UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

PROIECT DE DIPLOMĂ

Corectarea erorilor gramaticale în limba română

Roșca Nicolae

Coordonator științific:
Sl. dr. ing Ștefan RUȘEȚI

BUCUREŞTI

2023

UNIVERSITY POLITEHNICA OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE DEPARTMENT

DIPLOMA PROJECT

Romanian grammatical error correction

Rosca Nicolae

Thesis advisor:
Sl. dr. ing Ștefan RUȘEȚI

BUCHAREST

2023

CONTENT

Abstract **2**
Sinopsis **3**
1 INTRODUCTION **3**
1.1 Context 4
1.1.1 Grammatical Error Correction 4
1.2 Problem 4
1.3 Obiective 4
1.4 Proposed Solution 5
1.5 Thesis Structure 6
2. STATE OF THE ART **6**
2.1 Literature review 6
2.2 Related Works 8
2.3 Background 8
2.3.1 Token Tagging 9
2.3.2 Levenshtein distance (Wagner–Fischer algorithm) 9
2.3.3 Transformers 10
2.3.4 Bert 11
2.3.5 Roberta 11
2.3.6 XLM-RoBERTa 12
2.3.7 T5 12
2.3.8 mT5 12
2.3.9 Gpt 3.5 12
3.METHOD **13**
3.1 Corpus 13
3.1.1 10M Corpus 13
3.1.2 100k Corpus 14
3.1.3 Golden Corpus 15
3.2. Dataset generation 16
3.3 Training 18
3.4 Bringing it all together 18
4. Testing and Results **20**
5. Conclusions and further work **20**

Abstract

For as long as there has been written language, grammatical errors have been a constant problem. In any written medium, text/internet messages, email, blogs, internet searches etc. errors can make the communication/information retrieval either harder or near impossible.

While in the natural language processing (NLP) circles the task of grammatical error correction (GEC) has made significant progress due to the innovative transformer architecture and the availability of large corrected datasets in English. The same can’t be said about other languages where such data isn’t available. This has given rise to a number of low-resource solutions, such as the use of (neural) language models, data augmentation and synthetic data generation.

This thesis tries to tackle the natural language processing task of grammatical error correction for the Romanian language, using the pre-trained language model XLM-RoBERTa and fine-tuning it (on a token tagging task) to detect errors. Additionally, the pre-trained masked language head is employed to generate alternative suggestions, from which the most likely one is selected. Utilizing the same token tagging technique, we have trained an additional model to indicate the precise positions for the insertion of text of varying lengths. Said text is then generated by the mT5 language model, which was fine-tuned for text unmasking in romanian, similarly to BERT.

The error detection modules have both been trained on a corpus of 10 million incorrect/corrected sentence pairs that have been generated synthetically and on 100.000 sentence pairs generated by the GPT3.5 large language model. This is done by asking it to corrupt grammatically correct text. The whole configuration has been tested on the RONACC corpus made available in “Neural grammatical error correction for romanian” authored by Teodor-Mihai Cotet, Stefan Ruseti, and Mihai Dascalu [1]. We have not reached state of the art for Romanian grammatical error correction but we have seen improved results with the GPT3.5 generated corpus compared to the 10M synthetically generated corpus (it’s important to note that the GPT3.5 models took significantly less resources to train, by a factor of approximately 100).

Sinopsis

De când există limbajul scris, erorile gramaticale au reprezentat o problemă constantă. În orice mediu de comunicare scris, precum mesaje text, e-mailuri, bloguri, căutări pe internet etc., erorile pot face comunicarea/recuperarea informației să fie mai dificilă sau aproape imposibilă.

În domeniul de prelucrare al limbajului natural (NLP), sarcina corectării erorilor gramaticale (GEC) a progresat semnificative datorită arhitecturii inovatoare denumite “transformer” și disponibilității unor seturi mari de date corectate în limba engleză. Nu se poate spune același lucru despre alte limbi unde astfel de date nu sunt disponibile. Aceasta a dus la dezvoltarea unor soluții care au scopul de a funcționa eficient în condiții de resurse limitate. Cateva exemple ar fi: utilizarea modelelor de limbaj (neurale), augmentarea datelor și generarea de date sintetice.

Această teză încearcă să abordeze sarcina de corectare a erorilor gramaticale pentru limba română, folosind modelul de limbaj pre-antrenat XLM-RoBERTa și fine-tuned (prin marcarea tokenurilor) pentru detectarea erorilor. În plus, este folosit “head-ul” pre-antrenat pentru a genera alternative, din care este selectată cea mai probabilă. Utilizând aceeași tehnică de marcarea a tokenurilor, un model suplimentar este antrenat pentru a indica pozițiile unde trebuie inserate fragmente de text de lungime variabila. Acest text este apoi generat de modelul de limbaj mT5, care a fost fine-tuned pentru “unmasking” textului în limba română, într-un mod similar cu BERT.

Modulele de detectare a erorilor au fost antrenate pe un corpus format din 10 milioane de perechi de fraze incorecte/corectate, generate sintetic, și pe 100.000 de perechi de fraze generate de modelul de limbaj GP T3.5. Întreaga configurație a fost testată pe corpusul RONACC pus la dispoziție în lucrarea "Corecția neurală a erorilor gramaticale pentru limba română" scrisă de Teodor-Mihai Cotet, Stefan Ruseti și Mihai Dascalu [1]. Nu am atins “state of the art” pentru corecția erorilor gramaticale in limba română, dar am obținut rezultate îmbunătățite cu corpusul generat de GPT 3.5 în comparație cu corpusul generat sintetic de 10 milioane (este important de menționat că modelele GPT 3.5 au necesitat semnificativ mai puține resurse pentru antrenare, cu un factor de aproximativ 100).

1 INTRODUCTION

1.1 Context

The process of pinpointing and rectifying grammatical issues within an erroneous statement is referred to as grammatical error correction, commonly abbreviated as GEC. All types of grammatical errors, including misspellings, improper use of articles, prepositions, pronouns, nouns, etc., as well as weak sentence structure, can be included in this list. Grammar checkers are used in a variety of applications: email, programs, text editors, messages, and others. One key point is that many downstream tasks are dependent on the grammaticality of the input, making this task especially important.

There are multiple approaches to this task: classifiers (statistical and neural), machine translation (statistical and neural), edit based approaches, language models etc. [2]

GEC encounters distinct challenges in both extensive data and scarce data scenarios. Coping with large datasets in the big data context necessitates robust infrastructure, data quality assurance, and extended model training time. Conversely, low data scenarios present limited training examples, data sparsity, and a heightened risk of overfitting.

1.2 Problem

Generally speaking, individuals frequently commit numerous errors in their writing. This can arise from factors such as being limited in their understanding of the language's syntactic and grammatical structure, or simply being inattentiveness and haste. Regardless of the underlying cause, it's nearly inevitable that grammatical errors will be encountered in virtually any form of written communication.

Besides GEC being a complex problem in of itself the difficulties compound from a lack of high quality training data. In languages like Romanian, with limited online presence, addressing the shortage of training data is crucial for our solution.

Another problem is the trade off between fully correcting a sequence, and making granular changes. On one hand, fully correcting a sequence is easier to implement, through the use of sequence to sequence models that turns the GEC task to a machine translation task. On the other hand, a more granular strategy would be more human friendly for something like a text editor. One approach for the ladder style of correction would involve an user’s (or perhaps another model’s) ability to choose from an array of plausible corrections.

1.3 Obiective

In this thesis we want to develop a robust GEC system, that is modular in nature, can adapt to low-resource languages (Romanian in this case), and provides a balance between full correction and granularity. The system should be capable of handling error from various sources, including but not limited to: lack of knowledge, inattentiveness, or non-native language use. We intend to evaluate our results against a standard corpus, as introduced in the previous paper [1], to benchmark the performance of our system.

1.4 Proposed Solution

Our solutions primarily leverage the token annotation task to identify which tokens should be deleted or modified, and where to insert tokens. The goal, through the use of these operations, is to transform the incorrect sentence to the corrected variant. For this our system builds upon previous work. It uses both data generated using text corruption that was made available by COTET Et al. [1] and data generated using the GPT-3.5 model [3] and prompt engineering. We have named the first dataset 10M (after the total number of pairs, 10 millions) and the second one 100k (100.000 pairs).

We use a tokenizer and 4 models: the XLM-roBERTa tokenizer for pre-processing inputs, XLM-roBERTa-large for the Mask-Fill task, two XLM-XLM-roBERTa-base models using the Token Tagging head [4], where the first is finetuned to detect if a certain token should be deleted or modified (called from now on the MOD/DEL model) and the other if at a certain position there should be inserted a varied length sequence (called the ADD model). The last is a mT5 model fine-tuned for generating sequences of varied length.

We preprocess the data by taking pairs of sentences (incorrect and corrected) tokenizing them and passing them through a modified Wagner–Fischer algorithm to detect the operation necessary to transform the incorrect sentence to the corrected variant. Using this intermediary form we create 5 datasets: the MOD/DEL dataset (10M and 100k), the ADD dataset(10M and 100k) and the mT5 unmask dataset(10M).

For a grammatically incorrect sentence, we begin by tokenizing it using the XLM-RoBERTa tokenizer. Next, the token sequence is simultaneously inputted into both the MOD/DEL model and, if enabled, the ADD model.
Using the tags generated by the MOD/DEL model we delete the tokens that are marked for deletion and we replace the tokens that are marked for replacement with a <mask> token. We then input the new token sequence into the mask fill model, and get the topK replacements. In this implementation the most likely token is always used. If the ADD model is activated, it results in determining the specific location within the sequence where a

variable-length string needs to be inserted. This string is subsequently generated by the fine-tuned mT5 model.

1.5 Thesis Structure

The thesis is split into 5 parts: introduction, state of the art, method, testing (and results) and finally conclusions.

In the state of the art we are going to review, from a bird's-eye view, the current approaches for solving GEC spanning both data-rich and data-scarce languages. In the same chapter we will give a background for the technologies that we have used, and our rationale behind their selection.

Transitioning to the "method" chapter, we delve into the practical implementation of our chosen models, and the generation of the training data for said models.

In the testing and results we showcase the outcomes of our testing on the RONACC corpus and subsequently draw a comparison with the findings from COTET, Et al. [1].

In conclusions we reflect on our solution, its strong and weak points, and what could be further improved.

2. STATE OF THE ART

In this section, we delve into the current state of the art for grammatical error correction. We begin by conducting a comprehensive literature review (2.1) to examine the existing body of knowledge and research in this domain. Following this, we explore related works (2.2) that complement or extend the findings from the literature review, and we offer essential background information (2.3) for the building of our solution.

Within the 'Background' subsection (2.3), we dissect crucial elements that underpin our research, offering a concise overview of token tagging (2.3.1) as a fundamental concept. Subsequently, we delve into the Levenshtein distance, particularly the Wagner–Fischer algorithm (2.3.2), an essential mathematical tool central to our investigation. Additionally, we provide further insight into the role of Transformers (2.3.3), followed by specific Transformer models such as BERT (2.3.4), RoBERTa (2.3.5), XLM-RoBERTa (2.3.6), T5 (2.3.7), mT5 (2.3.8), and GPT-3.5 (2.3.9).

2.1 Literature review

We initiate our literature review with the work authored by BRYANT, Christopher, et al. [2], titled “Grammatical error correction: A survey of the state of the art”. This paper serves as the core for our review, offering a comprehensive synthesis of research up to the present day.

The core approaches incorporate: classifiers (both statistical and neural), machine translation (statistical, neural), edit-based methodologies, language models and low-resource scenarios.

Classifiers are a type of machine learning or statistical model that is used to both identify and classify grammatical errors in a given input text. The main reason for their use is that the set of common mistakes for non-native English speakers is relatively limited. Consequently, they lend themselves well to multiclass classification. Some popular models for classifiers are: naive Bayes, maximum entropy, decision trees, support-vector machines and the averaged perceptron.

In machine translation the text is treated like a translation problem, from the base “ungrammatical” language to the correct form of said language. The advantages of this approach is the limited amount of expert knowledge or feature engineering needed and the simultaneous correction of all classes of error.

Statistical machine translation learns from parallel corpora, aligns sentences, and estimates translation probabilities to determine how words and phrases are translated. During translation, it selects the most likely translations based on these probabilities.

Neural machine translation takes from the advances in deep learning of sequence to sequence tasks and applies it to GEC. It uses a single language model, which is based on the encoder-decode framework. The architecture of language models has evolved over the years, going from Recurrent Neural Networks and Convolutional Neural Networks to the Transformer architecture [5].

The edit based approach generates a list of edits that when applied to the inputted incorrect sentence it gets converted to the theoretical corrected pair. The advantage for this approach is that it’s often faster to generate the edits, rather than the whole sentence. The downside is that this approach tends to be limited only to the token level, and doesn’t always capture more complex edits.

In low-resource scenarios language models are often used due to their lack of need for parallel data sets, as they are pre-trained on a substantial corpus of data, focusing on tasks aimed at equipping the models with an understanding of the underlying syntax and semantics of the language(s). These can be used as discriminators (where low probability sentences are more likely to be erroneous), as generators (where models are prompted to correct the imputed wrong sentence) or as the base for more complex solutions. Another common approach to low-resource scenarios is the use of synthetically generated data.

2.2 Related Works

In the subsequent subsection we present some remarkable approaches and results, both in resource rich and poor languages. GEC solutions are benchmarked on various datasets that are often split into erroneous input and corrected output. The objective of a given model is to rectify as many of the former sentences to align with the latter. The most commonly used dataset are: the CoNLL-2014 Shared Task [12], the JFLEG [13], the BEA Shared Task - 2019 [14] and the WI-LOCNESS test sets.

The current state of the art on the CoNLL-2014 Shared Task is a paper titled “Frustratingly Easy System Combination for Grammatical Error Correction” by QORIB Et al.e with an impressive F0.5 score of 69.51. The authors present their approach, named ESC (Edit-based System Combination), for combining the outputs of multiple base Grammatical Error Correction (GEC) systems. They formulate GEC system combination as a binary classification task, extracting edits from base systems' hypotheses. These edits are then represented by feature vectors. A logistic regression model is employed to predict the probability of each edit being correct. Conflicts among multiple edits are resolved by post-processing and selecting those with probabilities above a threshold and ensuring no overlapping or conflicting changes. The ESC method focuses on edit types and their sources in different hypotheses to generate a final, improved GEC output.

On the WI-LOCNESS low resource track where the current state of the art is the paper titled “Neural Grammatical Error Correction Systems with Unsupervised Pre-training on Synthetic Data” by GRUNDKIEWICZ Et al. [16], it achieves an F0.5 score of 64.24.
The authors employ customized Transformer models for GEC tasks, implementing various refinements to enhance performance. These refinements include robust regularization techniques to combat overfitting, increasing mini-batch sizes, and pre-training the entire model using synthetic parallel data. They introduce a novel approach to generate synthetic training data, emphasizing the replication of genuine error patterns by substituting words within predefined confusion sets. Their ensemble technique combines sequence-to-sequence models with a language model, and they introduce a re-ranking method using right-to-left models to further enhance correction capabilities.

2.3 Background

In the following subsection we present the foundational technologies, techniques and approaches for our solution.

2.3.1 Token Tagging

Token tagging is a fundamental task in the field of natural language processing (NLP), essential for understanding and processing textual data effectively. In this task, a sequence of tokens, typically derived from an input sentence using a tokenizer, is annotated with specific tags or labels that convey crucial linguistic or semantic information. The primary goal is to assign each token a tag that reflects its role or meaning within the context of the sentence.

Token tagging finds applications in various NLP subfields, including Named Entity Recognition (NER), Part-of-Speech (PoS) tagging, and syntactic chunking, among others.
We have taken a keen interest in it because of its theoretical capability to identify erroneous tokens by tagging them, which forms the foundation of our solution

2.3.2 Levenshtein distance (Wagner–Fischer algorithm)

The Levenshtein distance[10] represents the number of basic operations (the 2 main ones being “add” and “delete”, but can include others like “modify” or “swap”) that can be applied to a source string to convert it into the target string. The pseudocode definition for the Levenshtein distance is:

```
int lev(a, b){
    if len(b) == 0:
        return len(a)
    if len(a) == 0:
        return len(b)
    return 1 + min( lev(tail(a), b), lev(a, tail(b)), lev(tail(a), tail(b) )
}
```

There exist several implementations for the Levenshtein distance. The naive implementation (above) poses challenges due to its recurrent calls to the same substring. As such, the implementation has a time complexity of O(3^min(m, n)), exponential time, due to its branching logic.

The Wagner–Fischer algorithm [11] is a dynamic programming implementation of the naive algorithm, where all the distances between all the prefixes of the source and target string are stored in a (n + 1) X (m + 1) matrix. This matrix is built iteratively, and each cell is calculated, with the final result residing in the last cell, M[n, m], representing the actual distance between the two strings.

Within this matrix, each individual cell signifies the distance between the initial i characters of one string and the initial j characters of the other string. As we traverse the matrix from the top-left corner (which denotes empty strings) to the bottom-right corner (signifying the complete source and target strings), we keep track of distances between various prefixes.

Our computation at each cell revolves around three fundamental operations: insertion, deletion, and modification. For insertion, we assess the distance at M[i, j-1] and add 1 to account for inserting a character from the target string into the source string. For deletion, we consider the distance at M[i-1, j] and add 1 to account for deleting a character from the source string. Lastly, for modifying, we inspect the distance at M[i-1, j-1] and add either 1 (if the characters at positions i and j in the source and target strings differ) or 0 (if they match).

An example of the Wagner–Fischer matrix for the “man” and “maaan1” pair (3 deletions):

		m	**a**	**a**	**a**	**n**	**1**
	0	1	2	3	4	5	6
m	1	0	1	2	3	4	5
a	2	1	0	1	2	3	4
n	3	2	1	1	2	2	3

We have chosen the basic operations to be add, delete and modify, as more complex operations (like swapping) are less frequently encountered when working with tokens. We are placing particular emphasis on the "modify" operation, as it can be used by the various models within the BERT family with the unmasking head to generate plausible alternatives.

2.3.3 Transformers

The NLP landscape has been radically changed by the groundbreaking paper Attention Is All You Need [5] which introduced the transformer architecture. What makes this architecture especially powerful is the parallelization capability, leveraged by the self-attention mechanism. This allows the model to be trained significantly fastester compared with something like the Recurrent Neural Network (RNNs) and Long Short-Term Memory (LSTM) which are sequential models. Besides these, the transformer can easily capture long range dependencies, a feature that was missing in the RNN and was attempted with the LSTM.

The encoder-decoder model for machine translation was given a performance boost by the addition of the attention mechanism. By linearly combining all the encoded input, with the vectors receiving the highest weights, the attention mechanism was designed to enable the decoder to employ the most relevant portions of the input sequence in a flexible manner.

2.3.4 Bert

BERT (short for Bidirectional Encoder Representations from Transformers) [6] is a pre-trained language model built upon the Transformer architecture. Its main selling point is the fact that it (as the name suggests) for a given token it takes into account both the left and the right context which allows the model to capture dependencies and relationships that would be hard to detect from an only left to right context are now exposed to the model. The way BERT keeps track of the order of the input tokens is by augmenting the token embeddings with positional embeddings.

The base models are pre-trained on two downstream tasks aimed at enhancing language comprehension: masked language modeling and next sentence prediction.

In the "masked language modeling" task, the primary objective is to predict the correct token within a sentence, wherein one of its tokens is substituted with a special token (denoted as [MASK]). In the "next sentence prediction" task, the model is presented with two sentences, and its challenge lies in determining whether the second sentence seamlessly follows the first.

Bert and its language understanding can be leveraged for a wide array of downstream tasks using fine-tuning. The base model is *mostly* unchanged and the task-specific head is where most of the parameter adjustments happen. Using this approach it is way more cost effective compared to restraining the whole model. We have chosen the BERT family of models because of the presence of the masked language head, which allows our solution to generate correct tokens given erroneous ones.

2.3.5 Roberta

RoBERTa (Robustly Optimized BERT Pretraining Approach) [7] is a language model developed by the Facebook Ai team based on the BERT in which the model was designed to improve upon the pre-training methods. It trained for longer, with bigger batches and over more data. The next sequence prediction is removed, the sequences are longer and the masking pattern is dynamically changed.
RoBERTa's achievements are reflected in its performance on various benchmarks and natural language understanding tasks. It consistently outperforms BERT and has set new standards in several domains, including text classification, question answering, and language understanding.

2.3.6 XLM-RoBERTa

XLM-RoBERTa (also named XLM-R) [4] is an evolution of RoBERTa, in which the model is trained on 100 languages, while maintaining state of the art on the GLUE and XNLI benchmark. The decision to adopt this model was guided by our hands-on experience on mask fill task in the Romanian language and maintaining competence in languages that might be used in native day to day conversation, like English. This has been brought up in COTET Et Al. [1] and we have taken this into account when choosing the model.

2.3.7 T5

The T5 model (Text-to-Text Transfer Transformer) [8] is a sequence to sequence model that unifies multiple NLP tasks, aiming to facilitate the transfer of knowledge. Built upon the transformer architecture, it initially undergoes pre-training on the C4 dataset (Colossal Clean Crawled Corpus) using a masking objective similar to BERT. Subsequently, it further hones its

capabilities through a sequence of tasks, in the hope that learning transfers from one task to another.
The unsupervised objective consists of a sentence that has random spans of text obfuscated, and the model is tasked with guessing the right spans.
What makes this especially useful is this approach’s inherent flexibility. It allows for an arbitrary number of text spans to be concealed within the sentence, and equally significant, it accommodates text spans of varying lengths to be hidden.

2.3.8 mT5

mT5 [9] is the multilingual variant of the T5 model trained on 101 languages. It fixes a long standing issue with cross language models where the resulting output would be in another language than the input. We selected this model for its ability to generate sequences of arbitrary lengths, as well as its proficiency in generating text in both Romanian and English, owing to its multilingual training. In our experiments, we observed that text generation in Romanian lacked fluency, prompting us to fine-tune its span masking capabilities using the 10M corpus.

2.3.9 Gpt 3.5

GPT 3.5 is an improved version of the sequence to sequence GPT-3 model, developed by OpenAI in BROWN, Tom, et al. Language models are few-shot learners [3]. GPT-3.5 is an improved version, with impressive life-like results stemming from the use of Reinforcement Learning from Human Feedback (RLHF). We have chosen to use GPT-3.5 due to its state of the art results in all NLP domains, and its availability to the general public for a fee.

3.METHOD

In this chapter, we provide a comprehensive account of the finished system (in both its base form and with the added ADD module) and detail the data generation procedures that facilitated the training of said system.

3.1 Corpus

As the foundation for our project, we have employed the corpus made available by COTET Et Al. [1] for both training and testing. We have used the 10 million synthetic sentence pairs to generate the training data for the MOD/DEL model, the ADD model and the mT5 unmasking model. We then tested our models on the RONACC in the same way it was tested in the paper, but only on the written phrases partition (W-sentence). We have also generated a custom corpus, using the first 100k correct sentences from the 10M corpus and corrupting them using the GP3.5 API, using a prompt engineering approach, making sure that in 20% of cases the phrase remains unchanged, in 60% there is added one mistake (at least that's what we have prompted the GPT-turbo bla bla bla) and in 20% we have prompted the model to create multiple mistakes. We then have manually combed the corpus for any egregious corruption, deleting any sentences that have under 50% Levenshtein similarity.
All of our corpora have been filtered so all inputs have a length of maximum 128 tokens, because of performance concerns.

3.1.1 10M Corpus

The 10M corpus, as described in COTET Et Al. [1] is made using the rule based corruption using as base phrases the romanian wikipedia. Some examples would include:

Original Sentence	Corrupted sentence
Rocarta este o enciclopedie în format electronic care conține articole legate de România, Republica Moldova și români.	Rocarta este o enciclopedie în format electronica care conține articole ee România, Moldova Repuțlica și români.
Ediția din 2000 conține aproximativ 3.000 de articole și peste 3.300 de imagini.	Ediția din 2000 conține uproximativ 3.000 de articole și peste 3.300 de imagini.
Enciclopedia a fost alcătuită de Bogdan Pătruț.	Enciclopedic na fost alcătuit ude Bogdan Pătrat.

We have noticed (unfortunately too late, after we have trained our 10M models) that some phrases (estimated between 25% and 30%) have unnatural corruptions and we have theorized that these phrases significantly impact the model's performance.

Original Sentence	Overly corrupted sentence
Terra nu are alți sateliți naturali în afară de Lună.	Tekra science%20citation%20index%2095%25%20in%20english nut are alți sateliți naturalu în afara de Lună.
Propunere pentru rubrica Știați că...?	Propunere pentru rubrica Ștxați...konrad că?

3.1.2 100k Corpus

Through our experiments, we've trained a model on the golden corpus and achieved remarkable performance given its size (10k). Unfortunately, we couldn't utilize the golden corpus for training because we needed it for testing. To simulate lifelike corrupted text, we explored an alternative approach. Using the OpenAI API, we generated 100k pairs of phrases as described earlier.
In our corpus, we preserved 20% of the sentences without any alterations, introduced a single error in 60% of them, and deliberately induced multiple errors in the remaining 20%. The GPT-3.5 API allows us to provide a prompt to the model to ensure it adheres to a specific character or style. To align the GPT-3.5 model with our intended approach for text corruption, we employed the following model prompt in Romanian:

> "Esti un sistem REALISTSIC de corupere al frazelor in limba romana. Scopul tau este sa produci greseli gramaticale plauzibile”

For text generation, we employed the following prompts:

> “Creeaza O SINGURA greșeală gramaticala sau ortografică pentru următoarea fraza. NU OFERI NICIO EXPLICAȚIE!!!
>
> Text introdus: {original text}
> Text corupt:”

and

> “Creeaza MULTIPLE greșeli gramaticale și ortografice pentru următoarea fraza. NU OFERI NICIO EXPLICAȚIE!!!
>
> Text introdus: {original text}
> Text corupt:”

Here we have some examples of the generated phrases:

Original sentence	Corrupted (single corruption) sentence
Trupele americane părăseasc Cuba.	Trupele americane părăesca Cuba.
Traian începe a doua sa campanie de cucerire a Daciei.	Traian începe a două sa campanie de cucerirea Daciei.
Peste 1.500 de pesoane au fost ucise în luptele de stradă dintre armată și populație.	Peste 1.500 de pesoane au fost ucise în luptele de stradă dintre armarta și populație.

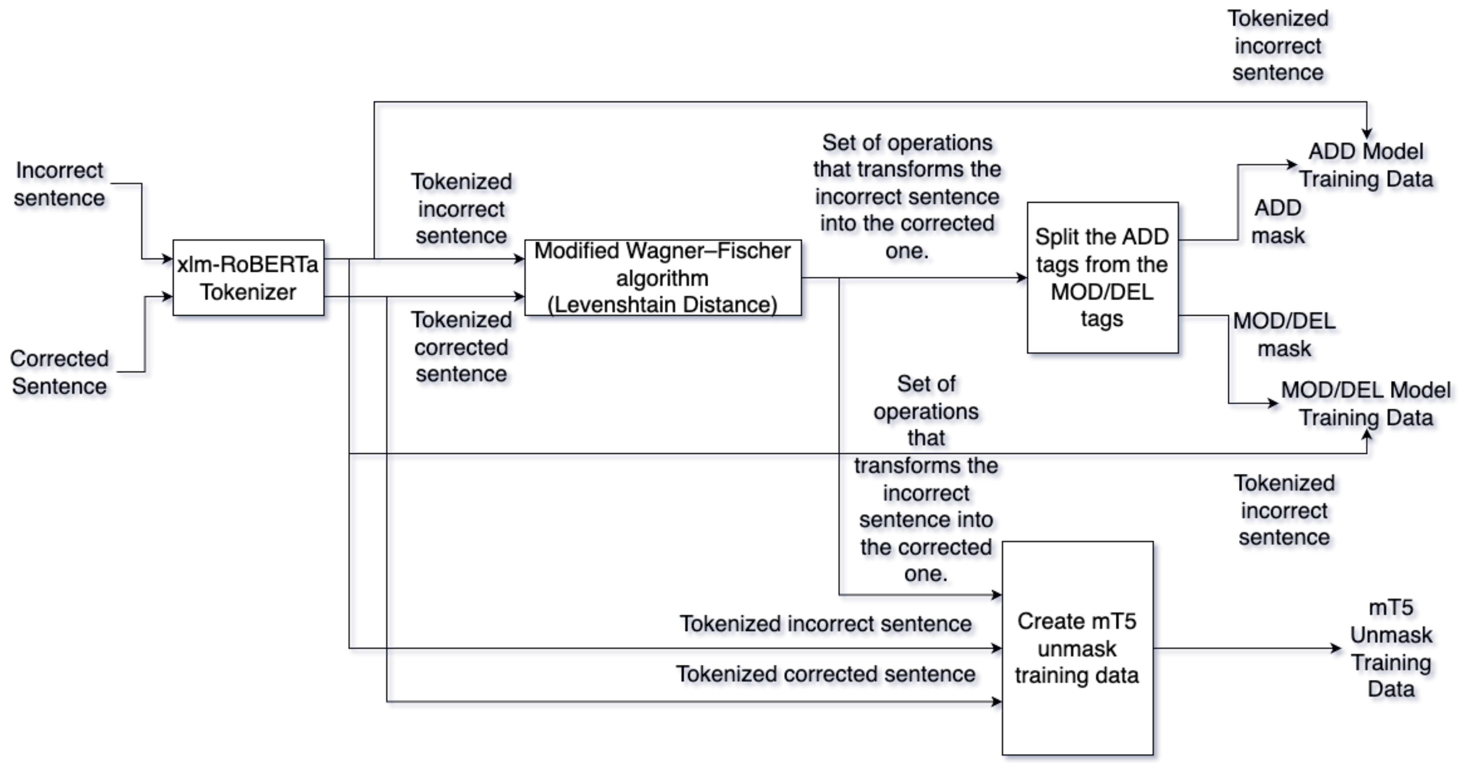
Original sentence	Corrupted (multiple corruption) sentence
Jean Calvin a murit la 27 mai 1564 și a fost îngropat într-un mormânt simplu și anonim, într-o parte a Genevei, așa cum a cerut el.	Cean Calvin a murir la 27 mai 1564 și au fost îngrobați într-un mormânt simpu și anonim, într-o parte a Genevi, așa cum a cerut ui.
Într-adevăr nu a fost cea mai inspirată idee a mea.	Într-adevăr nu a fost ea mai inspirate idee a mea.
Țăranii și sanculoții n-au fost mulțumiți cu ce au obținut de pe urma revoluției.	Țăranii și sancoloții náu fost mũltumiți cu ce +au obținût de pe urmà revoluția.

While this technique shows significant potential, we've observed significant downsides to it. It tends to make almost “made up” errors, something no real human would actually make in real-world conditions, which would naturally result in a decrease in performance.

3.1.3 Golden Corpus

We conducted experiments using the RONACC corpus and performed our tests on the W-Sentence split, comprising grammatically correct sentences with written mistakes. We did not assess the NAC-phrases or NAC-sentences because our training data exclusively consisted of well-formed written sentences. This training bias is evident in the model's tendency to transform phrase fragments into well-structured phrases.

3.2. Dataset generation


Incorrect sentence
Corrected Sentence
xlm-RoBERTa Tokenizer
Tokenized incorrect sentence
Tokenized corrected sentence
Modified Wagner–Fischer algorithm (Levenshtain Distance)
Set of operations that transforms the incorrect sentence into the corrected one.
Split the ADD tags from the MOD/DEL tags
Tokenized incorrect sentence
ADD Model Training Data
ADD mask
MOD/DEL mask
MOD/DEL Model Training Data
Set of operations that transforms the incorrect sentence into the corrected one.
Tokenized incorrect sentence
Tokenized incorrect sentence
Tokenized corrected sentence
Create mT5 unmask training data
mT5 Unmask Training Data

We created five training datasets for our models using 3 distinct formats(MOD/DEL, ADD and mT5). Four of these datasets are designed for token tagging—two were trained on the base 10M dataset, and two were trained on the 100k dataset. Additionally, we have one seq2seq dataset, which was built using the 10M base dataset.

The base for all of our datasets is a pair of sentences: one that is incorrectly formed and its corresponding corrected version. For the following section we will use the pair

("Salut faci tu? Cum o mai druci", "Salut ce faci? Cum o mai duci")

named from now on (incorrect, correct) as examples for all of our data generation procedures.

Next, we tokenize both sentences using the XLM-RoBERTa tokenizer, which includes both a start and an end token. We get the following list of tokens:

incorrect: [0, 48721, 405, 10965, 32, 7140, 36, 409, 115, 318, 2]

correct: [0, 48721, 10965, 370, 32, 7140, 36, 409, 26223, 318, 2]

These tokens map to the initial sentences in the following way:

0:<s> 48721:Salut 10965:faci 370:tu 32:? 7140:Cum 36:o 409:mai 26223:dru 318:ci 2:</s>

0:<s> 48721:Salut 405:ce 10965:faci 32:? 7140:Cum 36:o 409:mai 115:du 318:ci 2:</s>

The start and end tokens (0=<s> and 2=</s>) are vital, particularly when adding text to the sentence's end. Without the end token, adding text would be impossible.

We use the modified Wagner–Fischer algorithm algorithm to generate a list of operations that transforms the incorrect sentence to the corrected one. The main addition over the standard Wagner–Fischer algorithm is that each cell in the matrix contains not only the count of operations required to translate a specific prefix to another but also the previous prefix matrix coordinates from which it was derived.

We start from the final cell of the matrix, at M[n, m] and we iteratively compare the current prefix with the one from which it was formed from, and from their difference we can infer what operation was applied.
This leads to a list of operations that, when applied to the source sentence, yield the target sentence.

> [0, 0, 3, 0, 2, 0, 0, 0, 0, 1, 0, 0] where 0 is the “do nothing operation”, 1 is “modify”, 2 is “delete” and 3 is “add”.

This list is the base for all of our datasets.

Trimming the add operations, and packaging it alongside the tokenization incorrect sentence, yields the MOD/DEL dataset. The final form of our above example for the MOD/DEL dataset is:

```
{
        “input_ids”: [0, 48721, 405, 10965, 32, 7140, 36, 409, 115, 318, 2],
        “attention_mask”: [1, 1, 1, 1 , 1, 1, 1, 1, 1, 1, 1],
        “labels”: [0, 0, 0, 2, 0, 0, 0, 0, 1, 0, 0]
}
```

If we replace the mod and del operations with 0, and map all consecutive add operations to a single operation and couple this with the tokenization incorrect sentence we have the ADD dataset. Similarly to above we have:

```
{
        “input_ids”: [0, 48721, 405, 10965, 32, 7140, 36, 409, 115, 318, 2],
        “attention_mask”: [1, 1, 1, 1 , 1, 1, 1, 1, 1, 1, 1],
        “labels”: [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
}
```

The generation process for the mT5 dataset follows a different approach. Given the inherent nature of mT5 as a seq2seq model, it necessitates both of the input and output to be string or arbitrary size. The input will be a clean sentence with some parts of it masked out, and the output are the masked tokens. The goal of the system is, for a given input masked sentence, to generate the most plausible missing tokens.
Initially, we identify and extract tokens marked with "add" from the operation list in the target sentence. Subsequently, we employ the mT5 mask ("<extra_id_{n}>," where 'n' varies

between 0 and 99) to mask the positions from which these tokens were extracted. The output will be the extracted tokens, formatted like this: <extra_id_0> first group of tokens to generate <extra_id_1> second group of tokens to generate <extra_id_2>...

For our above example the resulting mT5 dataset would be:

```
{
        "input":"Salut<extra_id_0> faci? Cum o mai duci",
        "output":"<extra_id_0> ce<extra_id_1>"
}
```

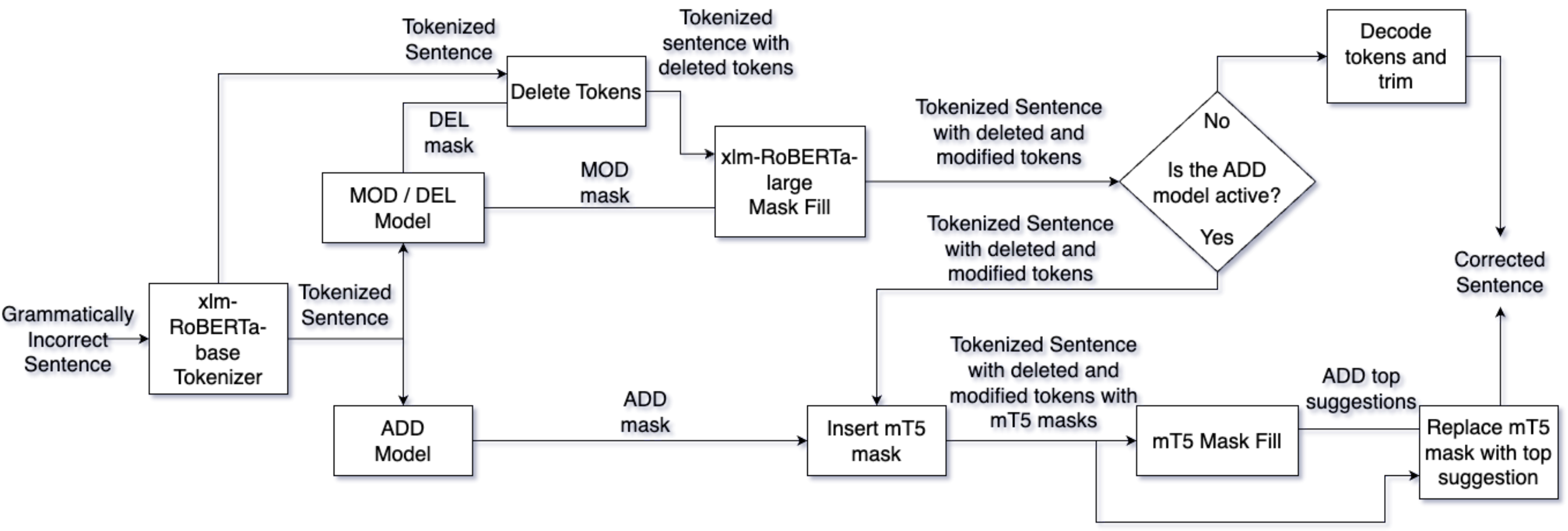
3.3 Training

We have trained the models using the Hugging Face framework. For all of the XLM-R models we have used the “base” version, to maintain a balance between performance and training costs. We used the following hyperparameters as our training arguments:

```
{
   evaluation_strategy = "epoch",

   learning_rate=2e-5,

   per_device_train_batch_size=64,

   per_device_eval_batch_size=64,

   num_train_epochs=2,

   weight_decay=0.01,

   save_total_limit = 1,

   save_strategy="epoch",
}
```

The number of epochs has been limited to 2-3 due to performance constraints and to reduce overfitting on the training data.

3.4 Bringing it all together


Grammatically Incorrect Sentence
xlm-RoBERTa-base Tokenizer
Tokenized Sentence
Tokenized Sentence
Delete Tokens
DEL mask
MOD / DEL Model
MOD mask
Tokenized sentence with deleted tokens
xlm-RoBERTa-large Mask Fill
Tokenized Sentence with deleted and modified tokens
Is the ADD model active?
No
Yes
Decode tokens and trim
Corrected Sentence
Tokenized Sentence with deleted and modified tokens
ADD Model
ADD mask
Insert mT5 mask
Tokenized Sentence with deleted and modified tokens with mT5 masks
mT5 Mask Fill
ADD top suggestions
Replace mT5 mask with top suggestion

Leveraging the fine-tuned models from the aforementioned generated corpora, we are now ready to construct our grammatical error correction system.

The initial step involves tokenizing the grammatically incorrect sentence. It is imperative to note that both our MOD/DEL and ADD models exclusively accept input in the form of token lists. Following tokenization, the resultant tokens are subsequently processed through the MOD/DEL model. This results in a token mask, where on the 0 tag a token is correct, on the 1 tag it should be modified, and on the 2 tag it should be deleted.

The process proceeds as follows: Initially, tokens marked with "2" undergo deletion, accompanied by the concurrent updating of all associated masks, including both the MOD/DEL and ADD tags specific to the affected index.

Subsequently, tokens designated with "1" are replaced with the "<mask>" token and presented to the XLM-R large model, from which the top alternative is chosen as the token replacement.

If the ADD model is inactive, the final tokens are trimmed (removing start and end tokens) and refined, resulting in the corrected sentence.

However, when the ADD model is active, tokens are strategically inserted into this model, producing a tag mask. A tag located at a specific token's position indicates that immediately preceding that token, the mT5 mask should be inserted.
To accomplish this, we utilize tokens from the preceding steps, introducing a custom token (e.g., "<extra_id_0>"), which is then processed through the mT5 model to generate the

required sentence components. These generated strings are seamlessly integrated into the original sentence, resulting in the corrected version.

4. Testing and Results

In this chapter we will analyze our results and compare them to the results presented in [1]. We have used the F0.5 metric, which values precision twice as much as recall, being the standard in testing GEC.

We categorized our models in two distinct manners. The initial categorization depends on whether the model has the ADD module activated, which also implies the presence of the mT5 module. The second categorization is based on whether the MOD/DEL and ADD models were trained on the 10M corpus or the 100K corpus. Consequently, we have four distinct final models: the 10M-base, 10M-add, 100K-base, and 100K-add models.

Our rationale for this was that the addition of the ADD module could improve performance in some cases, but also be detrimental to it (as it could easily generate plausible yet irrelevant alternatives).

We evaluated our dataset using the Romanian version of ERRANT (foot note), which generated the required edits to transform the original incorrect sentences into their corrected counterparts. These are referred to as "original-golden" for the accurate, base edits, and "original-predicted" for the model's predicted revisions. By comparing the original-golden and the original-predicted edit lists, we can calculate the percentages of True Positives, False Positives, and False Negatives, which enables us to calculate the F0.5 score.

In our evaluation, a True Positive is identified when an edit appears in both the original-golden and original-predicted edit lists. Conversely, a False Positive occurs when an edit is found in the original-predicted list but not in the original-golden edit list. Similarly, a False Negative arises when an edit is absent from the original-predicted list but is present in the original-golden edit list.

We have tested on the RONACC corpus presented in COTET Et Al. [1], and more specifically on the well formed written sentences partition (W-sentence). These are our result, which do not reach the state of the art presented in COTET Et al. [1] of 62.7:

F0.5 10M-base	F0.5 10M-add	F0.5 100k-base	F0.5 100k-add
35.4	36.5	35.1	38.1

5. Conclusions and further work

The task of correcting grammatical errors is not simple, and that problem is compounded by a target language's lack of high quality data. The works pioneered by COTET Et Al. [1] are very useful; however, it serves as a preliminary exploration. Bigger hand corrected datasets and more sophisticated synthetic data generation would greatly improve any type of model. Our model's suffered from a performance standpoint, due to a lack of data, yet much can be done with little.

Even though using token tagging to detect and correct grammatical mistakes seems promising, there is room for improvement. Even with larger and higher quality data, correcting sentences would still be challenging. Part of the difficulty with this current model is that it loses a lot of information when masking a token. There is no mechanism to compare the generated alternative with the initial, erroneous, token. Perhaps a classifier model could be used to pick the most likely token, given the list of probable tokens, their probabilities, the sentence context, and the initial token. Another observed flow is that the underlying models can't communicate properly with each other. Even if one part of the system correctly detects a mistake, it is not guaranteed that the other will act with that same mistake in mind.

There are also a lot of possible improvements that can be gained from more sophisticated prompting of large language models (like GPT-4) to generate more life-like data, besides just generating more data.

The ADD module is also questionable from a quality increase perspective. Even though the dataset scores imply that it has a positive influence on the overall result, it's not that usable for something like an everyday grammar corrector that a text editor user would use. This is due to its tendency to hallucinate. It could theoretically be better used in automatic correction, without the input of any human/system.

REFERENCES

[1] COTET, Teodor-Mihai; RUSETI, Stefan; DASCALU, Mihai. Neural grammatical error correction for romanian. In: *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2020. p. 625-631.

[2] BRYANT, Christopher, et al. Grammatical error correction: A survey of the state of the art. *Computational Linguistics*, 2022, 1-59.

[3] BROWN, Tom, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 2020, 33: 1877-1901.

[4] CONNEAU, Alexis, et al. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv:1911.02116*, 2019.

[5] VASWANI, Ashish, et al. Attention is all you need. *Advances in neural information processing systems*, 2017, 30.

[6] DEVLIN, Jacob, et al. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018

[7] LIU, Yinhan, et al. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[8] RAFFEL, Colin, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 2020, 21.1: 5485-5551.

[9] XUE, Linting, et al. mT5: A massively multilingual pre-trained text-to-text transformer. *arXiv preprint arXiv:2010.11934*, 2020.

[10] LEVENSHTEIN, Vladimir I., et al. Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet physics doklady*. 1966. p. 707-710.

[11] WAGNER, Robert A.; FISCHER, Michael J. The string-to-string correction problem. *Journal of the ACM (JACM)*, 1974, 21.1: 168-173.

[12] NG, Hwee Tou, et al. The CoNLL-2014 shared task on grammatical error correction. In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*. 2014. p. 1-14.

[13] NAPOLES, Courtney; SAKAGUCHI, Keisuke; TETREAULT, Joel. JFLEG: A fluency corpus and benchmark for grammatical error correction. *arXiv preprint arXiv:1702.04066*, 2017.

[14] BRYANT, Christopher, et al. The BEA-2019 shared task on grammatical error correction. In: *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. 2019. p. 52-75.

[15] QORIB, Muhammad; NA, Seung-Hoon; NG, Hwee Tou. Frustratingly easy system combination for grammatical error correction. In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2022. p. 1964-1974.

[16] GRUNDKIEWICZ, Roman; JUNCZYS-DOWMUNT, Marcin; HEAFIELD, Kenneth. Neural grammatical error correction systems with unsupervised pre-training on synthetic data. In: *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*. 2019. p. 252-263.