiple Domains and Languages".
15. Kaggle: Your Machine Learning and Data Science Community [Electronic resource]. URL: https://www.kaggle.com/ (accessed: 07.02.2025).
16. Hugging Face – The AI community building the future. [Electronic resource]. URL: https://huggingface.co/ (accessed: 07.02.2025).
17. Language-tool-python.PyPI [Electronic resource]. URL: https://pypi.org/project/language-tool-python/ (accessed: 11.12.2024).
18. Hunspell download | SourceForge.net [Electronic resource]. URL: https://sourceforge.net/projects/hunspell/ (accessed: 02.02.2025).
19. Goslin, K. and Hofmann, M. (2022), "English Language Spelling Correction as an Information Retrieval Task Using Wikipedia Search Statistics", *Proceedings of the 13th Conference on Language Resources and Evaluation* (LREC 2022). Marseille: European Language Resources Association (ELRA), pp. 458-464.
20. SymSpell, Github [Electronic resource]. URL: https://github.com/wolfgarbe/SymSpell (accessed: 02.02.2025).
21. Audah, H. A., Yuliawati, A. and Alfina I. (2023) "A Comparison Between SymSpell and a Combination of Damerau-Levenshtein Distance with the Trie Data Structure, *2023 10th International Conference on Advanced Informatics: Concept, Theory and Application* (ICAICTA). IEEE, pp. 1-6.
22. Spelling, grammar and style checker online – LanguageTool [Electronic resource] URL: https://languagetool.org/ru (accessed: 02.02.2025).
23. Sorokin, A. A. and Shavrina, T. (2016), "Automatic spelling correction for Russian social media texts", *in: Dialogue, International Conference on Computational Linguistics,* https://www.researchgate.net/publication/303813582_Automatic_spelling_ correction_for_Russian_social_media_texts, Moscow.
24. Pandas 2.2.3 documentation [Electronic resource]. URL: https://pandas.pydata. org/docs/ (accessed: 02.02.2025).

Информация об авторах:

Е. В. Исаева – кандидат филологических наук, доцент, зав. кафедрой английского языка профессиональной коммуникации, Пермский государственный национальный исследовательский университет (614990, Россия, г. Пермь, ул. Букирева, 15), Scopus Author ID: 57204498718, ResearcherID: O-6777-2015;

Б. З. Сафарбеков – студент 2-го курса магистратуры, Национальный исследовательский технологический университет "МИСиС" (НИТУ "МИСиС") (119049, г. Москва, Ленинский проспект, д. 4, стр. 1).

Information about the authors:

E. V. Isaeva – Candidate of Science (in philology), Associate Professor, Head of the Department of English Language of Professional Communication, Perm State University (15, Bukireva St., Perm, Russia, 614990), Scopus Author ID: 57204498718, ResearcherID: O-6777-2015;

B. Z. Safarbekov – 2nd year Master's student, National University of Science and Technology "MISIS" (4, B. 1, Leninsky pr., Moscow, Russia, 119049).