UKRAINIAN CATHOLIC UNIVERSITY

MASTER THESIS

Improving Sequence Tagging for Grammatical Error Correction

Author:
Maksym TARNAVSKYI

Supervisor:
Kostiantyn OMELIANCHUK

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2021

Declaration of Authorship

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Master of Science

Improving Sequence Tagging for Grammatical Error Correction

by Maksym TARNAVSKYI

Abstract

In this work, we investigated the recent sequence tagging approach for the Grammatical Error Correction task. We compared the impact of different transformer-based encoders of base and large configurations and showed the influence of tags' vocabulary size. Also, we discovered ensembling methods on data and model levels. We proposed two methods for selecting better quality data and filtering noisy data. We generated new training GEC data based on knowledge distillation from an ensemble of models and discovered strategies for its usage. Our best ensemble without pre-training on the synthetic data achieves a new SOTA result of an $F_{0.5}$ 76.05 on BEA-2019 (test), in contrast, when the newest obtained results were achieved with pre-training on synthetic data. Our best single model with pre-training on synthetic data achieves $F_{0.5}$ of 73.21 on BEA-2019 (test). Our investigation improved the previous results by 0.8/2.45 points for the single/ensemble sequence tagging models. The code, generated datasets, and trained models are publicly available.

Acknowledgements

First of all, I would like to thank my supervisor Kostiantyn Omelianchuk, who directed me throughout this research, for his enthusiasm, encouragement, support, patience, valuable ideas, and bits of advice.

Also, I would like to thank the company Grammarly for their financial assistance in obtaining the computing resources, which helped a lot during the investigation.

I am grateful to Ukrainian Catholic University for the incredible master's program and Oleksiy Molchanovsky for coordinating it and providing valuable advice during the whole education.

Finally, I would like to thank my group mates for the memorable moments and time spent together.

Contents

List of Tables

List of Abbreviations

NLP	**N**atural **L**anguage **P**rocessing
GEC	**G**rammatical **E**rror **C**orrection
SOTA	**S**tate-**O**f-**T**he-**A**rt
BERT	**B**idirectional **E**ncoder **R**epresentations from **T**ransformers
NMT	**N**eural **M**achine **T**ranslation
WI	**W**rite **I**mprove dataset
BEA-2019	**B**uilding **E**ducational **A**pplications 2019 shared task

Chapter 1

Introduction

1.1 Importance of Grammatical Error Correction

In today's world, literacy is highly valued. Competent writing requires knowledge of various language rules, exceptions, grammar constructions. Lack of this knowledge or inattention can be the cause of making errors. People might have an unpleasant impression when they notice mistakes in the text. Therefore, it is necessary to check it. However, proofreading is time-consuming, requires attention and in-depth linguistic knowledge. That is why systems that can automatically correct errors in the written text are beneficial and in demand.

With the advent and development of Grammatical Error Correction systems, more and more people have the opportunity to check their texts and write more competently. Such systems can be applied in many scenarios, such as writing essays, papers, reports, emails, messages. They significantly speed up checking, allow to correct mistakes both visually and interactively, and at the same time, teach grammar (Tajiri, Komachi, and Matsumoto, 2012).

GEC system receives a sentence with mistakes and outputs a corrected version. This system should be able to cope with different types of errors. They can be morphological, lexical, punctuation, and others. Both the quality of the corrections and the performance speed of the system are important.

1.2 Motivation

We selected a Grammatical Error Correction topic as it has significant social value and demand. Research on this topic is actively developing, and substantial progress has already been achieved, as described in (Wang et al., 2020). However, there is still a need to improve the accuracy, completeness, and speed of error correction systems. Also, the speed of the model and the amount of memory it takes up are essential, especially in cases when the model needs to be placed on-device (Kairouz et al., 2019).

The Grammatical error correction task is complicated and has many challenges. First of all, each language is rich, and its grammar contains many rules and exceptions. Professional linguists are needed to create high-quality annotated training data (Dahlmeier, Ng, and Wu, 2013, Bryant et al., 2019). The amount of available public GEC data is limited, and creating new data requires much effort. That is why researchers have to develop and improve the architecture of models and create methods for generating and augmenting training data and strategies for their usage (Lichtarge et al., 2019, Stahlberg and Kumar, 2021).

Most grammatical error correction research is based on sequence-to-sequence Neural Machine Translation when a new grammatically correct sentence is generated based on the erroneous sentence(Yuan and Briscoe, 2016, Kaneko et al., 2020).

The sequence tagging approach, which generates a sequence of operations (tags) that must be performed on a sentence to make it grammatically correct, has recently been proposed (Awasthi et al., 2019, Omelianchuk et al., 2020). Models based on this approach also managed to achieve State-of-the-art results.

The sequence tagging approach is poorly studied, and its further research and improvement can have a considerable impact on the development of the GEC industry.

In our study, we considered using English as the most significant progress has been made for it. English has the most considerable amount of annotated training data, and most existing GEC approaches report their results on English common benchmarks (Ng et al., 2014,Bryant et al., 2019). However, it is essential to remember that new things developed for English can eventually be applied to other languages. In our work, we wanted to focus on improving the recent sequence tagging approach and leaving its application to other languages for future work.

1.3 Goals of the master thesis

The main goal of our Master's thesis is to explore and improve the sequence tagging approach for Grammatical Error Correction and strengthen result on the newest closed BEA-2019 shared task (Bryant et al., 2019).

We will first analyze existing GEC approaches, identify their advantages and disadvantages, suggest methods for improving correction quality, and verify them. The tasks we set before ourselves are the following:

1. We want to compare the impact of transformer-based encoders such as BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019), DeBERTa (He et al., 2020), XLNet (Yang et al., 2019) of the base, and large configurations on grammatical error correction.

2. We want to check if models with large encoder configurations perform better than models with base configurations. Also, we plan to compare their inference time.

3. We want to discover if increasing/reducing the tags vocabulary size for models improves the quality of corrections.

4. We plan to compare ensembling methods such as stacking on model and data levels. We want to check whether both methods have the same results when combining models with the same vocabulary. Furthermore, investigate whether combining models with different vocabulary sizes will improve the quality of correction.

5. We want to analyze the quality of existing training GEC datasets and propose methods for filtering low-quality data samples. We plan to check how training on the filtered dataset will influence the result. Furthermore, we plan to investigate the impact of the training model on the dataset extended with additional potential good quality samples.

6. We want to investigate generating training data based on the distillation of knowledge by an ensemble of models. We plan to compare pre-training on

generated data by an ensemble of models, pre-training on publicly available synthetic data, and pre-training on their combination. Also, we want to train the models in one stage on joined corpora of generated data and the high-quality Write Improve dataset (Bryant et al., 2019).

1.4 Structure of the thesis

In chapter 1, we make a brief overview of the GEC topic and define research goals.

Chapter 2 reviews the main existing GEC models, data augmentation techniques, and strategies for their usage. This review allows us to substantiate our proposed ideas for improvement.

In chapter 3, we discuss the main public English datasets that are available for use. Also, we will talk about the main shared task benchmarks that are used to validate models. We will also talk about the datasets we have chosen to generate new distilled training data.

In chapter 4, we start with a more detailed overview of the chosen sequence tagging approach. Next, we present a comparison of the usage of different transformer-based encoders and the impact of different tag dictionary sizes. Also, in this section, we compare methods of models ensembling.

In chapter 5, we present data filtering techniques. We will also discuss our proposed approach for data selection based on data clustering and the approach based on the similarity between text embeddings generated by model checkpoints before and after fine-tuning.

Chapter 6 discusses the method of generating distilled training data and the effect of different datasets on training.

In section 7, we summarize the results and indicate the direction for further research.

Chapter 2

Literature Review

A comprehensive overview of the GEC area was done in (Wang et al., 2020). It contains an overview of the main public datasets, evaluation benchmarks, existing approaches and their development.

2.1 Existing GEC approaches

2.1.1 The first GEC models

The initial GEC approaches were hand-crafted rule-based models developed by professional linguists (similar to Naber, 2003). However, it was complicated to describe so many rules and exceptions used in the language which would not contradict each other.

The first data-driven approaches were based on Statistical Machine Translation (SMT). An example of an application is described in (Yuan and Felice, 2013). This work describes phrase-based statistical machine translation (PBSMT) for the automatic correction of errors. This model produces conditional Bayesian probabilities between transitions from erroneous phrases to grammatically correct phrases. It doesn't yield high performance on CoNLL-2013 shared task (Kao et al., 2013), which is a common benchmark for English GEC, but reveals many problems that require careful attention when building SMT systems for error correction.

The approach based on Neural Machine Translation (NMT) is first described in the article (Yuan and Briscoe, 2016). It used the Encoder-Decoder method. The encoder encodes an erroneous sentence into a vector, and the decoder generates an already corrected sentence based on this vector. Both the encoder and the decoder were RNNs(Cho et al., 2014) composing of GRU or LSTM (Hochreiter and Schmidhuber, 1997) units. The model achieved the new SOTA $F_{0.5}$ 39 on CoNLL-2014(Ng et al., 2014) at that time.

The advent of the Transformer architecture (Vaswani et al., 2017) has made a significant contribution to NLP development, and most modern GEC approaches use BERT-based (Devlin et al., 2018) encoders and decoders.

2.1.2 Recent sequence-to-sequence models

The application of BERT as an encoder and decoder for the GEC task is investigated in (Kaneko et al., 2020). Authors evaluated three methods:

(a) initializing an Encoder-Decoder GEC model using pre-trained BERT as BERT-init (Lample and Conneau, 2019);

(b) passing the output of pre-trained BERT into the Encoder-Decoder GEC model as additional features (BERTfuse) (Zhu et al., 2020);

(c) combining the best parts of (a) and (b) in their new method (c), when first they fine-tune BERT with the GEC corpus and then use the output of the fine-tuned BERT model as additional features in the GEC model. Their approach (c) had better performance with results of $F_{0.5} = 65.2$ at CoNLL-2014 (Ng et al., 2014) and $F_{0.5} = 69.8$ at BEA-2019 (Bryant et al., 2019), which is a newer benchmark for English GEC.

Sequence-to-sequence NMT-based approaches give quite good results but also have limitations. Their learning requires a lot of training data, and they are relatively slow. Therefore, researchers are trying to find methods that would speed up the performance of the models. One of them is the approach which improves GEC's efficiency by dividing the task into two subtasks: Erroneous Span Detection (ESD) and Erroneous Span Correction (ESC)(Chen et al., 2020). ESD identifies grammatically incorrect text spans with an efficient sequence tagging model. Then, ESC leverages a sequence-to-sequence model to take the sentence with annotated erroneous spans as input and only outputs the corrected text for these spans. Experiments show that their approach performs comparably to conventional sequence-to-sequence models, having F0.5 = 61.0 on CoNLL-14 (Ng et al., 2014) benchmark with less than 50% time cost for inference.

2.1.3 Recent sequence tagging models

Most of what the sequence-to-sequence Encoder-Decoder models output is almost the same sentence fed to the input. The Encoder-Decoder models autoregressively capture full dependency among output tokens but are slow due to sequential decoding. A much more straightforward task is to predict the edit tags that need to be applied to turn erroneous sentences into correct ones.

One of the recent attempts to apply a sequence tagging approach to GEC was done in LaserTagger paper (Malmi et al., 2019). LaserTagger is a sequence tagging model that casts text generation as a text editing task. Corrected texts are reconstructed from the inputs using three main edit operations: keeping a token, deleting it, and adding a phrase before the token. The model combines a BERT encoder with an autoregressive Transformer decoder, which predicts edit operations. LaserTagger had $F_{0.5} = 40.5$ on the BEA-19 (Bryant et al., 2019) test. However, it gave great impetus to the development of sequence tagging models.

Another approach was proposed in the article (Awasthi et al., 2019). Their Parallel Iterative Edit (PIE) model does parallel decoding, giving competitive accuracy with the Encoder-Decoder models. This is possible because model:

- Predicts edits instead of tokens.
- Labels sequences instead of generating sequences.
- Iteratively refines predictions to capture dependencies.
- Factorizes logits over edits and their token argument to harness pre-trained language models like BERT.

This method achieves $F_{0.5} = 59.7$ on the CoNLL-14Ng et al., 2014 task and is a significantly faster alternative for local sequence transduction.

A similar approach, which currently has state-of-the-art results, is proposed in the article (Omelianchuk et al., 2020). The authors present a simple and efficient GEC sequence tagger using a Transformer as an encoder and linear layers with softmax for tag prediction and error detection instead of a decoder. In their experiments, encoders from XLNet(Yang et al., 2019) and RoBERTa(Liu et al., 2019) outperform three other cutting-edge Transformer encoders (ALBERT(Lan et al., 2020), BERT(Devlin et al., 2018), and GPT-2(Radford et al., 2018)).

The authors consistently used pre-trained transformers in their Base configurations and the size of tag vocabulary was 5000. Their GEC system is pre-trained on synthetic data and then fine-tuned in two stages: first on errorful synthetic corpora and second on a combination of errorful and error-free parallel corpora. Also, they designed custom token-level transformations to map input tokens to target corrections. These transformations increase grammatical error correction coverage for limited output vocabulary size for the most common grammatical errors, such as *Spelling, Noun Number, Subject-Verb Agreement,* and *Verb Form* (Yuan and Briscoe, 2016).

Their best single-model, GECToR (XLNet) achieves $F_{0.5} = 65.3$ on CoNLL-2014(Ng et al., 2014) (test) and $F_{0.5} = 72.4$ on BEA-2019(Bryant et al., 2019) (test). Best ensemble model, GECToR (BERT + RoBERTa + XLNet) where they simply average output probabilities from 3 single models achieves $F_{0.5} = 66.5$ on CoNLL-2014 (Ng et al., 2014) and $F_{0.5} = 73.6$ on BEA-2019 (Bryant et al., 2019), correspondingly. Their inference speed is up to 10 times as fast as a Transformer-based sequence-to-sequence GEC system.

2.1.4 Data augmentation techniques and pre-training strategies

Encoder training requires as much quality data as possible. However, the most extensive set of publicly available parallel data (Lang-8, Tajiri, Komachi, and Matsumoto, 2012) in GEC has only one million sentence pairs. Therefore, many researchers are now actively exploring data augmentation methods and strategies for their use.

For instance, in the investigation (Kiyono et al., 2019), the authors proposed two approaches to error generation. In the first approach (a1), they set the probability distributions of different types of errors, such as deleting, repeating, substituting words in a sentence, and then directly added these errors to the correct sentences. In the second approach (a2), they trained the Encoder-Decoder Transformer model (Vaswani et al., 2017) in a reversed way, which received correct sentences as input and returned sentences with errors at the output. As a result, the GEC model, which trained on the data generated by approach (a1), had better results on the BEA-19 test (Bryant et al., 2019).

Furthermore, they compared two strategies for model training on synthetic data. In the first experiment, they trained the model on the joined synthetic and real data. As a result, they got a slightly worse score than when they trained the model only on real data. In the second experiment, they first pre-trained the model only on synthetic data and then trained the model only on real data. The second approach yielded significant improvements. This investigation demonstrated that training strategy is essential for outcome. Their final best pre-trained models achieved $F_{0.5} = 65.0$ on CoNLL-14 (Ng et al., 2014) and $F_{0.5} = 70.2$ on BEA-19 (Bryant et al., 2019).

Another two approaches for generating large parallel corpora for Grammatical Error Correction using publicly available Wikipedia data are described in (Lichtarge et al., 2019). The first method extracts source-target pairs from Wikipedia edit histories with minimal filtration heuristics. In contrast, the second method introduces noise into Wikipedia sentences via round-trip translation through bridge languages. The authors demonstrated that neural GEC models trained using either type of corpora gave a similar performance. Fine-tuning models on the Lang-8 corpus and ensembling allowed them to achieve $F_{0.5} = 60.4$ on CoNLL-14 (Ng et al., 2014) task.

Further investigation about data-weighting strategies of using training data described in (Lichtarge, Alberti, and Kumar, 2020). The authors proposed the metric delta-log-perplexity (Δppl), defined as the difference in negative log-probability (log-perplexity) of an individual training example between two checkpoints in model training. The first checkpoint corresponds to model (θ^-), which is trained on a base dataset D^-, while the second checkpoint corresponds to the model (θ^+) after further fine-tuning on a second target dataset D^+ (with trusted quality). Δppl between those models for a given example (composed of input, output pair (i, o)) should suggest which of the datasets the example is more similar to, from the successive models θ^- and θ^+.

$$\Delta ppl(i, o; \theta^-, \theta^+) = log\ p(o|i; \theta^-) - log\ p(o|i; \theta^+)$$

When D^+ is selected to be "higher quality" than D^-, then the Δppl scores of examples drawn from D^- provide a heuristic for assessing their quality. Having these scores, the authors proposed a strategy for their use. First, they sorted the training examples according to scores. Since the model can forget more what it learned before and remember what it learned recently, the authors gave fewer quality examples at the beginning of training and high-quality - at the end. Thus, the model was able to study these more meaningful examples better and, as a result, achieved higher results on benchmarks. Also, they down-weighted the loss of low-scoring examples during training, which further improved the final result.

Using the weighting strategy, they increased $F_{0.5}$ for their single / ensemble models from 61.1 to 62.1 / from 65.3 to 66.8 on CoNLL-14(Ng et al., 2014) and from 66.1 to 66.5 / from 71.9 to 73.0 on BEA-19 (Bryant et al., 2019) tests.

The recent work about synthetic data generation presented in (Stahlberg and Kumar, 2021). In this work, the authors used error type tags from automatic annotation

Model	CoNLL-2014. $F_{0.5}$	BEA-2019 $F_{0.5}$
Single systems		
(Yuan and Briscoe, 2016)	39	-
(Kaneko et al., 2020)	62.6	65.6
(Chen et al., 2020)	61.0	-
(Malmi et al., 2019)	-	40.5
(Awasthi et al., 2019)	59.7	-
(Omelianchuk et al., 2020)	65.3	**72.4**
(Kiyono et al., 2019)	65.0	72.0
(Lichtarge et al., 2019)	56.8	-
(Lichtarge, Alberti, and Kumar, 2020)	62.1	66.5
(Stahlberg and Kumar, 2021)	**66.6**	70.4
Ensembles		
(Kaneko et al., 2020)	65.2	69.8
(Awasthi et al., 2019)	61.2	-
(Omelianchuk et al., 2020)	66.5	73.6
(Lichtarge et al., 2019)	60.4	-
(Lichtarge, Alberti, and Kumar, 2020)	66.8	73.0
(Stahlberg and Kumar, 2021)	**68.3**	**74.9**

TABLE 2.1: The evaluation of models on GEC benchmarks

tools such as ERRANT to guide synthetic data generation. They used sequence-to-edits tagging models (Stahlberg and Kumar, 2020) to build a new, large synthetic pre-training data set with error tag frequency distributions matching a given development set. Their synthetic data set yields large and consistent gains, surpass the SOTA on the BEA-19 test with $F_{0.5} = 74.9$ and on CoNLL-14 test with $F_{0.5} = 68.3$.

To sum up, the articles in this section have shown that it is essential to choose not only the model's architecture but also its training strategy.

2.2 Analysis

The comparison of model's performance is presented in Table 2.1.

Based on the newer closed benchmark BEA-19, the best result among single models has sequence tagging GECToR (XLNet) model (Omelianchuk et al., 2020), which has slightly better results than (Kiyono et al., 2019) and (Stahlberg and Kumar, 2021). Among the ensembles, the sequence-to-sequence models with pre-training on recent generated data from (Stahlberg and Kumar, 2021) have significantly surpassed previous SOTA result of GECToR (BERT + RoBERTa + XLNet) models from (Omelianchuk et al., 2020).

Most approaches use sequence-to-sequence models, while the use of sequence tagging models is much less studied. In our research, we decided to focus on the latest sequence tagging approach based on work (Omelianchuk et al., 2020) and explore its model architecture configurations and pre-training strategies.

Chapter 3

Datasets and evaluation

3.1 Public GEC datasets observation

The main public datasets used for supervised learning of GEC models are NUCLE (Dahlmeier, Ng, and Wu, 2013), Lang-8 (Tajiri, Komachi, and Matsumoto, 2012), FCE (Yannakoudakis, Briscoe, and Medlock, 2011), JFLEG (Napoles, Sakaguchi, and Tetreault, 2017), WriteImprove+LOCNESS (Bryant et al., 2019). They consist of parallel pairs of erroneous and grammatically correct sentences. Statistical information about them are given in Table 3.1 based on information from (Wang et al., 2020).

3.1.1 NUCLE

The NUS Corpus of Learner English (NUCLE) is the first GEC dataset freely available for research purposes. NUCLE consists of essays written by undergraduate students at the National University of Singapore (NUS). The essays were written as course assignments on different topics, like technology innovation or health care. Dataset was annotated by NUS teachers.

3.1.2 CoNLL-2014

The CoNLL-2014 shared task is a grammatical error correction benchmark, which tests models to cope with all types of errors. Its data is a deferred part of the NUCLE corpus. Two professional annotators independently annotated test essays.

3.1.3 FCE

The First Certificate in English Corpus (FCE) is a public subset of the private Cambridge Learner Corpus (CLC), and it is a collection of sentences written by English language learners in response to FCE exam questions for upper-intermediate level. FCE contains diverse sentence proficiency and wide topics.

3.1.4 Lang-8

The Lang-8 Corpus of Learner English is a dataset based on data from the Lang-8 online language learning website where native speakers correct grammar in essays posted by language learners. This dataset has an immense amount of GEC training samples. However, sentences in this dataset were corrected by annotators with different proficiency levels, which influenced data quality.

Corpus	Component	Sentences	Tokens	Characters	Rate of changed sentences
NUCLE	Train	57k	1.16M	115	38%
CoNLL-2014	Test	1.3k	30.1 k	23	89%
FCE	Train Dev Test	28k 2.1k 2.7k	455k 35k 42k	74	62%
Lang-8	Train	1.04 M	11.86 M	56	42%
JFLEG	Dev Test	754 747	14k 13k	94	86%
W&I	Train Dev Test	34.3k 3.4k 3.5k	628.7k 63.9k 62.5k	60 94 -	67% 69%
LOCNESS	Dev Test	1k 1k	23.1k 23.1k	123 -	52%

TABLE 3.1: Statistics and properties of public GEC datasets.

3.1.5 JFLEG

JHU FLuency-Extended GUG corpus (JFLEG) is a parallel corpus for developing and evaluating grammatical error correction. It represents a wide range of language proficiency levels and uses edits to correct grammatical errors and make the original text more native.

3.1.6 Write and Improve + LOCNESS

Write Improve Corpus (WI), and LOCNESS Corpus are newer datasets introduced by The BEA-2019 Shared Task on Grammatical Error Correction.

WI consists of 3600 annotated essay submissions from Write Improve, an online web platform that assists non-native English students with their writing. The essays are split into train, development, and test sets with 3000, 300, and 300 samples.

LOCNESS Corpus is a collection of about 400 essays written by British and American undergraduates. It contains only the development and test sets.

3.2 Monolingual datasets

This section presents the datasets which we have chosen to generate new training GEC data based on them. These datasets are One Billion Word Benchmark (Chelba et al., 2013), The Blog Authorship Corpus (Schler et al., 2005), and Amazon reviews dataset (Ni, Li, and McAuley, 2019). Our best ensemble of models will detect sentences that have potential errors and try to correct them. The more data and the more errors they have, the more we can generate training data.

3.2.1 One Billion Word Benchmark

The One Billion Word Benchmark is a corpus with more than one billion words of training data. We chose this corpus as it is one of the well-known popular datasets that are publicly available. It contains many texts so that it can generate a lot of training data. However, it mainly consists of news data, so we did not expect a very high rate of errors.

3.2.2 The Blog Authorship Corpus

The Blog Authorship Corpus consists of the collected posts of 19 320 bloggers gathered from blogger.com in August 2004. The corpus incorporates a total of 681 288 posts and over 140 million words. Bloggers included in the corpus are of all ages - from schoolchildren, students and more mature. Texts in this dataset are on various topics - traveling, cooking, movies, lifestyle. The amount of data is limited. We selected this dataset because its texts were written by ordinary people without correction by editors.

3.2.3 Amazon reviews dataset

The Amazon reviews dataset contains 82.83 million unique product reviews from around 20 million users. There are about 30 different categories, such as (Books, Clothing, Electronics, Sports, and others). In our investigation, we only used data from the "Clothing, Shoes and Jewelry" category for annotation, but we want to try to combine comments from different categories in the future. Ordinary users wrote these reviews, and there are many possibilities for grammatical error correction.

3.3 Evaluation

The commonly accepted shared tasks CoNLL-2014(Ng et al., 2014) and BEA-2019(Bryant et al., 2019) are used to check and compare existing GEC models' quality.

These benchmarks test the ability of models to cope with all types of errors (punctuation, spelling, and others), and the sentences for them were annotated by professional linguists.

The main metrics for evaluating GEC models are Precision, Recall, and $F_{0.5}$, which gives more weight to the Precision of the corrections than to Recall. Any edits with the same span and correction in target and corrected sentences are true positive (TP). In contrast, unmatched edits in the corrected and target sentences are false positives (FP) and false negatives (FN), respectively.

$$F_{0.5} = \frac{(1+0.5)^2 * Precision * Recall}{(0.5)^2 * Precision + Recall}$$

Where

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Tools M2Scorer (Dahlmeier and Ng, 2012) and ERRANT (Bryant, Felice, and Briscoe, 2017) are most often used to assess text correction quality. Their main difference is how they extract edits from parallel original and corrected sentences and compare their overlap with ground truth edits. M2Scorer is the official scorer for the CoNLL-2014 task, and ERRANT is for the BEA-2019 shard task.

3.4 Conclusion

In this section, we discussed the main public available GEC datasets for the English language. We used sequentially joined datasets Lang-8 + NUCLE + FCE + WI for training models in our investigation. For final fine-tuning, we used the WI dataset. The metric that we tried to optimize was $F_{0.5}$ score. All the experiments we evaluated on the BEA-2019 dev part. We chose this benchmark as it is newer than CoNLL-2014, and its target sentences are publicly unavailable, making the comparison of the models fairer.

We decided to use One Billion Word Benchmark, The Blog Authorship Corpus, and Amazon reviews dataset for generation new training data. We chose datasets of different origins to discover the impact of initial data on the quality of generated GEC training samples.

Chapter 4

Model observation

4.1 Baseline approach

The sequence tagging approach implemented in (Omelianchuk et al., 2020) was taken as a basis. The architecture of their model consists of two main components. They use a transformer-based encoder and two linear layers instead of a decoder. The first linear layer detects whether it is necessary to make changes for a particular word. The second linear layer is responsible for selecting the tag, which will be executed on the word. Then there is post-processing, during which operations are performed depending on the predicted tags.

The primary tag operations are "KEEP" - leave the word unchanged, "DELETE" - delete the word, "APPEND word" - add the new word after the current word, "REPLACE word" - replace the current word with a new word. In addition to these tags, their implementation included custom transformation tags, such as lead to uppercase or lowercase, lead to 3rd forms of irregular verbs, add "s" endings, etc.

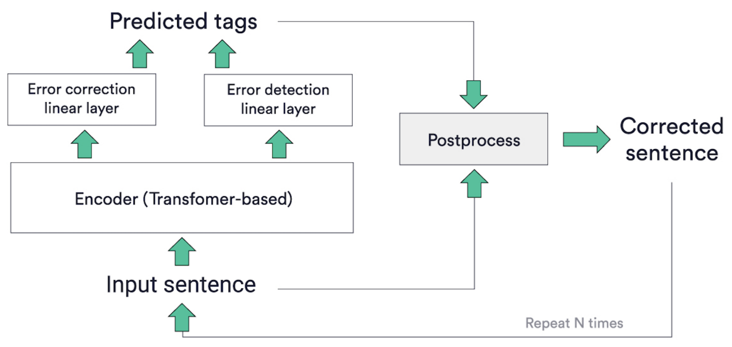
Predicted tags
Error correction linear layer
Error detection linear layer
Postprocess
Corrected sentence
Encoder (Transfomer-based)
Input sentence
Repeat N times

FIGURE 4.1: GECToR model: iterative pipline.
Source: https://www.grammarly.com/blog/engineering/gec-tag-not-rewrite

The task we optimize is a multi-class classification of tags, which we assign sequentially for each word. Loss function is a multi-class categorical entropy. The criterion for stopping learning was early stopping if the model did not improve the loss function value for the dev train part during the following three epochs. This way, we chose the best models and compared them on the deferred BEA-2019 dev part. And only the models that had performed best on the dev part, we later tested on the BEA-2019 test part.

Training a language model requires a lot of data. When the amount of data is limited, it is often used to initialize the model with pre-trained weights and then train the model on task-specific data, as investigated in (Kaneko et al., 2020). Therefore, we initialized the encoder with pre-trained weights, and the linear layers were initialized with random weights. As long as the linear layers are not trained not to spoil the good initialized weights of the encoder, training with several cold steps is used. For example, in the first two epochs, we train only linear layers, and then the whole model and the encoder and linear layers are trained. Comparison of training with two cold steps and without them can be seen in Table A.1

Model training occurs in several stages. The initial training takes place on the following joined datasets: Lang8, FCE, NUCLE, WL. After that, we continue training the model only on the WI dataset. It allows us to train a model on a large amount of data of different quality and then fine-tune it on the highest quality data. The ablation study, when we trained model in two stages and when we only trained the model on the WI dataset, can be seen in Table A.2.

Usually, a small number of corrections need to be applied in a sentence. The most common operation is not to change anything (use the KEEP tag). In section 3, we showed that the Lang8, FCE, NUCLE, and WI datasets contain a high percent of sentences without changes, increasing the share of KEEP tags. Training on many unchanged sentences makes the model much harder to learn, as it is easier for a model to recommend the KEEP tag than any other. Therefore, when the model is trained on joined data, we filter out those sentences in which all tags are only KEEP. A comparison of the training model with and without filtering can be seen in Table A.3.

However, it is necessary to cover all possible sentences on the latest training stage so that the model knows that there are sentences for which no changes need to be made. Not filtering data during fine-tuning on the WI dataset gives much better results than fine-tuning on filtered data.

Since Precision is more critical than Recall in the GEC task, we can find optimal threshold confidence for correcting using hyperparameters. For example, it is possible to artificially increase the probability for a KEEP tag or set minimum confidence of a correction. We selected these hyperparameters on the dev part. The results after hyperparameter selection are shown in the following sections.

Sequence tagging GEC framework, developed in (Omelianchuk et al., 2020), mainly based on libraries AllenNLP and transformers.

In the original GECToR framework, the authors used their implementation for the BPE tokenizer. They used it to speed up tokenization. In our work, we moved on to the recently implemented fast tokenizers from transformers. They are written on Rust and are a quicker version of previous tokenizers. Such code migration allowed encoders in the models to have the same tokenizer, which they used during their initial pre-training. Usage of original tokenizers allowed the models to achieve slightly better results after training.

4.2 Comparison of transformer-based encoders

4.2.1 Transformer-based encoders

The basis of sequence tagging models is the transformer-based tokenizers. In (Omelianchuk et al., 2020), they tried to use the LSTM(Hochreiter and Schmidhuber, 1997) network as an encoder, but such a model had a much worse result than transformer-based encoders. They also found that the RoBERTa (Liu et al., 2019) and XLNet(Yang et al., 2019) encoders performed better than BERT(Devlin et al., 2018), ALBERT(Lan et al., 2020), and GPT-2(Radford et al., 2018) encoders. The authors studied the model only with base configurations. In our study, we would like to compare the effect of base and large configurations for the BERT, RoBERTa, XLNet models, and the recent DeBERTa(He et al., 2020) encoder.

Information on the main configuration parameters of these encoders we provided in Table 4.1.

Encoder name	**Layers**	**Hidden**	**Heads**	**Parameters**
BERT base	12	768	12	109 M
RoBERTa base	12	768	12	125 M
DeBERTa base	12	768	12	140 M
XLNet base	12	768	12	110 M
BERT large	24	1024	16	335 M
RoBERTa large	24	1024	16	335 M
DeBERTa large	24	1024	16	400 M
XLNet large	24	1024	16	340 M

TABLE 4.1: Parameters of transformer-based models

Most encoders have a BERT-like architecture. However, each of the encoders has its improvements and differences. Among the base models, BERT base and XLNet base have the fewest parameters, RoBERTa base has slightly more parameters and DeBERTa base has the broadest range of parameters. There are almost the same number of parameters in the encoders BERT large, RoBERTa large, XLNet large, and many more parameters in the model DeBERTa large. The number of parameters affects the number of calculations, so models with more parameters will train longer and take more time for predictions.

4.2.2 Comparison of the transformer-based encoders on GEC

In this experiment, we trained models that we will consider as a baseline for comparison with the following experiments. As justified in Section 4.1, we used two cold steps, early stopping three, and a tag dictionary size of 5000. Usage of the same dictionary with 5000 tags made it possible to compare our baseline with the results obtained in (Omelianchuk et al., 2020). Initially, we trained base and large models only on errorful sentences of the joined dataset (Lang-8, Nucle, FCE, WI connected sequentially, without shuffling). The training results are presented in Table 4.2.

Encoder	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa base	**50.12**	34.04	**45.79**
DeBERTa base	49.27	**35.5**	45.73
XLNet base	47.78	33.39	43.98
BERT base	48.6	28.95	42.79
RoBERTa large	**52.11**	37.34	**48.29**
DeBERTa large	48.62	**41.16**	46.92
XLNet large	48.46	39.57	46.37
BERT large	49.39	29.96	43.72

TABLE 4.2: Comparing transformer-based encoders trained on joined dataset (Lang-8 + NUCLE + FCE + WI)

As we can see, the best results were achieved by the models with RoBERTa and DeBERTa encoders for both configurations.

The dynamics for the models' results remained the same, which means that those models that performed better with base configurations performed better also with large configurations and vice versa. All models with large encoders had a higher Recall value than models with base encoders. For most larger models, the Precision value became slightly better, and in general, according to the $F_{0.5}$ metric, such models had better results.

Encoder	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
ALBERT	43.8	22.3	36.7
BERT base	48.3	29.0	42.6
GPT-2 base	44.5	5.0	17.2
RoBERTa base	**50.3**	30.05	**44.5**
XLNet base	47.1	**34.2**	43.8

TABLE 4.3: Results for models trained on joined dataset (Omelianchuk et al., 2020)

If we compare with the results given in (Omelianchuk et al., 2020), shown in Table 4.3, our trained model RoBERTa base had a somewhat better Recall value and almost the same Precision value. Such improvement in the result can be explained by moving from a custom implementation of the BPE tokenizer to the original, which we described in previous paragraphs. The results for the XLNet base models are pretty similar, which can be explained by the fact that those models used the same tokenizer.

After that, we fine-tuned our models only on the WI dataset, using errorful and error-free sentences. A comparison of the results can be seen in Table 4.4.

Encoder	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa base	**53.77**	39.23	**50.06**
DeBERTa base	53.02	**39.54**	49.63
XLNet base	50.87	39.43	48.08
BERT base	49.88	33.95	45.6
RoBERTa large	**54.85**	42.54	**51.85**
DeBERTa large	53.97	42.45	51.19
XLNet large	53.06	**42.65**	50.59
BERT large	51.96	37.93	48.38

TABLE 4.4: Comparing transformer-based encoders fine-tuned on WI

After fine-tuning, the RoBERTa base, DeBERTa base, and XLNet base models had almost the same recall value. In contrast, the precision value for RoBERTa base was the highest, DeBERTa base it was slightly lower, and for XLNet, it was much lower. The model with the BERT base encoder had significantly worse results compared to other models.

A similar case was for models with large encoder configurations. They had better and almost identical Recall values for RoBERTa, DeBERTa, and XLNet encoders, and their Precision was better than results from the previous stage. The model with the BERT-large encoder also had the worst results among large models, although its results were better than for BERT large without fine-tuning.

Finally, we made a selection of hyperparameters. Based on the grid search on the BEA-2019 dev part, a minimum error probability (mep) and additional confidence (ac) for the KEEP token were selected to improve Precision reducing Recall. The results can be seen in Table 4.5.

Encoder	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa base	62.49	**32.26**	52.63
DeBERTa base	**64.22**	31.87	**53.38**
XLNet base	63.16	30.59	52.07
BERT base	57.21	29.93	48.39
RoBERTa large	65.76	33.86	**55.33**
DeBERTa large	**66.35**	32.77	55.07
XLNet large	64.27	**35.17**	55.14
BERT large	61.18	31.26	51.35

TABLE 4.5: Comparing transformer-based encoders after the selection of hyperparameters

As a result, the model with the RoBERTa large encoder had the highest value of 55.33 $F_{0.5}$. For base configurations, the best value of 53.38 belonged to the DeBERTa base.

In general, the RoBERTa and DeBERTa encoders have very similar results. For a further, more detailed study of the models, we chose models with RoBERTa base and RoBERTa large encoders. We selected them to compare our results for models with synthetic data pre-training with the presented results in (Omelianchuk et al., 2020). Also, the training time of RoBERTa is shorter than for DeBERTa (this is due to the different number of parameters), which allowed us to do more experiments faster.

4.3 Tags vocabulary

An essential component of the sequence tagging approach is the tags dictionary. It determines which corrections the model will learn to make. We formed the dictionary based on the most common correction operations encountered in the Lang-8 + NUCLE + FCE + WI corpuses joined GEC dataset.

4.3.1 Vocabulary size

Since the tag space of operations is limited, the increase of tags vocabulary size should increase Recall (more corrections can be covered) and potentially decrease Precision (it is more challenging to choose the correct tag from available tags). And vise versa, reducing the dictionary size should increase Precision and reduce Recall.

It's also worth noting that expanding or cutting tags vocabulary size can affect the amount of training data we use for basic training on a joined dataset. Since we filter out sentences that contain only the KEEP tokens, increasing the number of tags reduces the number of sentences we filter out. On the one hand, this can help to use more training data, show more corrections during model training. On the other hand, there may be a small number of training cases for rare tags, and the training data may include more incorrect corrections.

To test our hypothesis, we chose vocabulary sizes of 1k, 5k, and 10k. Such values allowed us to investigate cases that differ significantly from each other.

The training results on the Joined dataset can be seen in Table 4.6.

Model	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa base + voc 5k	**50.12**	**34.04**	**45.79**
RoBERTa base + voc 10k	49.4	33.96	45.28
RoBERTa large + voc 1k	**53.57**	34.97	48.42
RoBERTa large + voc 5k	52.11	37.34	48.29
RoBERTa large + voc 10k	52.22	**37.89**	**48.55**

TABLE 4.6: Comparing the impact of vocabulary size on models trained on the joined dataset (Lang-8 + NUCLE + FCE + WI)

As expected, using a 1k dictionary for RoBERTa large increased Precision and reduced Recall, compared to the RoBERTa large model with a 5k vocabulary size. However, the use of a more extensive 10k dictionary for RoBERTa large did not change the learning outcome. Most likely, the model did not use new additional operations from the dictionary. For RoBERTa base, usage of the 10k dictionary, on the contrary, slightly worsened the result. Recall remained almost the same, but Precision decreased.

Next, we fine-tuned models on the WI dataset. The 10k model managed to get a bit more Recall and more Precision values than the model with the 5k dictionary.

The model with the 1k dictionary, on the other hand, had a smaller Recall and the highest Precision among all models. However, a smaller Recall value did not allow for it to obtain a high metric $F_{0.5}$, and it has almost the same result as the model with a 5k vocabulary size.

The results after fine-tuning can be seen in Table 4.7.

Model	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa base + voc 5k	53.77	39.23	50.06
RoBERTa base + voc 10k	–	–	–
RoBERTa large + voc 1k	**56.12**	39.87	51.89
RoBERTa large + voc 5k	54.85	42.54	51.85
RoBERTa large + voc 10k	55.34	**43.05**	**52.35**

TABLE 4.7: Comparing the impact of vocabulary size on models fine-tuned on the WI dataset

Next, we selected the hyperparameters for the model RoBERTa large with a vocabulary size of 10k, which allowed us to obtain the highest $F_{0.5}$ score among all models trained with such a training strategy. We did not select hyperparameters for the model with the RoBERTa large encoder and the 1k dictionary size as it had a much smaller Recall than the model with the 10k dictionary. We expected a further reduction in Recall would not allow achieving a much better $F_{0.5}$ metric than for the model with size 10k.

The results after the selection of hyperparameters can be seen in Table 4.8.

Model	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa large + voc 1k	–	–	–
RoBERTa large + voc 5k	**65.76**	33.86	55.33
RoBERTa large + voc 10k	64.72	**36.04**	**55.83**

TABLE 4.8: Comparing the impact of vocabulary size on models after the selection of hyperparameters

Since the 10k dictionary for RoBERTa large made it possible to improve the result, we decided to train the models XLNet large and DeBERTa large also with this dictionary.

Comparisons of large model training at different stages of training can be seen in Tables 4.9, 4.10, and 4.11.

Model	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa large + voc 5k	**52.11**	37.34	**48.29**
DeBERTa large + voc 5k	48.62	**41.16**	46.92
XLNet large + voc 5k	48.46	39.57	46.37
RoBERTa large + voc 10k	**52.22**	37.89	**48.55**
DeBERTa large + voc 10k	51.27	**39.32**	48.33
XLNet large + voc 10k	50.03	38.16	47.1

TABLE 4.9: Comparing the impact of 10k vocabulary size on models trained on the joined dataset (Lang-8 + NUCLE + FCE + WI)

In contrast to RoBERTa large, for which the increase in vocabulary in the first stage had almost no effect, for both DeBERTa and XLNet the Recall values decreased slightly, and the Precision increased.

Model	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa large + voc 5k	**54.85**	42.54	**51.85**
DeBERTa large + voc 5k	53.97	42.45	51.19
XLNet large + voc 5k	53.06	**42.65**	50.59
RoBERTa large + voc 10k	**55.34**	43.05	**52.35**
DeBERTa large + voc 10k	54.36	**43.31**	51.72
XLNet large + voc 10k	53.11	42.76	50.66

TABLE 4.10: Comparing the impact of 10k vocabulary size on models fine-tuned on the WI dataset

After fine-tuning on the WI dataset, the DeBERTa model with a 10k dictionary had better Precision and Recall values than the DeBERTa model with a 5k dictionary. Models with an XLNet encoder had almost exactly the same metrics.

After selecting the hyperparameters, DeBERTa large with voc of 10k was also able to improve the results compared to the DeBERTa model with the 5k dictionary, and the XLNet model was not. For the model with an XLNet encoder from 10k dictionaries, all metrics deteriorated after selecting hyperparameters.

Model	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa large + voc 5k	65.76	33.86	**55.33**
DeBERTa large + voc 5k	**66.35**	32.77	55.07
XLNet large + voc 5k	64.27	**35.17**	55.14
RoBERTa large + voc 10k	64.72	**36.04**	**55.83**
DeBERTa large + voc 10k	**65.46**	34.59	55.55
XLNet large + voc 10k	64.12	34.02	54.48

TABLE 4.11: Comparing the impact of 10k vocabulary size on models after the selection of hyperparameters

Therefore, increasing the dictionary size to 10k for large encoders can help improve the result, as it happened for models with RoBERTa large and DeBERTa large, but there are cases when this leads to deterioration, as for the model with XLNet large.

4.4 Compare inference time

In addition to the quality of grammatical error corrections, we also decided to compare the running time of the models on the BEA-2019 dev part, which is presented in Table 4.12. Each time value is the arithmetic mean of the time of 5 model inferences.

Encoder	Vocabulary size	Time
RoBERTa base	5k	19.05 s
BERT base	5k	19.28 s
DeBERTa base	5k	23.75 s
XLNet base	5k	30.46 s
RoBERTa base	10k	20.46 s
RoBERTa large	1k	45.66 s
RoBERTa large	5k	47.23 s
BERT large	5k	49.17 s
DeBERTa large	5k	58.32 s
XLNet large	5k	71.19 s
RoBERTa large	10k	48.06 s

TABLE 4.12: Comparing the impact of transformer-based encoders and vocabulary size on the inference time

Absolute time values may differ on different computers, but the order should be the same. We did not use any optimizations for inference, like weights quantization for exmaple, so this time can be improved.

Among the base models with the 5k dictionary, the fastest was the RoBERTa base model. The BERT base model had almost the same speed, the DeBERTa base model was a bit slower, and the XLNet base model was the slowest.

A similar case was for large models with a 5k dictionary. The fastest was RoBERTa large, the slowest was the XLNet large model.

If we talk about the size of the dictionary, then, as expected, for a smaller dictionary - the prediction time is shorter, while for a larger dictionary, the prediction time is longer.

It is worth noting that although large models have better results, they are much slower. The prediction time for large models is twice as long as for base models. This is because large models have twice as many parameters as base models.

4.5 Ensembles

As already shown in the literature review, many studies show the final result obtained with a single model and an ensemble of models. Combining the results of models that differ from each other helps to improve the quality of corrections.

There are two main ensembling methods, such as stacking on model and data levels.

The first approach averages the probabilities which produce models during the prediction. In the case of the sequence tagging models, it averages the probabilities of tag assignment. All models must have the same size and order of tags in the dictionary to perform this averaging. Such an ensemble method was used in (Omelianchuk et al., 2020).

Ensemble	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Base RoBERTa + XLNet	53.45	34.3	48.08
Base RoBERTa + DeBERTa	53.44	**34.91**	48.31
Base RoBERTa + XLNet + DeBERTa	54.78	34.87	49.17
Base RoBERTa + XLNet + DeBERTa +BERT	**56.34**	33.76	**49.69**
Large RoBERTa + XLNet	53.83	38.65	49.91
Large RoBERTa + DeBERTa	54.12	39.77	50.48
Large RoBERTa + XLNet + DeBERTa	54.3	**39.95**	50.66
Large RoBERTa + DeBERTa + BERT	**57.31**	37.41	51.8
Large RoBERTa + XLNet + DeBERTa +BERT	56.97	38.52	**51.99**
Base RoBERTa + lage RoBERTa	54.83	35.93	49.61
Base and large RoBERTa + XLNet + DeBERTa +BERT	**58.69**	**36.12**	**52.17**

TABLE 4.13: Comparing model level ensembles based on models trained on the joined dataset

The second approach combines the corrections in the post-processing phase. First, each model outputs corrected sentences individually. Then, we take the corrected sentences and compare which corrections they had in common. For example, if at least two of the three models have made the same correction, we can make this correction.

A somewhat similar approach was used in (Liang et al., 2020), in which the authors combined sequence tagging and sequence-to-sequence models for the Chinese language. The advantage of this ensemble method is that we can combine the results of models with different architectures. In our work, this allowed, for example, to combine the results of models with different vocabulary sizes, which was impossible with ensembling at the model level.

First, we explored a model-level ensembling approach. We took models with the same 5k dictionary, which were trained on the Joined dataset. The results of combining different combinations of models can be seen in Table 4.13.

As we can see, the ensemble improves the quality of corrections, and the more models we combine, the better the result we have.

Another interesting fact was that if we combine models with large and base configurations of the same encoder, as was done for RoBERTa, we can get a slightly better result than we got for the base and large models separately.

Although the ensemble RoBERTa + DeBERTa + XLNet + BERT gave the best result, we decided not to use it. This ensemble had a much smaller Recall than the ensemble RoBERTa + DeBERTa + XLNet, which would not allow a further increase $F_{0.5}$ due to hyperparameter optimization that would further reduce the Recall. Therefore, we used an ensemble of models with RoBERTa + DeBERTa + XLNet encoders for further research.

Next, we compared the ensembles at the model and data level for trained models on the Joined dataset. A comparison of the results can be seen in Table 4.14.

Ensemble	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
base RoBERTa + XLNet + DeBERTa (model)	54.78	**34.87**	49.17
base RoBERTa + XLNet + DeBERTa (data)	**56.44**	33.24	**49.53**
large RoBERTa + XLNet + DeBERTa (model)	54.3	**39.95**	50.66
large RoBERTa + XLNet + DeBERTa (data)	**56.74**	38.53	**51.84**

TABLE 4.14: Comparing model and data level ensembles based on models trained on the joined dataset

The ensemble, based on post-processing, gave the better result on $F_{0.5}$ metrics as Precision became larger and Recall - smaller.

Next, we compared the results of large models that were fine-tuned on the WI dataset. The results can be seen in Table 4.15.

Ensemble	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
large RoBERTa + XLNet + DeBERTa (model)	58.08	**43.17**	54.33
large RoBERTa + XLNet + DeBERTa (data)	**60.58**	41.92	**55.63**

TABLE 4.15: Comparing model and data level ensembles based on models fine-tuned on the WI dataset

The ensemble based on the data level had the better result of the $F_{0.5}$ metric.

Finally, we selected the additional confidence for the KEEP token and minimal error probability based on the grid search for both ensembles on model and date levels. The comparison is shown in Table 4.16.

Ensemble	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
large RoBERTa + XLNet + DeBERTa (model)	68.45	**35.56**	57.76
large RoBERTa + XLNet + DeBERTa (data)	**69.67**	34.51	**57.88**

TABLE 4.16: Comparing model and data level ensembles based on models after the selection of hyperparameters

In the final result, both ensemble approaches had almost the same value of the F.05 metric.

Since the method based on post-processing may combine any models, we decided to test the ensemble of the best models that we trained. The results of this comparison are shown in Table 4.18.

Ensemble	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
large RoBERTa 5k + XLNet 5k + DeBERTa 5k (data)	69.67	34.51	57.88
large RoBERTa 10k + XLNet 10k + DeBERTa 10k (data)	70.13	34.23	57.97
large RoBERTa 10k + XLNet 5k + DeBERTa 5k (data)	**70.71**	33.78	58.02
large RoBERTa 10k + XLNet 5k + DeBERTa 10k (data)	70.32	**34.62**	**58.3**

TABLE 4.17: Comparing data level ensembles based on best trained models

The ensemble, based on the model with large encoder Roberta + vocabulary size 10k, the model with large XLNet encoder + vocabulary size 5k, and the model with DeBERTa + vocabulary size 10k, achieved the highest result, 58.3 at BEA-2019 dev part.

Ensemble	BEA-2019 (test)		
	Precision	*Recall*	$F_{0.5}$
large RoBERTa 10k + XLNet 5k + DeBERTa 10k (data)	**84.44**	**54.42**	**76.05**

TABLE 4.18: Comparing data level ensembles based on best trained models

We also evaluated this ensemble on the BEA-2019 test part, where it reached 76.05 of $F_{0.5}$ score. This result is much higher than those we have met in the existing GEC papers.

4.6 Conclusion

We have shown that models with large encoder configurations had better error correction quality than models with basic encoder configurations. However, improving the quality slows down the inference time of the models. Models with large encoder configurations are more than twice as slow as models with basic configurations.

We have also shown that models based on the RoBERTa, DeBERTa, and XLNet encoders perform better than models based on the BERT encoder for both configurations.

We have discovered that increasing the vocabulary size for models with RoBERTa and DeBERTa large encoders improves the quality of corrections. In contrast, for the model with XLNet large encoder, the quality has deteriorated.

Finally, we showed that ensembling approaches on model and date levels have the same result when combining models with the same vocabulary size. However, the post-processing ensemble approach has the advantage that we can combine any models, regardless of their structure.

Our ensemble based on the model with large encoder Roberta + vocabulary size 10k, the model with large XLNet encoder + vocabulary size 5k, and the model with DeBERTa + vocabulary size 10k reached a value 76.05 for the metric F0.5 on the BEA-2019 test part. This result is a significant improvement over 73.7, which was previously achieved in (Omelianchuk et al., 2020). It is worth noting that this result was obtained without pre-training on synthetic data.

Chapter 5

Data filtering

The quantity and quality of training data play an essential role in training models: the more quality training data, the better the results. This section would first explore the quality of existing GEC training data and suggest methods for cleaning and filtering noisy data.

5.1 Basic data filtering

We started with a simple data cleanup that could potentially filter out sentences that most likely are not grammatically correct. We decided to remove the repetition of identical sentences. Also, to filter out those sentences for which:

1. The target sentences were empty, containing less than five letters, or only one token.
2. The target sentences started with a lowercase word (assumption - that these were fragments of sentences)
3. The target sentences contained all words in uppercase (also anomalous sentences).

We also decided to filter sentences for which the cosine similarity between the source and target sentences was less than 0.5. We expected that their source sentences could be significantly damaged or semantically different from the target sentence.

$$\cos(\mathbf{a},\mathbf{b}) = \frac{\mathbf{ab}}{\|\mathbf{a}\|\|\mathbf{b}\|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n}(a_i)^2}\sqrt{\sum_{i=1}^{n}(b_i)^2}} \tag{5.1}$$

An example of several training sentences for which the cosine similarity was less than 0.5 can be seen in Table 5.2.

Source sentence	Target sentence	Cosine similarity
So I do easy to the society 's exchange .	I do n't understand this phrase .	-0.1298
I am stydying envaermantal saents .	I am studying environmental science .	0.0576
Good Morning :	Dear Sir / Madam	0.1863

TABLE 5.1: An example of sentences for which cosine similarity between the source and target sentences have less than 0.5.

We decided to choose the pre-trained sentence RoBERTa base (Reimers and Gurevych, 2019) model from the Sentence Transformers [1] library to vectorize the sentences. This

[1] https://github.com/UKPLab/sentence-transformers

model is trained to help distinguish the semantic similarity of sentences. The results of sentence filtering for Lang-8, NUCLE, FCE, and WI datasets are presented in Table 5.

	Lang-8	**NUCLE**	**FCE**	**WI**
Initial total size	1037561	57151	28350	34308
Drop duplicates	86675	3470	2643	815
Filter minimal length	9502	406	157	291
Drop start from non-capital	55162	462	124	291
Drop all capital	3609	263	393	98
Cosine similarity less than 0.5	6407	62	156	82
Size after filtering	876206	52488	24877	32731
Rate of filtered data	16%	8%	12%	5%

TABLE 5.2: Data cleaning for datasets

It is worth noting that the table shows the number of all sentences in the datasets, regardless of whether they differ between source and target sentences or not.

First, we took the RoBERTa base model, which trained on an uncleaned joined dataset, and tried to fine-tune it on a cleaned WI dataset. This fine-tuning gave a slight increase in quality, which is presented in Table 5.3.

Experiment	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Fine-tune on WI	53.77	39.23	50.06
Fine-tune on cleaned WI	**53.86**	**39.43**	**50.19**

TABLE 5.3: Fine-tuning RoBERTa base model on basic and cleaned WI dataset

Next, we decided to train the model with the encoder RoBERTa base and vocabulary size 5k on the filtered data of the joined dataset. We compared it with the model, which trained on the data without any cleaning. The results are shown in Table 5.4.

Experiment	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Train on basic joined dataset	50.1	34.22	45.84
Train on cleaned joined dataset	**51.74**	**35.0**	**47.22**

TABLE 5.4: Comparing model training on basic and cleaned joined dataset

It can be seen that the model trained on cleaned data had a slightly better value of metrics Precision, Recall, and $F_{0.5}$.

However, after fine-tuning on the WI dataset, the final result was almost the same for both models trained with and without data cleaning. These results presented in Table 5.5.

Dataset	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Without filtering	**53.77**	39.23	**50.06**
With filtering	53.44	**39.78**	50.01

TABLE 5.5: Comparing models after fine-tuning on basic and cleaned WI dataset

5.2 Data selection based on clusters similarity

In the following method, we decided to expand the WI dataset with the most similar sentences from the Lang-8, NUCLE, and FCE datasets and compare fine-tuning models on such data with baseline results. To select the most similar sentences, we first grouped the sentences into clusters using hierarchical clustering based on the cosine similarity of the sentence vectors.

To obtain the vectors of target sentences, we again used the sentence base RoBERTa model mentioned in Section 5.1.

Next, for each cluster, we calculated the averaged centroid vectors. We compared the cluster centroids' similarity between the WI dataset and Lang-8, NUCLE, and FCE datasets. We selected the closest clusters to WI based on cosine similarity, and from these clusters, we finally selected sentences for the extension WI dataset.

Then, we compared how the number of sentences that we add to WI influences model fine-tuning. The results for models RoBERTa base and RoBERTa large presented in Tables 5.6 and 5.7.

Experiment	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Train on WI	53.77	**39.23**	50.06
Train on WI + 5k (clustering)	**55.03**	37.9	**50.47**
Train on WI + 10k (clustering)	54.63	37.61	50.1
Train on WI + 15k (clustering)	54.6	36.98	49.85

TABLE 5.6: Fine-tuning RoBERTa base on the extended WI dataset. Additional data were selected from the most similar clusters to WI data.

Experiment	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Train on WI	54.85	**42.54**	51.85
Train on WI + 5k (clustering)	**55.91**	41.67	**52.33**
Train on WI + 10k (clustering)	55.18	42.02	51.93
Train on WI + 15k (clustering)	55.21	41.55	51.8

TABLE 5.7: Fine-tuning RoBERTa large on the extended WI dataset. Additional data were selected from the most similar clusters to WI data.

As we can see, adding 5k close sentences based on clustering allowed to increase Precision but significantly reduced Recall. Further expansion of training data, on the contrary, only worsened the result.

Although training on such data did not significantly improve results, it allowed identifying clusters of sentences with poor quality data, such as clusters that consist only of punctuation or clusters that contain only URL links or date of annotation and others.

We planned to discover the impact of such data cleaning in future works.

5.3 Influence of data samples between fine tuning

We also tried another approach to extend WI with training sentences. It is based on comparing the model's sentence embeddings obtained before and after fine-tuning on the WI dataset.

For this experiment, we used embeddings from our trained GEC model with base RoBERTa encoder and vocabulary size 5k.

For those sentences for which the similarity between embeddings has slightly changed, we expected that they should be of better quality than sentences with significant deviation in similarity. Then we took the sentences with the slightest change in cosine similarity and expanded with them the WI dataset. The results can be seen in Tables 5.8 and 5.9.

Experiment	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Train on WI	53.77	**39.23**	**50.06**
Train on WI + 5k (checkpoints)	**54.26**	38.06	50.01
Train on WI + 10k (checkpoints)	53.5	37.68	49.36

TABLE 5.8: Fine-tuning RoBERTa base on the extended WI dataset. Additional data were selected based on the cosine similarity of sentence embeddings, which was produced by two checkpoints of the model before and after fine-tuning.

Experiment	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Train on WI	54.85	**42.54**	51.85
Train on WI + 5k (checkpoints)	**55.93**	41.23	**52.2**
Train on WI + 10k (checkpoints)	55.31	40.77	51.63

TABLE 5.9: Fine-tuning RoBERTa large on the extended WI dataset. Additional data were selected based on the cosine similarity of sentence embeddings, which was produced by two checkpoints of the model before and after fine-tuning.

As we can see, adding 5k sentences to WI also helped increase Precision and reduce Recall. Adding more sentences worsened the results.

This method also helped to identify noisy sentences in training data, for which we had the most significant deviation in similarity. An example of such sentences can be seen in Table 5.10. We want to discover the impact of such data cleaning also in future works.

Source sentence	**Target sentence**	**Cosine similarity**
SLUNG	SLUNG Slang	0.8814
CONCEIVE	ONCEIVE v . ENVISION	0.8875
(Khalik , S. (2009 , August 17)	(Khalik , 2009) .	0.934
" CITATION Tan00 l 1033 (Tanzi .	" (Tanzi .	0.9346

TABLE 5.10: An example of sentences with a significant deviation of cosine similarity between embeddings produced by model checkpoints before and after fine-tuning.

5.4 Conclusion

This section analyzed the quality of existing GEC datasets and proved that the WI dataset has the highest quality. Although the proposed methods of extending the WI dataset with additional data did not significantly improve the result, these methods helped reveal low-quality training data samples.

Chapter 6

Knowledge distilled data

In this chapter, we would like to discover an approach to mark a new GEC training data based on the knowledge distillation from an ensemble of models.

6.1 Approach for data generation

The idea of the approach was to first train the models on existing GEC data, then combine them into an ensemble of models that would provide a better quality of grammar correction. The next step is to take texts that may contain grammatical errors and annotate them with an ensemble of models. In this way, we could obtain new synthetic data on which the student GEC model could then be trained.

The advantage of this approach is that it is quite simple and versatile. You do not need to invent logic on how to make synthetic corrections, because the corrections will be taken from real human texts.

The drawback is that we can only detect a limited number of corrections, only those that the model ensemble can correct. Additional types of corrections in this way will not be possible to learn during training.

The second disadvantage is that the model ensemble is imperfect and can make false positive corrections. And during the training, the student model will also be studied on false-positive corrections, which is not good.

Despite the shortcomings, this method still generates new training data, and in the following paragraphs we want to test the strategies for their use. We expect that training on such data will allow the model to quickly learn to correct common errors, and during training on high-quality annotated data - to focus on those corrections that previously could not be studied.

6.2 Generated datasets overview

In section 3.3, we discussed the nature of the data we chose for data markup.

The first dataset is One Billion Words Benchmark (1BW), which consists mainly of the news.

The second dataset is The Blog Authorship Corpus (Blogs), which consists of blog texts on various topics, and the authors are of different ages.

And the third dataset is Amazon reviews, in which customers gave feedback on the goods they bought.

Each dataset contained a different rate of grammatical errors. The percentage of sentences in which the ensemble found changes presented in Table 6.1.

It is also worth noting that the 1BW and Amazon datasets contain a huge amount of text, while the Blogs dataset contains a much smaller amount. It can be a limitation if we consider generating more data.

Dataset	Processed sentences	Sentences with changes	Rate of changes
1BW	23 156 858	1 196 622	5.16 %
Blogs	6 191 342	1 704 229	27.52 %
Amazon	5 236 480	1 469 285	28.06 %

TABLE 6.1: Statistics on how many sentences were processed by ensemble and rate of sentences with found grammatical errors

For markup, we used a model level ensemble of sequence tagging GEC models with encoders RoBERTa large, DeBERTa large, XLNet large, and a 5k dictionary size.

As for marking the ensemble based on post-processing by models with a larger size of the dictionary, we left it for future research.

It is also worth noting that before the markup, the data had to be tokenized (punctuated by spaces) and blocks of sentences had to be divided into separate sentences. For preprocessing, we used a tokenizer from Spacy[1].

After generating the data, we trimmed each dataset to 1.2 M sentences so that the dimensions of the generated datasets were the same and the model quality comparison did not affect the amount of training data.

6.3 Pre-training using generated data

First, we tested the quality of the training on the generated data using the RoBERTa base model. In all subsequent experiments, we used a 5k dictionary, which was also used by the ensemble to mark the data.

Dataset	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
1BW	49.88	35.37	46.1
Blogs	49.59	**39.14**	**47.08**
Amazon	**52.6**	32.26	46.75

TABLE 6.2: Comparison of model training on 1BW, Blogs, Amazon datasets (stage 1) based on RoBERTa base model

During the pre-training, it was important that the model studied as many corrections as possible (i.e. had a larger Recall), so that in the end we could increase the accuracy by selecting hyperparameters.

The model received the largest Recall from training on the Blogs dataset, a slightly smaller one on the 1BW dataset, and the smallest on the Amazon dataset.

This can be explained by the fact that the Blogs dataset consists of texts on different topics, different data quality, and as a result, contains a greater variety of types of errors.

Although the texts in the 1BW dataset are also of various topics, they follow the same information and journalistic style.

As for the Amazon reviews dataset, its full version contains reviews on various product categories. However, we only used the subcategory "Clothing, shoes and jewelry" to mark the training data, and it is possible that the use of only one category resulted in the same type of text and less variety of errors. In further research,

[1]https://spacy.io/

we could generate a dataset based on different product categories, and test this assumption.

For further experiments, we decided to compare the results of the training on 1BW and Blogs datasets.

First, we decided to test the training scheme used in (Omelianchuk et al., 2020). They first trained their models on synthetically generated data (stage 1), then trained on the joined dataset (stage 2), then trained only on the high-quality WI (stage 3) dataset, and finally made a selection of hyperparameters to increase Precision by reducing Recall during model interference (Inf).

Instead of PIE synthetic data(Awasthi et al., 2019), we used datasets that were generated by ensembles. The results of training at different stages for models with RoBERTa base and RoBERTa large encoders can be seen in Tables 6.3, 6.4, 6.5, 6.6.

Stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	49.88	35.37	46.1
Stage 2	50.34	37.19	47.02
Stage 3	**55.16**	**39.32**	**51.05**
Inf	–	–	–

TABLE 6.3: Pre-training on 1BW dataset for RoBERTa base

Stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	49.59	39.14	47.08
Stage 2	51.77	35.99	47.6
Stage 3	54.53	**40.54**	51.01
Inf	**67.51**	30.97	**54.63**

TABLE 6.4: Pre-training on Blogs dataset for RoBERTa base

Stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	51.18	39.24	48.25
Stage 2	52.01	39.45	48.89
Stage 3	55.91	**42.27**	52.52
Inf	**67.05**	32.42	**55.25**

TABLE 6.5: Pre-training on 1BW dataset for RoBERTa large

Training at different stages had a very interesting behavior. Pre-training on stage 1 on both datasets allowed us to achieve a slightly higher result in just two epochs than if we simply trained models on the joined dataset without pre-training.

However, when we started training on stage 2, there was a sharp deterioration in results, and it seemed that the model made almost no use of the information it had learned on stage 1. This is especially true for models trained on the Blogs dataset, which significantly dropped Recall during training.

Stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	50.36	40.97	48.16
Stage 2	52.8	39.28	49.4
Stage 3	55.98	**41.31**	52.26
Inf	**66.22**	34.26	**55.81**

TABLE 6.6: Pre-training on Blogs dataset for RoBERTa large

But still, with using pre-training, on stage 2 the $F_{0.5}$ metric was better than without using pre-training, and the models learned much faster, they needed fewer epochs.

After pre-training on stage 3, the result was also slightly better than for models without pre-training. However, when we tried to select the hyperparameters for the pre-trained model with the RoBERTa large encoder, we obtained almost the same result as for the model with RoBERTa large encoder, for which no pre-training was performed.

It is also worth noting that on stage 3 there were almost no differences in the result of the models, depending on which dataset we pre-trained them, whether on Blogs or 1BW datasets.

6.4 Pre Training using synthetic public data

Next, we pre-trained models based on a RoBERTa base and large encoders using synthetic training data proposed in (Awasthi et al., 2019), which were also used to train the GECToR model (Omelianchuk et al., 2020).

Our results for the RoBERTa base and RoBERTa large models are shown in Tables 6.7 and 6.8, respectively.

Stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	40.88	22.66	35.22
Stage 2	49.86	36.93	46.59
Stage 3	54.32	**42.25**	51.38
Inf	**65.48**	33.66	**55.06**

TABLE 6.7: Pre-training on PIE-synthetic data for RoBERTa base

Stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	43.79	20.94	35.94
Stage 2	46.71	43.24	45.97
Stage 3	53.22	**45.25**	51.41
Inf	**66.47**	34.0	**55.81**

TABLE 6.8: Pre-training on PIE-synthetic data for RoBERTa large

Pre-training on such synthetic data had a completely different behavior compared to the results obtained in paragraph 6.4.

On stage 1, the models trained very slowly and had a much lower $F_{0.5}$ metric than our pre-training models. However, during training on stage 2, the models continued to train slowly with growing Recall, albeit with less Precision. On stage 3, the models had a smaller metric of $F_{0.5}$, compared to the models in paragraph 6.4, but a significantly bigger value of Recall. After selecting the hyperparameters for such models, we could obtain a higher value of $F_{0.5}$ compared to models pre-trained on 1BW dataset and almost the same value for large model pre-trained on Blogs dataset.

It is also worth noting that the results our obtained results for RoBERTa base are quite similar and slightly better than those obtained in (Omelianchuk et al., 2020), which are shown in Table 6.9.

The difference can be explained by the use of the original tokenizer, which was explained in paragraph 4.2.

Training stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	40.8	22.1	34.9
Stage 2	51.6	33.8	46.7
Stage 3	54.2	**41.0**	50.9
Inf	**62.3**	35.1	**54.0**

TABLE 6.9: Pre-training on PIE-synthetic data for RoBERTa base (Omelianchuk et al., 2020)

6.5 Combining distilled data with synthetic data

As the next step, we decided to combine the synthetic data from paragraph 6.5 with our generated data from paragraph 6.4 and train according to the same scheme as described earlier in this section.

To the 1.2 M data that we generated, we added 1.2 M synthetic data to have an equal proportion of training samples.

The results of such training are shown in Tables 6.10, 6.11, 6.12.

Stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	43.64	40.21	42.9
Stage 2	51.23	37.42	47.71
Stage 3	54.08	**41.91**	51.11
Inf	**65.38**	33.41	**54.88**

TABLE 6.10: Pre-training on PIE+1BW data for RoBERTa base

The behavior of the models during training is very similar to the behavior of the models from paragraph 6.4.

However, with this pre-training, we managed to get the best result of the $F_{0.5}$ metric on the BEA-2019 dev part for the single model roberta-large with a dictionary size of 5k, which we trained on the combined 1BW and PIE synthetic data.

Stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	47.57	40.91	46.07
Stage 2	50.8	41.29	48.56
Stage 3	55.22	**43.63**	52.43
Inf	**67.53**	33.59	**56.18**

TABLE 6.11: Pre-training on PIE+1BW data for RoBERTa large

Stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	51.5	40.49	48.84
Stage 2	52.46	39.75	49.31
Stage 3	55.66	**42.7**	52.48
Inf	**65.78**	35.32	**56.1**

TABLE 6.12: Pre-training on PIE+Blogs data for RoBERTa large

At the BEA-2019 test, this model reached a value of 73.21 in $F_{0.5}$ metric, which significantly improved the result for the single model.

6.6 Training in one stage

During the experiments in section 6.4, we noticed that models with pre-training on the Blogs dataset train fairly quickly and have good results on stage 1, but lose this advantage after training on stage 2.

Stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	53.98	**40.28**	50.54
Inf	**64.6**	32.74	**54.07**

TABLE 6.13: Pre-training on Blogs+WI data for RoBERTa base

We decided to combine the data generated from the Blogs dataset with the high-quality data of the WI dataset. The results of such training are shown for models with roberta-base and roberta-large encoders in Tables 6.13 and 6.14, respectively.

Thus, we were able to significantly simplify the training scheme, replacing it with only one stage, and achieve even better results, compared to the baseline models that we trained in section 4.2.

However, when we tried to fine-tune these models on the WI dataset, further training only worsened the $F_{0.5}$ metric on the BEA-2019 dev part.

Seeing improvements for the RoBERTa large model, we also tried to train the DeBERTa large and XLNet large models. The results can be seen in Table 6.15.

For models with DeBERTa large and XLNet large encoders, the results were almost the same or slightly worse compared to the baseline models in section 4.2. This shows that the distilled data generated by us are of fairly high quality, and further their research and use can help to improve existing GEC models.

Stage	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
Stage 1	55.98	**42.21**	52.55
Inf	**67.06**	33.4	**55.81**

TABLE 6.14: Pre-training on Blogs+WI data for RoBERTa large

Model	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa	**55.98**	**42.21**	**52.55**
DeBERTa	54.05	41.95	51.15
XLNet	53.01	41.66	50.27

TABLE 6.15: Pre-training on Blogs+WI data in one stage for large encoders

6.7 Conclusion

In this chapter, we explained how to generate new high-quality training data based on an ensemble of models. We investigated and compared different strategies for using this generated data, as well as comparing them with existing synthetic data. Pre-training with the distilled data generated by us improved the result, compared to baseline models for which we did not perform pre-training.

Pre-training with the distilled data generated by ensemble improved the result, compared to baseline models for which we did not perform pre-training. However, pre-training with public synthetic data had a better effect on the final result than pre-training with the data generated by ensemble. The combination of synthetic PIE data with the data generated by ensemble allowed to get our best result so far for the single model on the BEA-2019 dev part. At the BEA-2019 test part, this model reached 73.21, which is an improvement for the single model, compared to the previous result of the best GECToR model. Finally, we showed that the data generated by the ensemble allowed us to get good results in only one stage, proving the excellent quality of our data and significantly reduce training time and simplify the training schema.

Chapter 7

Conclusions

7.1 Contribution

In our work, we explored the sequence tagging approach.

1. We compared the impact of transformer-based encoders BERT, RoBERTa, DeBERTa, XLNet of base and large configurations on grammatical error correction. Models based on RoBERTa and DeBERTa encoders achieved the best results, in both base and large configurations.
2. We have shown that models with large encoder configurations perform better than models with base configurations, but their interference time is almost twice as slow.
3. We have discovered that increasing the vocabulary size for models with RoBERTa and DeBERTa large encoders improves the quality of corrections. In contrast, for the model with XLNet large encoder, the quality has deteriorated.
4. We showed that ensembling models on model and data levels have almost the same result when combining models with the same vocabulary size. However, the ensemble on the data level has the advantage that we can combine the output of any models, regardless of their structure.
5. Our best ensemble without pre-training on the synthetic data achieves a new SOTA result of an $F_{0.5}$ 76.05 on BEA-2019 (test), in contrast, when the newest obtained results were achieved with pre-training on synthetic data.
6. We analyzed the quality of existing training data, proposed methods for identifying low quality data and filtering them.
7. We investigated the approach of generating training data based on the distillation of knowledge by an ensemble of models. It was shown that such data are of high quality, and combining them with a WI dataset allows to achieve in one step the same results as training on the joined dataset. The further research and usage of such generated data might help to improve existing GEC models.
8. We have shown that pre-training on combined data generated by an ensemble of models and existing public synthetic data allowed to improve the result of the models. Our best single model (RoBERTa large) with pre-training on this combined synthetic data achieved a value of 73.21 of an $F_{0.5}$ metric on the BEA-2019 test part.
9. Our code, generated datasets, and trained models are publicly available[1].

[1] https://github.com/MaksTarnavskyi/gector-large

7.2 Future work

1. To investigate the impact of even larger transformer configurations. In particular, for the DeBERTa encoder, explore the xlarge and xxlarge configurations. These experiments were not performed due to the RAM size limitations of our GPU.

2. To generate data with a better ensemble of models and explore their impact. In particular, use models with a larger vocabulary size and pre-trained on synthetic data.

3. We also want to explore more strategies for using the data we generate. For example, combine the generated data with clean joined datasets and see how this will affect model training.

4. To explore different ensemble strategies on a data level. In particular, what will happen if we apply all the changes that do not contradict each other.

5. Try to combine sequence tagging with the sequence-to-sequence GEC model by the ensemble on the data level.

6. To explore how to improve the tagging space of vocabulary. In particular, think about adding additional custom transformation tags.

7. To experiment on the model's architecture, try to increase the number of linear layers and explore different ways to combine token level embeddings with the word level embeddings.

8. Try to train the sequence tagging GEC model for low-resource languages, in particular, for Ukrainian. To do that, we plan to use the Ukrainian Roberta[2] and the Ukrainian GEC corpus (Syvokon and Nahorna, 2021).

[2] `https://github.com/youscan/language-models`

Appendix A

Baseline approach

Experiment	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa base without 2 cold steps	44.34	**34.73**	42.02
RoBERTa base with 2 cold steps	**50.12**	34.04	**45.79**
RoBERTa large without 2 cold steps	46.33	36.84	44.06
RoBERTa large with 2 cold steps	**52.11**	**37.34**	**48.29**

TABLE A.1: Training with and without 2 cold steps

Experiment	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa base trained in 2 stages (Joined => WI)	**53.77**	**39.23**	**50.06**
RoBERTa base trained only on WI	11.69	5.4	9.48

TABLE A.2: Ablation study, when we trained model in two stages and when we only trained the model on the WI dataset

Experiment	BEA-2019 (dev)		
	Precision	*Recall*	$F_{0.5}$
RoBERTa base trained with filtering dataset	50.12	**34.04**	**45.79**
RoBERTa base trained without filtering dataset	**51.93**	28.7	44.69

TABLE A.3: A comparison of the training model with and without filtering sentences in which all tags are only KEEP tag.

Bibliography

Awasthi, Abhijeet et al. (2019). "Parallel Iterative Edit Models for Local Sequence Transduction". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 4259–4269.

Bryant, Christopher, Mariano Felice, and Ted Briscoe (2017). "Automatic Annotation and Evaluation of Error Types for Grammatical Error Correction". In: *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Vol. 1, pp. 793–805.

Bryant, Christopher et al. (2019). "The BEA-2019 Shared Task on Grammatical Error Correction." In: *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 52–75.

Chelba, Ciprian et al. (2013). "One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling". In: *CoRR* abs/1312.3005. arXiv: 1312.3005. URL: http://arxiv.org/abs/1312.3005.

Chen, Mengyun et al. (2020). "Improving the Efficiency of Grammatical Error Correction with Erroneous Span Detection and Correction". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 7162–7169.

Cho, Kyunghyun et al. (2014). "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734.

Dahlmeier, Daniel and Hwee Tou Ng (2012). "Better Evaluation for Grammatical Error Correction". In: *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 568–572.

Dahlmeier, Daniel, Hwee Tou Ng, and Siew Mei Wu (2013). "Building a Large Annotated Corpus of Learner English: The NUS Corpus of Learner English". In: *Proceedings of the Eighth Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 22–31.

Devlin, Jacob et al. (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186.

He, Pengcheng et al. (2020). "DeBERTa: Decoding-enhanced BERT with Disentangled Attention". In: *arXiv preprint arXiv:2006.03654*.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). "Long short-term memory". In: *Neural Computation* 9.8, pp. 1735–1780.

Kairouz, Peter et al. (2019). "Advances and Open Problems in Federated Learning". In: *arXiv: Learning*.

Kaneko, Masahiro et al. (2020). "Encoder-Decoder Models Can Benefit from Pre-trained Masked Language Models in Grammatical Error Correction". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4248–4254.

Kao, Ting hui et al. (2013). "CoNLL-2013 Shared Task: Grammatical Error Correction NTHU System Description". In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pp. 20–25.

Kiyono, Shun et al. (2019). "An Empirical Study of Incorporating Pseudo Data into Grammatical Error Correction". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 1236–1242.

Lample, Guillaume and Alexis Conneau (2019). "Cross-lingual Language Model Pretraining." In: *arXiv preprint arXiv:1901.07291*.

Lan, Zhenzhong et al. (2020). "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *ICLR 2020 : Eighth International Conference on Learning Representations*.

Liang, Deng et al. (2020). "BERT Enhanced Neural Machine Translation and Sequence Tagging Model for Chinese Grammatical Error Diagnosis". In: *Proceedings of the 6th Workshop on Natural Language Processing Techniques for Educational Applications*, pp. 57–66.

Lichtarge, Jared, Chris Alberti, and Shankar Kumar (2020). "Data Weighted Training Strategies for Grammatical Error Correction." In: *Trans. Assoc. Comput. Linguistics* 8, pp. 634–646.

Lichtarge, Jared et al. (2019). "Corpora Generation for Grammatical Error Correction". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 3291–3301.

Liu, Yinhan et al. (2019). "RoBERTa: A Robustly Optimized BERT Pretraining Approach". In: *arXiv preprint arXiv:1907.11692*.

Malmi, Eric et al. (2019). "Encode, Tag, Realize: High-Precision Text Editing". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 5053–5064.

Naber, Daniel (Jan. 2003). "A Rule-Based Style and Grammar Checker". In:

Napoles, Courtney, Keisuke Sakaguchi, and Joel R. Tetreault (2017). "JFLEG: A Fluency Corpus and Benchmark for Grammatical Error Correction". In: *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 229–234.

Ng, Hwee Tou et al. (2014). "The CoNLL-2014 Shared Task on Grammatical Error Correction". In: *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pp. 1–14.

Ni, Jianmo, Jiacheng Li, and Julian J. McAuley (2019). "Justifying Recommendations using Distantly-Labeled Reviews and Fine-Grained Aspects." In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 188–197.

Omelianchuk, Kostiantyn et al. (2020). "GECToR – Grammatical Error Correction: Tag, Not Rewrite". In: *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 163–170.

Radford, Alec et al. (2018). "Language Models are Unsupervised Multitask Learners". In: URL: https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf.

Reimers, Nils and Iryna Gurevych (Nov. 2019). "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on*

Empirical Methods in Natural Language Processing. Association for Computational Linguistics. URL: https://arxiv.org/abs/1908.10084.

Schler, Jonathan et al. (2005). "Effects of Age and Gender on Blogging". In: *AAAI Spring Symposium: Computational Approaches to Analyzing Weblogs*, pp. 199–205.

Stahlberg, Felix and Shankar Kumar (2020). "Seq2Edits: Sequence Transduction Using Span-level Edit Operations". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 5147–5159.

— (2021). "Synthetic Data Generation for Grammatical Error Correction with Tagged Corruption Models". In: *Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications*, pp. 37–47.

Syvokon, Oleksiy and Olena Nahorna (2021). *UA-GEC: Grammatical Error Correction and Fluency Corpus for the Ukrainian Language*. arXiv: 2103.16997 [cs.CL].

Tajiri, Toshikazu, Mamoru Komachi, and Yuji Matsumoto (2012). "Tense and Aspect Error Correction for ESL Learners Using Global Context". In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Vol. 2, pp. 198–202.

Vaswani, Ashish et al. (2017). "Attention is All You Need". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Vol. 30, pp. 5998–6008.

Wang, Yu et al. (2020). "A Comprehensive Survey of Grammar Error Correction." In: *arXiv preprint arXiv:2005.06600*.

Yang, Zhilin et al. (2019). "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *Advances in Neural Information Processing Systems*. Vol. 32, pp. 5753–5763.

Yannakoudakis, Helen, Ted Briscoe, and Ben Medlock (2011). "A New Dataset and Method for Automatically Grading ESOL Texts". In: *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pp. 180–189.

Yuan, Zheng and Ted Briscoe (2016). "Grammatical error correction using neural machine translation". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 380–386.

Yuan, Zheng and Mariano Felice (2013). "Constrained Grammatical Error Correction using Statistical Machine Translation". In: *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pp. 52–61.

Zhu, Jinhua et al. (2020). "Incorporating BERT into Neural Machine Translation". In: *ICLR 2020 : Eighth International Conference on Learning Representations*.