Ministry of Science and Higher Education
of the Russian Federation
ITMO University

Faculty of Digital Transformations
Educational program Big Data and Machine Learning
Subject area (major) Applied Mathematics and Informatics

REPORT

on practical training research internship

Task topic: Data Analysis in Natural Language in Medicine
Student: Dmitry Pogrebnoy, J4132c

Head of Practice from the trainee's host organization: Sergey Kovalchuk, PhD.
Head of Practice from ITMO University: Semyon Kraev, PhD

Practice completed with grade ____
Commission member signatures:

_____ full name
(signature)

_____ full name
(signature)

_____ full name
(signature)

Date _____

St. Petersburg
2022

Contents

Introduction 3
1 Overview of the spelling correction 5
1.1 Types of spelling errors 5
1.2 Edit distance 6
1.3 Automatic spelling correction 7
1.4 Principle design of spelling correction tools 9
1.5 Related works 11
2 Overview of existing tools 15
3 Overview of BERT models 19
3.1 BERT model architecture 19
3.2 Popular BERT model modificationы 21
4 Tool architecture and implementation 23
4.1 Edit Distance Index 27
4.2 Language Model 29
5 Tool approbation 31
Conclusion 34
Acknowledgments 36
References 37

Introduction

In today's world, digitalization is rapidly spreading into all areas of our lives. Electronic document management makes it possible to work more efficiently with documents, increases the speed of communication and improves the quality of work of many employees in different professions.

Due to breakthrough achievements in natural language processing in the last two decades it has become possible to build various intelligent systems and machine learning models based on unstructured document storages, of which there is a significant amount. These systems can be applied in many fields, such as medicine and healthcare.

In medicine and health care, it is possible to build various predictive models and decision-making models based on information from patients' medical records, which can potentially improve the quality and effectiveness of treatment, and even save people from the severe consequences of various diseases. However, the accuracy and quality of such models is highly dependent on the quality of the input electronic records, which are usually presented as unstructured text.

One of the main problems complicating automatic processing of electronic records is various spelling errors in medical terms and simple words. According to a study by Toutanova et al. [1], spelling errors occur mainly for two reasons. The first reason is mainly related to the person himself and is that the writer may not know exactly how to spell a word correctly and therefore make mistakes. The second reason has to do with technology and is due to poor-quality typing devices, which can also lead to spelling errors. As a result, electronic documents with spelling errors significantly reduce the quality of the final models, which makes it impossible to achieve acceptable results.

In order to try to solve the problem of spelling errors, Balabaeva et al. [2] conducted a research that resulted in the development of a tool to correct spelling errors in Russian medical texts. Despite the impressive accuracy metrics obtained by the tool, there are still many directions to improve the final result and the speed of the tool, thereby further improving the quality of the source texts, which will positively affect the quality of medical models.

Problem statement

The purpose of this research work is to develop an automatic spelling correction tool for analyzing clinical texts in Russian. The tool should accept raw medical text as input and return corrected text with a minimum number of incorrect words.

To achieve the goal in the first year, the following tasks were set.

— Perform an overview of the Russian medical texts correction.
— Analyze existing solutions in this area.
— Design the spelling correction process for a new tool.
— Design the architecture of a spellchecker tool.
— Conduct testing and approbation of the developed tool.

1 Overview of the spelling correction

This chapter provides general information on correcting errors in texts. Types of errors in texts are described and a list of the types of spelling errors that this paper focuses on is highlighted. An overview of the different edit distances is also made. In addition, a general approach to automatic correction of errors in texts is described and a generalized architecture of such tools is presented. A overview of relevant articles devoted to the correction of English, Russian, and separately medical texts is also made.

1.1 Types of spelling errors

The development of any system related to correcting errors in real texts requires determining the types of errors that the system should correct. For example, there are several fundamental works [3, 4, 5] for the English language that investigate the mechanisms and causes of errors in texts, as well as classify errors into various groups.

There are many classifications of errors in real texts, depending on the specific language and the task at hand. One of the most generalized and applicable to the Russian language is the classification by reason of an error [6, 1]. According to this classification, there are two main groups of errors.

1) Orthographic mistakes. Such mistakes are primarily related to the writer himself. A person may not know how to write a specific word or is just learning a new language and make mistakes even in simple words. In addition, the writer may also have some cognitive disorders that do not allow him to write a word without mistakes. In addition, the language may have specific rules for writing words that distinguish the pronunciation of a word from its spelling, which also leads to errors.

2) Type mistakes. These errors are mainly related to the technical devices of the writer. Sticking or unpressed keys on the keyboard, a tight spacebar, and the substitution of two close keys due to quick typing on a low-quality keyboard. All of these errors are related to the input device in one way or another, and almost do not depend on a particular language. At the same time, the keyboard

layout has a great impact, which can both significantly increase and decrease the number of such mistakes.

Unfortunately, this classification is quite general and cannot fully reflect the peculiarities of the Russian language. That is why we will use the following classical classification of errors for the Russian language.

1) Grammatical mistakes. Errors related to incorrect word formation.

2) Punctuation mistakes. Errors associated with incorrectly placed or missing punctuation marks in a sentence.

3) Spelling mistakes. Errors connected with a misspelled word.

This paper will focus mainly on correcting misspellings and will not deal with other groups of errors.

1.2 Edit distance

In 1966, the mathematician Levenshtein introduced [7] the edit distance model to estimate how much one string differs from another. This distance is the minimum number of operations to delete a character, insert a character and replace a character in order to get from one string to another. Later, Damerau and Levenshtein added another operation of transposition of two neighboring characters [8]. Damerau found that about 80% of spelling errors in English belong to the group of errors with an Damerau-Levenshtein edit distance of one.

The edit distance metric greatly depends on what operations are allowed with symbols. Therefore, there are several different edit distance metrics.

— Levenshtein distance [7]. Uses the operations of inserting a character, deleting a character, and replacing a character.

— Damerau–Levenshtein distance [8]. In addition to the operations of inserting a character, deleting a character and replacing a character, the transposition of two adjacent characters is also used.

— Longest common subsequence (LCS) [9]. Only inserting and deleting a character is allowed, without substituting a character.

— Hamming distance [10]. Only character substitution is allowed. Due to this limitation, this metric can only be applied to strings of the same length.

— Jaro–Winkler distance [11]. Uses only transpositions of two characters in a string.

Based on the edit distance model operations, the following types of spelling errors naturally stand out. Examples of spelling mistakes are shown in the Figure 1.1. Various combinations of basic errors are possible, and the more errors a word contains, the more difficult it is to correct it. Therefore, most often words with one or two errors are correctable, and with three or more they can no longer be effectively corrected using the edit distance.

Type of mistake	Incorrect text	Correct text
Wrong characters	тубиркулез	туберкулез
Missing characters	туб□ркулез	туберкулез
Extra characters	туберкпулез	туберкулез
Shuffled characters	тубрекулез	туберкулез
Missing word separator	острый|туберкулез	острый_туберкулез
Extra word separator	туб_еркулез	туберкулез

Figure 1.1 — Examples of spelling mistakes.

1.3 Automatic spelling correction

Modern tools for correcting spelling errors can help people even in the most ambiguous cases. However, the process of operation of such systems can be generalized to the following three stages.

1) Error detection.
2) Candidate proposal.
3) Ranking candidate.

These generalized stages are present in every stand-alone text correction tool. Some variation is possible, but in general these stages are applicable to the description of most such tools. Below we will look at each stage of correction in detail.

At the first stage, it is necessary to detect the presence of an error in the word. For this purpose, prepared dictionaries are usually used. The logic of the tool's work with the dictionary is quite simple. If the word in question is contained in the dictionary, then everything is fine with it, otherwise the tool considers that this word is written with an error and requires correction.

Once the tool has detected an invalid word, the correction process is performed. At first, a list of candidate words for correction is computed. Typically, the same prepared vocabulary is used for this as for detecting incorrect words. A predefined edit distance between the incorrect word and the dictionary word is calculated for each word. Usually the editing distance is limited to one or two, as a larger distance requires more calculations and greatly impairs the performance. Words with larger edit distances are not included in the list of candidates. As a result we get a list of possible correction candidates. Thus the problem of choosing the most suitable word among all candidates arises.

To find the most appropriate candidate word, usually some language model is used that is trained on texts with similar subject matter. Such language model allows using information about context, structure, and language features for better correction of spelling errors. For each candidate word, the model calculates the value of some metric which allows you to estimate how suitable the word is for fixing the incorrect one. In this way the model allows ranking the candidates from the most suitable to the most unsuitable. As a result, after ranking, the most suitable candidate is selected and replaces the incorrect word.

Stages of the spelling correction tools are shown in the Figure 1.2.

The effectiveness of a correction tool and its quality metrics depend largely on the vocabulary of correct words and on the quality of the language model. The more complete and error-free the dictionary is, the more words are correctly recognized by the tool as incorrect. However, in practice it is impossible to assemble a dictionary that will contain absolutely all possible words. In addition, real words may not be dictionary words and therefore also be recognized as incorrect. Therefore, already at this stage there are errors associated with recognition and obviously correct words are recognized as incorrect. In addition, the prepared dictionaries are not perfect and may contain

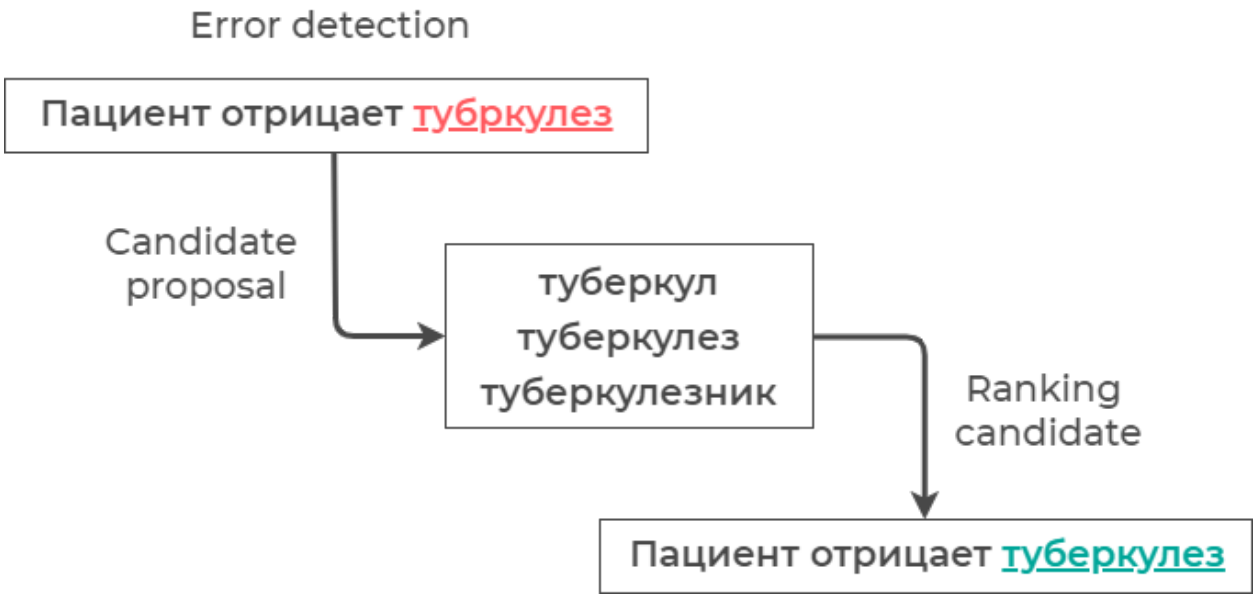

Figure 1.2 — The automatic spelling correction process.

incorrect words. Thus, the tool can skip an incorrect word because it is in the dictionary of correct words.

It should also be noted that in practice it is almost impossible to prepare a large enough corpus without errors to teach the language model. Therefore, errors in the word correction tool appear at this stage as well. The cleaner and better the training corpus of the language model, the more correct ranking it will show.

However, although it is almost impossible to get rid of these problems altogether, we should try to do what we can to improve the quality of the vocabulary and the language model.

1.4 Principle design of spelling correction tools

Special attention should be paid to the architecture of such tools. Typically, a spelling correction tool has pre-processing and post-processing components, as well as language and error models. The generalized architecture is shown in the Figure 1.3. Take a closer look at each component.

Pre-processor is responsible for pre-processing the input data for subsequent processing within the tool. The component accepts text for correction, removes punctuation, and splits text into separate words. This component may also be responsible for extracting lexemes from separate words if the tool logic requires it.

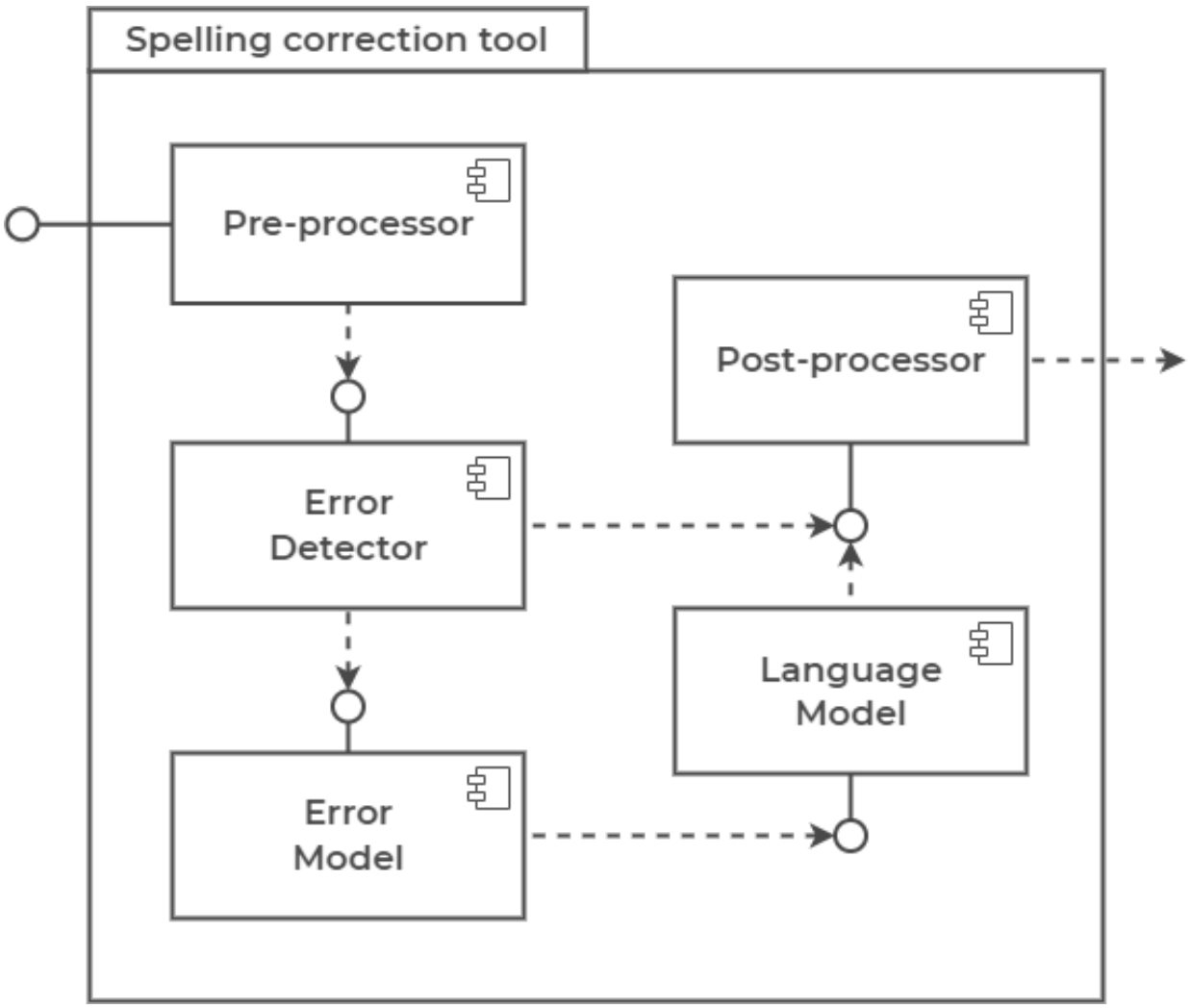

Figure 1.3 — Principle architecture of the spelling correction tool.

Error Detector checks if there is an error in a particular word. And if the word is recognized as incorrect, then a further process to correct it starts, otherwise the word is skipped. As already described in Section 1.3, usually a prepared dictionary is used for this purpose, which should contain only correct words. If a word is present in such a dictionary, it is considered correct. Otherwise, the word is considered a word with an error.

Error Detector component may also be responsible for word filtering. This can be useful for handling abbreviations or any names that are spelled with a capital letter. In this case, such words will be ignored and forwarded directly to the Post-processor component.

Error Model makes a list of potential candidates for which an incorrect word can be corrected. As described in Section 1.3, this usually uses some given edit distance and its operations. Applying valid editing distance operations to

an invalid word yields a set of words. Those contained in the prepared dictionary of correct words are added to the list of potential candidates. The number of possible operations on the word increases exponentially with the increasing value of the editing distance. Therefore, for performance reasons, a distance of one or two is typically used. In addition, according to the study [8], the majority of errors in words are at an edit distance of 1 or 2 from the correct word.

Language Model component is responsible for ranking the candidate list obtained from the error model. The language model can vary greatly from tool to tool. It can be built on frequency dictionaries, which rank words by their frequency of use in a thematic corpus of texts. Or the model can be built on the basis of n-grams. For example, when words around an incorrect word and a corpus of thematic texts are used to rank candidates. Also, various models based on embeddings are used in relevant works. Such language models are built on the basis of such models as Word2Vec [12], FastText [13, 14], ELMo [15], GloVe [16] and others. In the most recent works, language models often use neural networks, such as modifications of the BERT [17] neural network. As a result, after ranking, the component outputs the most appropriate candidate as a fix.

Post-Processor component reassembles the text you are looking for from checked and corrected words, and arranges punctuation marks. If necessary, this component can also put the corrected word or lexeme into the desired word form. As a result, this component returns a ready corrected text.

The described components are basic and are present in almost all tools for correcting spelling errors. Of course, each tool is unique. Therefore, in addition to the components described above, there may be other equally important functional parts of the tools.

1.5 Related works

There are a number of research papers devoted to different approaches to correcting spelling errors. The most advanced tools and articles focus on correcting spelling errors in English texts.

One of the major relevant studies is that of Daniel Hládek et al. [18]. This paper provides a basic overview of the various approaches to text correction

and makes an outline of most of the work in this field over the past 20 years. The study pays a lot of attention to methods and techniques of text correction for quite different languages. Among others, several articles aimed at correcting Russian texts are mentioned. In this way, this paper is a compilation of theoretical foundations and describes the progress in text correction to date.

Another relevant work is a new framework for correcting spelling mistakes in English created by Sai Muralidhar Jayanthi et al. [19]. In their paper they present a new open-source tool, which is based on 10 different pre-trained models. The distinguishing feature is that the models are trained on synthetically generated misspellings in words, which the authors say has increased the correction rate by 9 percent. In addition, this toolkit stands out because it takes into account the context around the word to be corrected, which greatly increases its accuracy relative to other well-known tools.

A paper by Yifei Hu et al. [20] investigates the approach of applying the BERT neural network and editing distance to the correction of spelling errors in texts. The authors consider two cases. The first case first applies a BERT model that predicts by context the word in place of the incorrect word. And then the word with the lowest editing distance and a larger metric is selected from the list of candidates. In the second case it is vice versa. First the edit distance is applied and a list of candidates is generated, and then for each candidate the BERT model is used to calculate how well it fits the context. The candidate with the highest metric is selected as the corrected one. As a result, the second method was the best with 84.91% accuracy.

However, for the Russian language, the correction of spelling errors in texts is much less developed. Nevertheless, there are several relevant articles aimed at this topic.

Correction of Russian texts

First of all, it is necessary to note the article by Alexei Sorokin et al. [21] about the SpellRuEval[1] competition on automatic correction of spelling errors in Russian media texts. Seven teams from different Russian

[1]Official website of the SpellRuEval competition: `https://www.dialog-21.ru/evaluation/2016/spelling_correction/`

universities participated. As a result, the winner was the team from Moscow State University, who published the results of their work in an article [22]. In order to win the competition they created a tool based on editing distance and phonetic similarity of words to generate a list of candidates, as well as using a probabilistic language model followed by the use of logistic regression to correctly rank the candidates.

Also one of the most recent articles is the work of Alla Rozovskaya [23] in which she uses three different language models to compare quality metrics for spelling error correction on three different datasets. Two of the models do not take context into account, while the other model does. According to the results of the experiments, the context-based language model showed the best results, which once again proves that context must be taken into account for a better word correction process.

There is also an article by Alexey Sorokin [24] describing a tool for correcting errors in Russian text based on the context noisy channel model and re-ranking the best candidates using logistic regression. According to the results of the experiments the obtained tool obviously outperforms the winners of the SpellRuEval contest.

Only a few works are focused to correcting spelling errors in medical texts.

Correction of medical texts

One such study is the work of Lai et al. [25] which investigates an approach for correcting spelling errors using the Aspell [26] tool for generating candidates and a developed noisy channel model for ranking the resulting candidates. The resulting tool was tested on several types of datasets with medical data. As a result, the obtained tool significantly outperformed Aspell on all metrics.

Another relevant study is the article by Pieter Fivez et al. [27], which presents an approach to editing spelling errors in medical texts based on an unsupervised context model using symbols n-gram embeddings. The resulting model outperforms the metrics of the model built by Lai et al. [25].

The only relevant article focused on the correction of spelling errors in Russian medical texts was published by Ksenia Balabaeva et al. [2]. In it the authors use the Levenshtein distance to generate a list of candidates and the trained FastText model to rank them. The results described in the article are impressive, according to the test results the best overall accuracy is 0.86.

It should be noted that despite the good results obtained in the article, the field of correction of Russian medical texts is underdeveloped and requires additional research. The results of which will help improve the quality of existing medical information systems and allow building new more advanced ones. This work will partially fill this gap.

2 Overview of existing tools

There are several popular and well-known open source tools that can automatically correct spelling errors in words. Such systems are used in almost all systems that work with unstructured texts such as electronic documents, search queries, news, articles, and others. Thanks to such tools, the quality of the source text increases, which opens up opportunities for further intelligent processing and the construction of effective information systems.

Most of these tools focus mainly on English text, while the other languages fall by the wayside and cannot boast such a rich range of tools. Nevertheless, there are several tools that support the Russian language.

After analyzing the tools to correct spelling errors, the following parameters were identified for comparison.

— Error precision. This is the percentage of words with an error that the tool correctly corrected relative to all incorrect words.

— Lexical precision. This is the percentage of correct words without mistakes that the tool does not correct relative to all correct words.

— Overall precision. This is the average of error precision and lexical precision.

— Average number of words processed per second.

A set of two hundred misspelled words was used to calculate the error precision. This list was formed on the basis of incorrect words in patient medical records provided by the Almazov National Medical Research Center (Almazov Center) during 2010-2015. Each incorrect word was submitted to the tool one by one, which makes it impossible to use the context around the incorrect words. However, the tools considered do not use context, so this limitation is not significant.

It is also worth noting that this word list is exactly the same as in the article by Balabaeva et al. [2]. Unfortunately, the list of words for calculating lexical precision in Balabayeva et al. article has not been preserved. Therefore, to calculate this metric, a test set of two hundred correct medical words was formed. Similarly to the calculation of error precision, these words were submitted to the tools one by one.

In order to better compare the tools was also calculated overall precision, which is the average of lexical and error precision. This metric allows estimating the quality of spelling error correction in general.

However, in addition to the quality of correction of spelling errors, it is also necessary to take into account the time in which these errors are corrected. Of course, the faster the tool works, the better. The performance test was done on a laptop on Ubuntu 20.04 with 16 GB RAM and Intel Core i7-9750H CPU @ 2.60GHz * 12.

An important limitation is that the compared tools must be able to run from Python. This limitation is due to the fact that the medical text correction tool must be easily built into the medical text processing pipeline in the library created by ITMO University Digital Healthcare Laboratory. Therefore, wrappers or bindings of the tools in focus were used in cases where it was not possible to run the tool from Python.

Table 2.1 presents a comparison of some popular open-source tools for correcting spelling errors. Unfortunately, there are no production-ready open-source tools that focus on Russian medical texts at the moment. Therefore, only general-purpose tools are presented in the table.

Tool name	Error precision	Lexical precision	Overall precision	Average words per second
Aspell-python [28]	0.65	0.775	0.7125	353
PyHunspell [29]	0.59	0.49	0.54	11.5
PyEnchant [30]	0.6	0.455	0.5275	26.4
LanguageTool-python [31]	0.64	0.845	0.7425	19.1
PySpellChecker [32]	0.335	0.765	0.55	4.3
SymspellPy [33]	0.42	0.78	0.6	15892.1
SymspellPy (compound) [34]	0.46	0.54	0.5	983.1
Jumspell [35]	0.395	0.925	0.66	2043.2
Spellchecker prototype [2]	0.41	0.83	0.62	0.07

Table 2.1 — Comparison of metrics for popular open source spelling correction tools.

The first three tools are Python wrappers of the well-known general-purpose spelling error correction tools Aspell [26], Hunspell [36], and Enchant [37], respectively. The best result of these three was shown by the Aspell tool, which is used to correct spelling errors in Linux distributions.

The following are metrics for the LanguageTool Python wrapper. LanguageTool is written in Java and uses a rule-based approach. This tool is used to fix texts in many systems. For example, in the Grazie [38] plugin for the popular integrated development environment Intellij IDEA [39].

The following are the metrics for the PySpellchecker tool, which implements Peter Norvig's approach described in the original article [40]. This tool showed the lowest error precision metrics.

Also shown in the table are the metrics of the Symspell tool Python wrapper. The Symspell [33] is based on the new Symmetric Delete spelling correction algorithm, which reduces the complexity of generating edit candidates and dictionary searches for a given Damerau-Levenstein distance. This tool is language-independent and requires only a dictionary of all possible words and a frequency dictionary of the subject texts.

It is worth noting that Symspell has two modes of operation. In normal mode, the tool corrects spelling errors, while in compound mode, the tool also tries to split coupled words. Therefore, the normal mode has an impressive performance, while the compound mode has higher error precision metric.

In addition, the table shows a modification of Symspell, the Jumspell [35] tool, which uses a language model in addition to the error model, and as a result has a higher lexical precision metric than Symspell. The tool is written in C++, but has Python bindings.

And the last tool is the prototype from the article by Balabayeva et al [2]. Unfortunately, the tool described in the article could not be obtained, but the prototype of this tool has been preserved. This instrument has an extremely low performance, but it has a good lexical precision.

It should be clarified that these tools have not been fine-tuned to work with medical texts. The default dictionaries and models provided by the developers of the tools were used for testing.

It should also be noted that there are also several commercial tools with closed source code, which are able to correct spelling errors in Russian texts. One of the most famous is Ynadex.Speller [41]. This tool in addition to the closed source code has a strict user agreement. Another tool is a commercial version of Jumspell [42] with an improved language model. Webiomed [43] and

Sberbank AI Lab [44] are also involved in processing medical texts. However, they do not have public tools for spelling. Nevertheless, any commercial tools were not considered in this comparison.

Required performance

In order to define the required tool performance, and then correctly evaluate the performance metric of the compared tools, it is necessary to extract information about the approximate number of words in the medical record. The average value of the number of words in the anamnesis will not suit, as it is necessary to quickly process most such documents, and the average value does not provide such information. Therefore, the decision on the required number of words processed per second will be based on percentiles. The corresponding percentile values are given in Table 2.2.

Percentiles	0.25	0.50	0.75	0.90	0.95	0.99
Word number	37	63	104	149	189	275

Table 2.2 — Percentile values according to the number of words in the anamnesis.

According to the data presented in the table, we will focus on the required performance of the tool equal to 150 words per second, which allows the tool to process almost every anamnesis in less than a second.

To summarize, among the reviewed tools, half of them do not meet the performance requirement, in addition to the fact that none of the tools shows acceptable accuracy in correcting medical texts in Russian. Thus, it is necessary to develop a tool that will satisfy the performance requirement and at the same time show the better accuracy metrics.

3 Overview of BERT models

This chapter describes the architecture of the language representation model BERT, as well as several popular models built on top of this architecture.

3.1 BERT model architecture

A group of scientists from Google Jacob Devlin et al. in 2019 published an article [17] presenting a new architecture of the language representation model called BERT — Bidirectional Encoder Representations from Transformers.

The architecture of the BERT model is a multi-layered bidirectional Transformer encoder, which is very close to the original implementation presented in the article by Vaswani et al. [45].

The article defines two models. The first model, $BERT_{BASE}$, is used for comparison with the Open AI GPT model[46] and is not important for this work. The second model, $BERT_{LARGE}$, is used to achieve the best results in tests. This model consists of 24 layers with a hidden size of 1024 and a number of self-attention heads as 16. As a result, the total number of model parameters is 340 million.

The training of this type of model takes place in two stages. At the first stage, the model is pre-trained. At this step, the model is trained on unlabeled data as part of various pre-training tasks. After that, at the stage of fine-tuning, the model is trained for a specific task from the real world. To fine-tune the model for a specific task, first the model is initialized with weights from the pre-trained model, and then this model is fine-tuned to the labeled data of a specific task. This approach allows to adapt the same pre-training model for different tasks by fine-tuning to a specific task.

As an example, consider the Masked LM procedure of the BERT model. First, the source text is tokenized, service tokens are added and the necessary tokens are replaced with the $[MASK]$ token. After that, the resulting tokens are converted into a vector form and transferred to to the transformer layers. After that, the result obtained from the transformers is decoded into a token and we get a prediction of the word under the tag $[MASK]$.

A visual example of this process is shown in Figure 3.1.

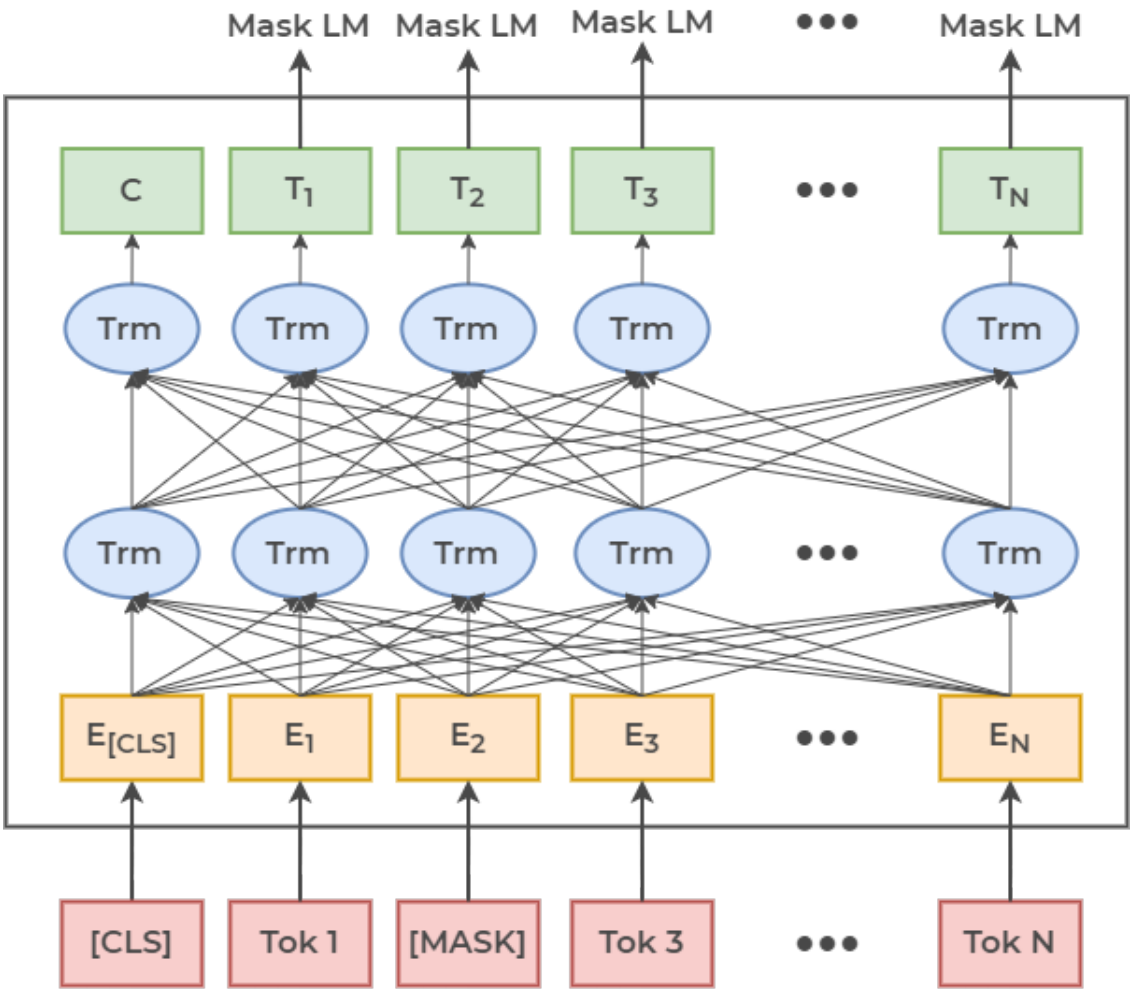

Figure 3.1 — Overall Masked LM procedure of BERT model.

The main key feature of this model, which made it possible to achieve such high results in the texts of the Masked LM (MLM) task [47], is that the model uses the context not only to the left or right of the missing word, but simultaneously takes into account both parts of the context, which makes it possible to predict the missing word more precise. Similar feature are also applicable for other tasks, for example, for the Next Sentence Prediction (NSP) task.

More detailed information about the architecture of BERT models and the pre-training process can be found in the original article [17].

3.2 Popular BERT model modificationы

The architecture of the BERT language model has become a breakthrough technology in the field of NLP and has gained great popularity. Many scientists and research groups wanted to contribute to the development of this approach, improve the architecture of the model and achieve even higher metrics. Next, several of the most famous and significant modifications of the original BERT model will be described.

One of the most popular modifications of the BERT model is the DistilBERT model. This model was presented by Victor Sanh et al. in their work [48] in 2019. The architecture of this model is in many ways similar to the $BERT_{BASE}$ architecture, but a significant difference is that DistilBERT has twice as many layers, which makes it significantly smaller than the original $BERT_{BASE}$ model. The authors claim that the key feature of the DistilBERT model is that it is 40% smaller and 60% faster than the original $BERT_{BASE}$. But despite the smaller size of the model, it retains 97% of the language understanding capabilities. This is achieved through the use of a distillation approach. Due to the reduced size and increased speed of operation, this model is less demanding on computing resources and can be widely used in various devices. The pre-trained $BERT_{BASE}$ acts as a teacher, and DistilBERT acts as a student, which is trained to reproduce the behavior of a larger model. More details about the training of the model and the results obtained can be found in the original article.

Another modification of BERT is called RoBERTa. A group of Facebook AI scientists Yinhan Liu et al. presented this model in their research [49] in 2019. The authors found out that the BERT model is still under-trained and proposed an improved model pre-training process to achieve better performance. To better pre-train the model, they used 10 times more data, 160GB instead of 16GB for BERT. Significantly increased the number of training iterations, increased the patch size several times, removed the NSP task from the pre-training process and used a dynamically changing masking pattern applied to training data. All this made it possible to significantly increase the metrics of the model and surpass the BERT model. More details about this model are available in the original article [49].

One more popular modification of the architecture of the BERT model is called ELECTRA. This model is presented in the work [50] of Kevin Clark et al. in 2020. The key feature of this modification is an improved model pre-training process. The authors present a new approach to retraining the model called replaced token detection. In this approach, some tokens are replaced with samples from a small generator network, and the model tries to determine whether this token has been replaced or not. Thus, the model is trained on all input tokens, and not just on a masked subset, which usually makes up about 15% percent of all input tokens. Thanks to a more efficient approach, the ELECTRA model, whose performance is comparable to the XLNet [51] and RoBERTa [49] models, requires less than 25% of computes for pre-training, which makes pre-training models more accessible to businesses and scientists.

In addition to the described modifications of the model, there are many other popular modifications and promising approaches. Despite this, the RoBERTa model is a good starting point for the study of spelling errors correction in Russian in medical texts and is therefore used in this work.

4 Tool architecture and implementation

The new tool for correcting the spelling of medical texts is written in Python and supports working only with Russian text. The tool consists of seven components. The architecture of the tool is shown in the Figure 4.1.

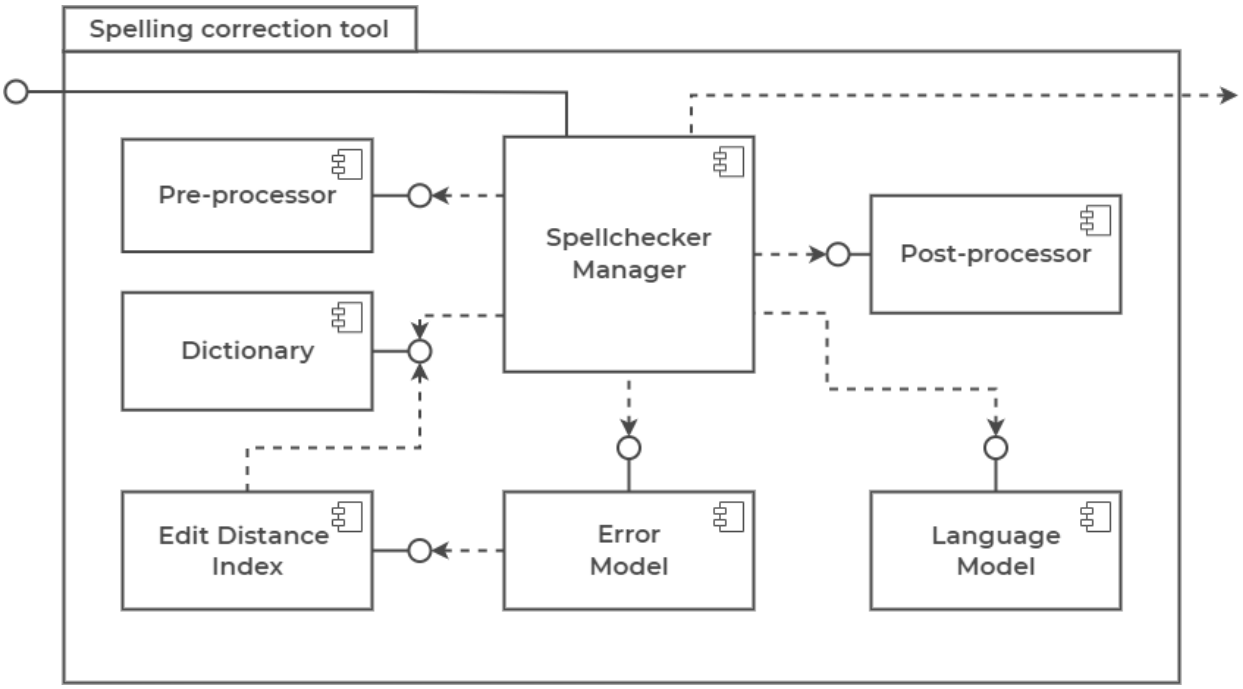

Figure 4.1 — Architecture of the new spelling correction tool.

The architecture of the tool is markedly different from the basic architecture. The center of the architecture is the Spellchecker Manager component, which is responsible for coordinating the other components of the system and the top-level business logic of the tool. This feature allows the developer to independently replace the corresponding components, while preserving only the interface of interaction with them. Thus, it becomes possible to independently develop and improve each component individually without having to make changes to the entire system as a whole.

Like the principle architecture, the architecture of the tool includes pre-processing and post-processing components. Pre-processing component is responsible for splitting the text into separate words, removing punctuation and capitalization. In contrast, Post-processing component reassembles the whole text from individual words. To split the text into tokens, the tool uses the

sacremoses[1] library. Also after tokenization, each token is lemmatized. This operation is performed using the pymorphy2[2] library.

Dictionary component contains a dictionary of correct words and allows you to quickly determine whether a word is correct or requires corrections. This dictionary contains lemmatized words from several source dictionaries. One of them is a dictionary prepared in the work of Balabaeva et al. [2]. The Russian dictionary of the popular open tool for spellchecking ASpell [26] was also used and processed. In addition, the OpenCorpora [52] dictionary was used, which contains most of the various forms of a large number of words. Each source dictionary was processed as follows. All words containing less than three letters were excluded, and words containing non-Russian symbols and numbers were also excluded. All words were reduced to lowercase and lemmatized. Finally, all three source dictionaries were combined into one final dictionary. The final dictionary contains 214629 words in the primary form.

Error Model component is responsible for generating a list of candidates for correcting an incorrect word. Compared to the main architecture of the tool, this architecture has some differences. The error model uses the Damerau-Levenstein editing distance to create a list of candidates. This is a computation-intensive operation, therefore, a special index is used to significantly speed up the operation and increase the overall performance of the tool.

Language Model is responsible for ranking the editing candidates and choosing the most suitable one to replace the incorrect word. To rank candidates, a fine-tuned machine learning model based on the BERT architecture is used in Russian medical texts. This allows you to take into account the context of an incorrect word and choose a suitable replacement more efficiently.

In general, all components together fully ensure the implementation of the spellchecking process. The diagram of the spellchecking process is shown in the Figure 4.2.

The process is arranged as follows. First of all, the medical text is divided into tokens. Next, it is checked whether there are non-Russian letters in the

[1]Sacremoses tokenizer library: `https://github.com/alvations/sacremoses`

[2]Pymorphy2 lemmatizer library: `https://pypi.org/project/pymorphy2/`

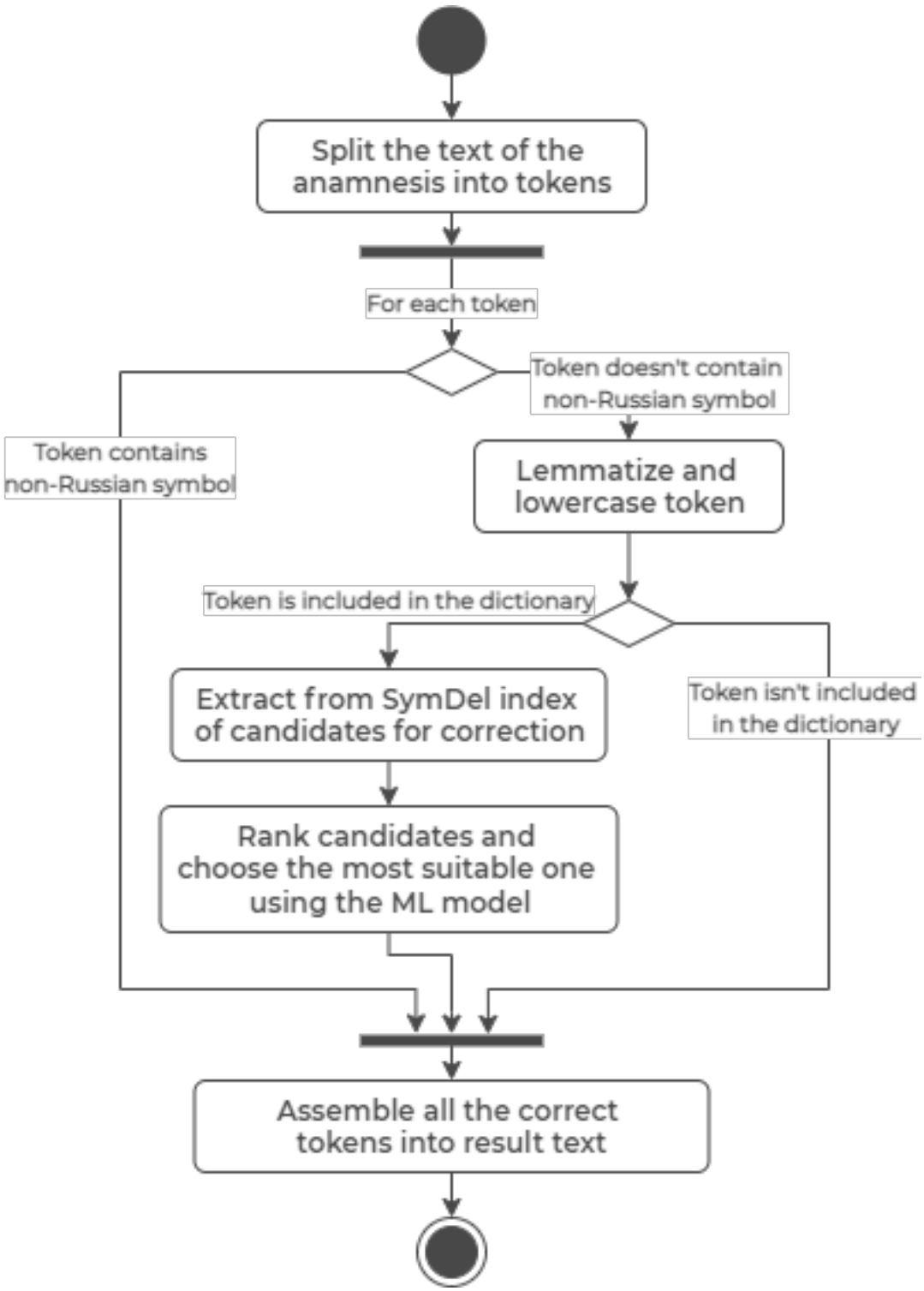

Figure 4.2 — The spelling correction process of the new tool.

token. If there are any, then the token remains untouched and gets into the final result as is. If the token contains only Russian letters, then the token is reduced to lowercase and lemmatized. After that, it is checked whether the token is included in the dictionary of correct words. If it is included, then such a token gets into the final text as it is. Otherwise, a list of candidates is

generated to replace the incorrect word. Then this list is ranked by a special machine learning model and the most suitable candidate gets into the final text. This happens with every token. At the end, the tokens are collected into the final text.

A slightly more complex schema is used to process and correct missing and extra spaces.

To process missing spaces, the following operations are performed. If the word is not contained in the dictionary of correct words, then all the partitions of the word into two parts are found, such that they are contained in the dictionary of correct words. Further, all such found pairs and generated candidates for correcting words for other errors are ranked by the language model. For a pair of words, a metric is calculated for each word and then the average of the two metrics is calculated. As a result, the option with the highest metric is selected and gets into the final text.

To handle extra spaces, a somewhat similar procedure is used. For each pair of words, an attempt is made to combine these two words into one by removing the space between them. If the resulted word is in the dictionary of correct words, then a metric is calculated for this word and it is ranked along with the other candidates for replacing.

Processing cases of missing and extra spaces, despite its external simplicity, requires significant calculations and greatly degrades the performance of the tool. To reduce this effect, it is necessary to develop more efficient algorithms and data structures to handle such cases, as well as more data for training the language model in order to correctly rank such candidates.

Given this, the tool has two modes of operation. The first mode without processing missing and extra spaces is the main one in this work. And the second mode with the processing of cases with a space, test, unfinished and requiring improvement.

In the following sections, the Edit Distance Index and Language Model components will be considered.

4.1 Edit Distance Index

The list of candidates is generated by calculating the Damerau-Levenshtein editing distance between an incorrect word and some correct word. If the distance is less than the preset value, then such a correct word gets into the list of candidates. Usually, the maximum editing distance for candidates is set as one or two. The developed tool supports working with both values of the editing distance. However, with an increase in the maximum editing distance, the number of operations for calculating editing distances also increases, since the words that require processing are much more.

In order to make a list of candidates, it is necessary to calculate the editing distance between the incorrect word and all the correct words from the dictionary, which differ in the number of characters in the word by no more than one. With this approach, at the moment of execution, it is necessary to calculate the editing distance a large number of times, which greatly reduces the performance of the tool when using this approach.

To improve performance, it is necessary to reduce the number of calculations of editing distances in runtime. To do this, it is most logical to use some index, which will avoid a large number of calculations. The most advanced approach to building an index is the approach used in the SymSpell [33] tool, the so-called (Symmetric Deletion) index.

SymDel index allows you to perform all calculations and build an index once at the start of the tool and then use it for the rest of the time.

This index is constructed over the whole vocabulary as follows. For each word in the dictionary, the operation of letter deletion in all possible places is applied. The index contains the words with the initial word after the deletion. When generating the candidates, the operation of letter deletion is also applied to the processed word and the resulting word is searched for in the index. As a result, the initial words found in the index are added to the list of candidates for correction.

Consider an example of constructing index entries for the word «домик» shown in Figure 4.3.

In order to build the necessary entries in the index with the maximum editing distance of one for the word «домик», the following operations happen.

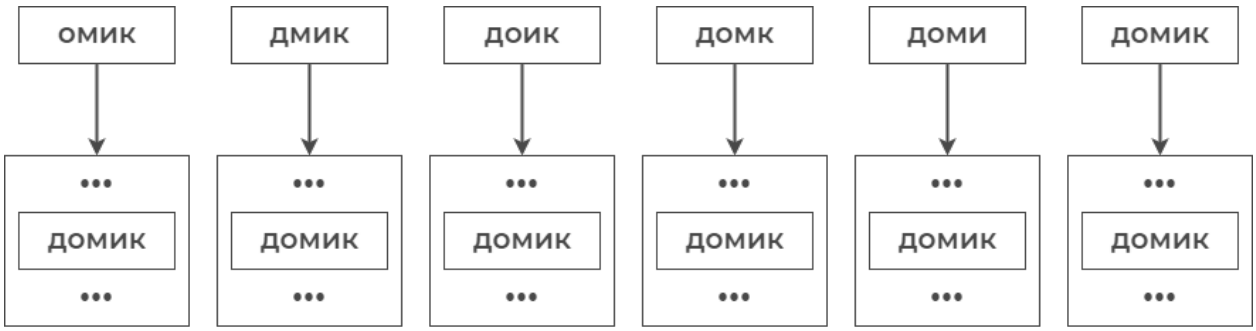

Figure 4.3 — The example of building index entries for the word «домик».

For the word «домик», all possible deletions of one letter from the word are made, as well as the original word itself. Then the produced words after deletion are written to the index and mapped to the original word. Since different words can generate the same delete results after deleting letters, the original words are saved in lists. Thus, each deleted result is mapped to a list of original words from which this delete result can be obtained.

When generating candidates for an incorrect word, all sorts of single letter deletions in the incorrect word also happen. The resulting delete words combined with the incorrect word itself, are searched in the index. As a result, we get a set of found lists of correct words that are candidates for replacement. Combining all these lists, we end up with the final set of candidates.

This generates all edit candidates at a distance of one. For a larger distance, it is necessary to remove the letters accordingly. The algorithm reduces the complexity of edit candidate generation and dictionary lookup by using deletes only instead of all four Damerau-Levenstein edit distance operations, thereby increasing performance many times over.

It's worth noting that increased productivity is not achieved for free. This approach significantly increases the amount of memory required to store the index. Therefore, in the future it will be necessary to build the index on a data structure that allows you to compress the index without significant loss in performance.

At the moment, the index of the developed tool takes up a little more than 4 GB of RAM, which is not a problem for modern computers and servers. However, as the dictionary increases, the amount of memory required will grow non-linearly and may require additional optimizations.

Separately, it should be noted that the developed tool returns the corrected text in a lemmatized form. Lammatization is a common operation for word processing, however, for some exotic tasks this may become a problem, since such tasks may explicitly require the text in its original form without lemmatization. In this paper, lemmatization is used due to the limited amount of data for training a language model. Using lemmatization, it is possible to reduce the number of different words several times by gluing different forms of a word into one initial form. Thus, it is possible to achieve an acceptable quality of the model on a relatively small train corpus.

Next, a machine learning-based language model trained specifically for the task of ranking candidates for the replacement of an incorrect word will be considered in more detail.

4.2 Language Model

A machine learning model based on the BERT architecture is used as a language model for ranking candidates. This approach has proven itself well in similar tasks for correcting spelling errors in various languages, so it was interesting to apply this approach to Russian medical texts. The main advantage of this approach is that the model is able to take into account the context around the incorrect word when ranking candidates, which improves the quality of ranking and accuracy of correction.

As described in the overview of BERT models section 3, in addition to the RoBERTa model, there are several other popular modifications of the BERT model that have their own advantages and weaknesses. However, this paper uses the RoBERTa model, as it has the best benchmarks in many tests and is a good starting point in a large research.

As such a model, a model based on the pre-trained ruRoBERTa-large[1] model from Sberbank is used. To adapt this model to the task of ranking candidates, the model was fine-tuned to 2516 medical anamneses from the V.A.Almazov National Medical Research Center and the corpus of clinical texts of the Institute of Systems Analysis of the Russian Academy of Sciences.

[1]ruRoBERTa-large model on Hugging Faces hub: `https://huggingface.co/sberbank-ai/ruRoberta-large`

Fine-tuning of the model took place using the transformers[1], datasets[2] and accelerate[3] libraries from the Hugging Faces platform. These libraries provide convenient tools for fine-tuning pre-trained models and have integration with the Hugging Faces hub.

The fine-tuning parameters of the model are presented in the Table 4.1

Parameter	**Value**
Train epoch	25
Learning rate	0.00005
Weight decay	25
FP16 training	True
Gradient accumulation steps	64
Per device train batch size	1
Per device eval batch size	1
Gradient checkpointing	True

Table 4.1 — Fine-tuning parameters of the ruRoBERTa-large model.

A special Jupiter Notebook[4] was written to fine-tune the model. Fine-tuning of the model was carried out on the Google Colab[5] platform. To speed up fine tuning, a GPU from the Google Colab platform was used, since more than 16 GB of GPU memory is required to work effectively with such a large model. When fine-tuning, Gradient Accumulation, Gradient Checkpointing and Reduced floating point precision approaches were used, which reduced the required GPU memory for training the model. During fine-tuning, the model reduced perplexity from 10024 to 30 on the test dataset.

[1]Transformers library: `https://github.com/huggingface/transformers`

[2]Datasets library: `https://github.com/huggingface/datasets`

[3]Accelerate library: `https://github.com/huggingface/accelerate`

[4]Jupiter Notebook for fine-tuning model: `https://github.com/DmitryPogrebnoy/MedSpellChecker/blob/main/spellchecker/ml_ranging/models/ru_roberta_large_fine_tune.ipynb`

[5]Google Colab platform: `https://colab.research.google.com/`

5 Tool approbation

As part of the approbation, the tool was tested on a test dataset of single words and on a test dataset of incorrect words with context.

It should be noted that the developed tool is being tested in the mode of processing all types of errors, except for missing and extra spaces. This is described in more detail in Chapter 4. It is also worth noting that the test with single words contains several instances with errors associated with spaces, while the second test consists entirely of instances without errors with spaces.

The first test is the same one on which the open source spelling correction tools were tested in Section 1. The newly developed spellchecker supports CPU and GPU operation. By default, in CPU mode, the language model runs on the CPU and does not require significant graphics resources. If an Nvidia graphics card is available in the system, then the language model switches to the GPU, which significantly improves the performance of the developed tool. The performance tests of the tool on the CPU were carried out on laptop on Ubuntu 20.04 with 16 GB RAM and Intel Core i7-9750H CPU @ 2.60GHz * 12. And the performance tests of the tool on the CPU were conducted on Ubuntu 20.04 with 32 GB RAM, Intel Core i9-10900 CPU @ 2.5GHz * 8 and Nvidia Tesla V100.

The test results for CPU and GPU modes of the developed tool, as well as the results of existing open source tools are presented in Table 5.1.

Tool name	Error precision	Lexical precision	Overall precision	Average words per second
Aspell-python [28]	0.65	0.775	0.7125	353
PyHunspell [29]	0.59	0.49	0.54	11.5
PyEnchant [30]	0.6	0.455	0.5275	26.4
LanguageTool-python [31]	0.64	0.845	0.7425	19.1
PySpellChecker [32]	0.335	0.765	0.55	4.3
SymspellPy [33]	0.42	0.78	0.6	15892.1
SymspellPy (compound) [34]	0.46	0.54	0.5	983.1
Jumspell [35]	0.395	0.925	0.66	2043.2
Spellchecker prototype [2]	0.41	0.83	0.62	0.07
New Spellchecker (CPU)	0.45	0.95	0.7	8.4
New Spellchecker (GPU)	0.45	0.95	0.7	12.5

Table 5.1 — Comparison of metrics on single word test for popular open source spelling correction tools and the new spellchecker.

It is worth noting that the metrics obtained are approximately in the middle in terms of performance and overall precision on single words. In this way, the tool closes the gap between precision but slow and imprecise but high-performance tools.

However, the developed tool, thanks to a language model based on a machine learning model, allows to take into account the context around an incorrect word. The presence of context is a natural condition, since the spellchecker will work with entire medical texts. In order to evaluate the metrics of the new approach and tool on incorrect words with context, a separate test dataset was composed. The dataset is constructed manually from the original anamnesis and contains 60 fragments with medical text. Each fragment contains 10 words, 9 of which are correct, and one with a spelling error.

The results of the test with incorrect words with context are presented in Table 5.2.

Metric	CPU mode	GPU mode
Error precision	0.7	0.7
Lexical precision	0.98	0.98
Overall precision	0.84	0.84
Average words per second	51.5	100

Table 5.2 — Comparison of metrics for popular open source spelling correction tools and new spellchecker.

It should be noted that in the context test, error precision has greatly increased and lexical precision has slightly increased. This effect is explained by the fact that the context of an incorrect word is taken into account by the model for ranking candidates. This makes the ranking more precise and gives the correct result more often.

Thus, the developed tool, when working in context, generally shows better metrics than existing open source tools on single words. The performance of the tool with a ratio of correct and incorrect words as 9:1 is in the middle. It does not meet the requirement of processing 275 words per second and loses slightly less than 3 times when using the GPU. Based on this, it can be conclude that the error correction algorithm and the tool that implements

it, despite the impressive results, requires further improvements, optimizations and experiments.

In the future, it is planned to test this tool on whole anamnesis in order to bring the test data closer to real medical data. At the moment, there is no available non-training real medical data for such testing. It is also planned to conduct tests of existing open source tools on a dataset with context and using GPU.

Conclusion

In the process of this work, the following results were obtained.

1) An overview of the subject area is made. Types of mistakes in texts are described and a list of types of spelling errors to correct in this paper is highlighted. An overview of the different approaches to edit distances is also made. In addition, a general approach to automatic correction of errors in texts is described, and a generalized architecture of such tools is presented. An overview of relevant articles focused on correcting English, Russian, and separately medical texts is also made.

2) The review and testing of existing open-source tools, which have the ability to correct spelling errors in Russian, are made. According to the test results, none of the tools has both sufficient precision and performance metrics. Also based on the available anamnesis the required performance of 275 words per second was specified.

3) A new algorithm for correcting spelling errors in Russian is developed. To generate candidates for correction, the algorithm uses the Damerau-Levenstein distance, and the SumDel index is used to optimize the distance calculation. A fine-tuned model based on the BERT architecture is used to rank candidates.

4) A working prototype of a tool for correcting medical text in Russian implementing a new algorithm is developed. For the developed tool, a model based on ruRoBERTa-base was specially fine-tuned on medical anamnesis.

5) The approbation of the developed tool was carried out. In the test with incorrect words with context, the tool showed error precision 0.7, and lexical precision 0.98. At the same time, the tool showed a performance of 51.5 words per minute on the CPU, and 100 words per minute on the GPU, which is slightly less than the required performance by a third.

This work was presented at the XI Conference of Young Scientists [53], the theses of the work were published [54] in an online collection.

Further development of the tool is planned in the future. One of the directions is to try fine-tuning several other BERT models and evaluate their impact on the precision of the tool. Another way is to develop and implement more advanced algorithm optimizations, as well as transfer critical calculations

to C/C++ code, which will significantly improve performance. In addition, it is necessary to refine the algorithms for correcting errors related to spaces so that their processing reduces the performance of the tool less. These are some of the many further ways of research and development of the tool.

Acknowledgments

I would like to thank Anastasia Funker for her help with obtaining medical data and reproducing the results of previous work.

And I also want to thank my supervisor, Sergey Kovalchuk, who efficiently and diligently assists his students in writing research papers, as well as for his help in gaining access to the GPUs cluster for testing the developed tool.

References

1. *Toutanova, Kristina.* Pronunciation Modeling for Improved Spelling Correction / Kristina Toutanova, Robert C. Moore // Proceedings of the 40th Annual Meeting on Association for Computational Linguistics. — ACL '02. — USA: Association for Computational Linguistics, 2002. — P. 144–151. https://doi.org/10.3115/1073083.1073109.

2. *Balabaeva, Ksenia.* Automated Spelling Correction for Clinical Text Mining in Russian / Ksenia Balabaeva, Anastasia A. Funkner, Sergey V. Kovalchuk // *Studies in health technology and informatics.* — 2020. — Vol. 270. — Pp. 43–47.

3. *Mitton, R.* English Spelling and the Computer / R. Mitton. Real Language Series. — Longman, 1996. https://books.google.cd/books?id=L1ANAQAAMAAJ.

4. *Yannakoudakis, E.J.* The rules of spelling errors / E.J. Yannakoudakis, D. Fawthrop // *Information Processing & Management.* — 1983. — Vol. 19, no. 2. — Pp. 87–99. https://www.sciencedirect.com/science/article/pii/0306457383900456.

5. *Kukich, Karen.* Techniques for Automatically Correcting Words in Text / Karen Kukich // *ACM Comput. Surv.* — 1992. — 12. — Vol. 24, no. 4. — Pp. 377–439. http://doi.acm.org/10.1145/146370.146380.

6. *Pirinen, Tommi A.* State-of-the-Art in Weighted Finite-State Spell-Checking / Tommi A. Pirinen, Krister Lindén // Computational Linguistics and Intelligent Text Processing / Ed. by Alexander Gelbukh. — Berlin, Heidelberg: Springer Berlin Heidelberg, 2014. — Pp. 519–532.

7. *Levenshtein, V. I.* Binary Codes Capable of Correcting Deletions, Insertions and Reversals / V. I. Levenshtein // *Soviet Physics Doklady.* — 1966. — 02. — Vol. 10. — P. 707.

8. *Damerau, Fred J.* A Technique for Computer Detection and Correction of Spelling Errors / Fred J. Damerau // *Commun. ACM.* — 1964. — mar. — Vol. 7, no. 3. — P. 171–176. https://doi.org/10.1145/363958.363994.

9. *Hirschberg, D.* A linear space algorithm for com-puting longest common subsequences / D. Hirschberg // *Communications of The ACM* -

CACM. — 1975. — 01.

10. *Hamming, Richard*. Coding and Information Theory / Richard Hamming. — 1980. — 01.

11. *Winkler, William*. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage / William Winkler // *Proceedings of the Section on Survey Research Methods*. — 1990. — 01.

12. *Mikolov, Tomas*. Efficient Estimation of Word Representations in Vector Space. — 2013.

13. *Bojanowski, Piotr*. Enriching Word Vectors with Subword Information. — 2017.

14. *Joulin, Armand*. Bag of Tricks for Efficient Text Classification. — 2016.

15. *Peters, Matthew E.* Deep contextualized word representations. — 2018.

16. *Pennington, Jeffrey*. Glove: Global Vectors for Word Representation. — 2014. — 01.

17. *Devlin, Jacob*. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. — 2019.

18. *Hládek, Daniel*. Survey of Automatic Spelling Correction / Daniel Hládek, Ján Staš, Matúš Pleva // *Electronics*. — 2020. — Vol. 9, no. 10. `https://www.mdpi.com/2079-9292/9/10/1670`.

19. *Jayanthi, Sai Muralidhar*. NeuSpell: A Neural Spelling Correction Toolkit. — 2020.

20. *Hu, Yifei*. Misspelling Correction with Pre-trained Contextual Language Model. — 2021.

21. *A., Sorokin A.* SpellRuEval: the First Competition on Automatic Spelling Correction for Russian. — 2016. — 06.

22. *Sorokin, Alexey*. Automatic spelling correction for Russian social media texts. — 2016. — 06.

23. *Rozovskaya, Alla*. Spelling Correction for Russian: A Comparative Study of Datasets and Methods / Alla Rozovskaya // Proceedings of the International Conference on Recent Advances in Natural Language Pro-

cessing (RANLP 2021). — Held Online: INCOMA Ltd., 2021. — 09. — Pp. 1206–1216. https://aclanthology.org/2021.ranlp-main.136.

24. *Sorokin, Alexey.* Spelling Correction for Morphologically Rich Language: a Case Study of Russian. — 2017. — 01.

25. Automated misspelling detection and correction in clinical free-text records / Kenneth H. Lai, Maxim Topaz, Foster R. Goss, Li Zhou // *Journal of Biomedical Informatics.* — 2015. — Vol. 55. — Pp. 188–195. https://www.sciencedirect.com/science/article/pii/S1532046415000751.

26. Official website of Aspell tool. — 2000. http://aspell.net/.

27. *Fivez, Pieter.* Unsupervised Context-Sensitive Spelling Correction of Clinical Free-Text with Word and Character N-Gram Embeddings / Pieter Fivez, Simon Šuster, Walter Daelemans // BioNLP 2017. — Vancouver, Canada,: Association for Computational Linguistics, 2017. — 08. — Pp. 143–148. https://aclanthology.org/W17-2317.

28. GitHub repository of Aspell-python — Python wrapper of Aspell tool. — 2021. https://github.com/WojciechMula/aspell-python.

29. GitHub repository of PyHunspell — Python wrapper of Hunspell tool. — 2014. https://github.com/blatinier/pyhunspell.

30. Official website of PyEnchant — Python wrapper of Enchant tool. — 2021. http://pyenchant.github.io/pyenchant/install.html.

31. GitHub repository of language-tool-python — Python wrapper of LanguageTool. — 2021. https://github.com/jxmorris12/language_tool_python.

32. GitHub repository of PySpellchecker. — 2021. https://github.com/barrust/pyspellchecker.

33. GitHub repository of SymSpell tool. — 2018. https://github.com/wolfgarbe/SymSpell.

34. GitHub repository of SymSpellPy — Python wrapper of SymSpell tool. — 2021. https://github.com/mammothb/symspellpy.

35. GitHub repository of JumSpell tool. — 2018. https://github.com/bakwc/JamSpell.

36. GitHub repository of Hunspell tool. — 2014. https://github.com/hunspell/hunspell.

37. Official website of Enchant tool. — 2021. https://github.com/AbiWord/enchant.

38. GitHub repository of Grazie plugin. — 2021. https://github.com/JetBrains/intellij-community/tree/master/plugins/grazie.

39. Official website of Intellij IDEA. — 2021. https://www.jetbrains.com/idea/.

40. Peter Norvig's article about spelling corrector. — 2007. https://norvig.com/spell-correct.html.

41. Official website of Yandex.Speller. — 2021. https://yandex.ru/dev/speller/.

42. Official website of Jumspell. — 2021. https://jamspell.com/.

43. Official website of Webiomed. — 2021. https://webiomed.ai/.

44. GitHub repository of Sberbank AI Lab. — 2021. https://github.com/sberbank-ai-lab.

45. Attention is All you Need / Ashish Vaswani, Noam Shazeer, Niki Parmar et al. // Advances in Neural Information Processing Systems / Ed. by I. Guyon, U. Von Luxburg, S. Bengio et al. — Vol. 30. — Curran Associates, Inc., 2017. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

46. *Radford, Alec.* Improving Language Understanding by Generative Pre-Training / Alec Radford, Karthik Narasimhan. — 2018.

47. *Taylor, Wilson L.* "Cloze Procedure": A New Tool for Measuring Readability / Wilson L. Taylor // *Journalism Quarterly.* — 1953. — Vol. 30, no. 4. — Pp. 415–433. https://doi.org/10.1177/107769905303000401.

48. *Sanh, Victor.* DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. — 2019. https://arxiv.org/abs/1910.01108.

49. *Liu, Yinhan.* RoBERTa: A Robustly Optimized BERT Pretraining Approach. — 2019. https://arxiv.org/abs/1907.11692.

50. *Clark, Kevin.* ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. — 2020. https://arxiv.org/abs/2003.10555.

51. *Yang, Zhilin.* XLNet: Generalized Autoregressive Pretraining for Language Understanding. — 2019. https://arxiv.org/abs/1906.08237.

52. OpenCorpora dictionaries. — 2021. http://opencorpora.org/dict.php.

53. XI Conference of Young Scientists. — 2022. https://kmu.itmo.ru/.

54. Theses 'Automatic spelling correction method for analyzing clinical text in Russian' on XI Conference of Young Scientists. — 2022. https://kmu.itmo.ru/digests/article/8367.