Improved grammatical error correction by ranking elementary edits

Alexey Sorokin
Yandex
Moscow State University
alexey dot sorokin at list dot ru

Abstract

We offer a two-stage reranking method for grammatical error correction: the first model serves as edit generator, while the second classifies the proposed edits as correct or false. We show how to use both encoder-decoder and sequence labeling models for the first step of our pipeline. We achieve state-of-the-art quality on BEA 2019 English dataset even using weak BERT-GEC edit generator. Combining our roberta-base scorer with state-of-the-art GECToR edit generator, we surpass GECToR by $2-3\%$. With a larger model we establish a new SOTA on BEA development and test sets. Our model also sets a new SOTA on Russian, despite using smaller models and less data than the previous approaches.

1 Introduction

Grammatical error correction (GEC) is the task of converting the source text to its clean version with no orthographic, punctuation, lexical or other errors. It becomes increasingly popular during last years due to its applications such as Second Language Learning. However, even for English with its numerous resources and wide research community, modern models are far from being perfect. In particular, the recall of the state-of-the-art GECToR model (Omelianchuk et al., 2020) on standard BEA2019 development set is lower than 40%. While GECToR uses sequence labeling approach with linguistically motivated label inventory, most works for other languages, such as (Náplava and Straka, 2019) and others, simply mimic machine translation methods, training a Transformer model (Vaswani et al., 2017) on the pairs of source and corrected sentences. This approach suffers from left-to-right decoding: the model can make a wrong decision not observing the future context.

This problem may be mitigated using reranking: the ranker observes entire corrected sequences and thus may utilize richer context. It also helps to discriminate between several possible edits with similar basic model probability . Due to these reasons, it was heavily used in machine translation both in statistical (Och et al., 2004) and neural (Yee et al., 2019) era.

In contrast to machine translation, sequence editing in GEC usually does not require complete rewriting and can be decomposed to elementary edits such as modifying a single word or a consecutive group of words. In this paper we propose to score elementary edits produced by the basic model and classify them as positive or negative on the second stage of the pipeline. Than the calculated probabilities can be either used directly or combined with the scores from the first stage.

We show that our scoring model achieves state-of-the-art performance on BEA2019 dataset even with rather weak first stage model. Its combination with GECToR edit generator outperforms the models of the same size by about 2 $F_{0.5}$ points. The large version of our model establishes the new SOTA on BEA dataset among models of all size. We also improve over current SOTA on Russian RULEC-GEC dataset by more than 3 $F_{0.5}$ points. We make most of our code and models freely available[1].

2 Pipeline description

As proposed by Alikaniotis and Raheja (2019), probably the simplest approach to grammatical error correction is to generate possible edits using a rule-based model and then extract those that increase the sentence probability by a sufficient margin. This probability may be estimated using a large pretrained language model, such as GPT (Radford et al., 2019) or BERT (Devlin et al., 2019). This approach requires no training data, only a development set for tuning the hyperparameters. As

[1] https://github.com/AlexeySorokin/EditScorer

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 11416 - 11429

a reverse side of its simplicity, this algorithm has two main limitations:

- Recall is limited to errors that can be specified by the rules.
- The probability estimators are imperfect, especially when the edit changes sequence length.

Therefore the main idea of our paper is to replace the scorer by a more powerful trainable model. Another key detail is that we apply the scorer not to the full corrections, but to the elementary edits. Namely, given the erroneous sentence **The boy fall on floor yesterday* and its correction *The boy fell on the floor yesterday*, our model should return **True** for sentences *The boy fell on floor yesterday* and *The boy fall on the floor yesterday* and **False** for other elementary corrections, for example, **The boy falls on floor yesterday*.

So, our model includes three main stages:

1. Extracting elementary edits from the edit generator.
2. Classifying these edits as positive or negative.
3. Applying the positively classified edits to the source sentence.

The first part is described in 3 and the remaining two in Section 4. A schematic description of our algorithm is given on Figure 1.

3 Edit generators

In this section we describe the first stage of our pipeline – the edit generator. We seek to make our pipeline independent of particular generator selection. Therefore we describe three possible variants: the rule-based generator (Subsection 3.1), the seq2seq model (3.2) and the sequence labeling one (3.3), based on the well-known GECToR model (Omelianchuk et al., 2020). Note that GECToR is available only for English and its development for languages with complex morphology is problematic since it needs a word inflection module to transform the predicted labels into surface forms.

3.1 Rule-based edit generator

We start with a rule-based edit generator, combined with a left-to-right neural language model. It may be viewed as our reimplementation of Alikaniotis and Raheja (2019). Our edit generation module takes as input a dependency tree of the sentence and applies rule-based edits corresponding to the most frequent errors, such as missing or incorrect determiners, commas and prepositions or wrong choice of word form. The exact list of applied rules is given in Appendix A.1.

These operations produce a fairly large number of possible corrections. To reduce computational burden we apply two-stage filtering based on left-to-right probability model p, such as GPT (Radford et al., 2019). First, for every hypothesis $\mathbf{u}$ we calculate the gain $\log p(u_{\pi+1}|w_1...w_\pi) - \log p(w_{\pi+1}|w_1...w_\pi)$, where π is the length of longest common prefix of $\mathbf{u}$ and source sequence $\mathbf{w}$.[2] We choose best K_{del} deletions, K_{ins} insertions and K_{sub} replacement edits according to this score. For the selected hypotheses we calculate their full log-probability and pick K best variants provided their score exceeds $\log p(\mathbf{w}) - \theta$, where θ is the predefined margin. [3]

3.2 Sequence-to-sequence edit generator

To generate edits using a sequence-to-sequence basic model we run standard beam search, align all the produced hypotheses with the source sentence and extract non-trivial parts of such alignments. The score of edit $\mathbf{e}$ equals $\log p(\mathbf{u}|\mathbf{w}) - \log p(\mathbf{v}|\mathbf{w})$, where $\mathbf{u}$ denotes the most probable hypothesis containing $\mathbf{e}$ and $\mathbf{v}$ is the most probable hypothesis that changes nothing in the span of $\mathbf{e}$. If there is no such hypothesis, we set the score to $\log p(\mathbf{u}|\mathbf{w}) - \log p(\mathbf{v}|\mathbf{w}) + 1$, where $\mathbf{v}$ is the last hypothesis in the beam.[4] We experimented with tracking only hypotheses with at most one edit, however, it requires implementing an additional control mechanism over beam states and does not bring performance gains. The same holds for diverse beam search, which also has additional hyperparameters such as diversity penalty.

This approach can be applied to any sequence-to-sequence model, in our paper we use BERT-GEC(Kaneko et al., 2020) as basic edit generator.

3.3 Sequence labeling generator

In contrast to other methods, the recent GECToR model (Omelianchuk et al., 2020) reduces grammar error correction to sequence tagging. We give an example of such reduction in Table 1 and refer the reader to Sections 3 and 5 of the original paper to

[2] It requires one pass of the pretrained GPT-like LM.

[3] We set $K_{del} = 10, K_{ins} = 10, K_{sub} = 30, K = 15, \theta = 3.0$.

[4] It is equivalent to assumption that 'no_change' hypothesis is one point worse than the last beam element.

Source	Edit generator	Model score	Stage 1	Stage 2	Stage 3	Target
The boy fall on floor yesterday	$(0, 1, \textit{boys})$	0.53	?	?	×	*The boy fell on the floor yesterday*
	$(1, 2, \textit{falls})$	0.7	?	×	-	
	$(1, 2, \textit{fell})$	0.83	?	✓	+	
	$(3, 3, \textit{the})$	0.9	✓	+	+	
	$(-1, -1, \text{None})$	0.57	?	?	✓ (stop)	

Figure 1: The pipeline of our algorithm. On each decoding stage, the most probable (labeled with check) remaining action is selected. It also eliminates other edits with intersecting spans (labeled with cross). In the end all the selected operations (labeled with plus) are applied in parallel.

better understand their approach. GECToR operations naturally correspond to elementary edits in our terminology. For each position i we extract all the tags $\mathbf{t}$ such that

$$\log p(t_i = \mathbf{t}) \geq \log p(t_i = \text{KEEP}) - \theta,$$

where p is the label probability GECTOR returns and θ is the predefined margin. For example, if on the first step of the example in Table 1 we have $p(t_3 = \text{VBD}) = 0.5$, $p(t_3 = \text{VBZ}) = 0.3$, $p(t_3 = \text{KEEP}) = 0.1$, then the VBZ transformation *fall* $\rightarrow$ *falls* will also be extracted. Again, we keep top K edits according to the difference between logarithmic probabilities of the edit and the the default "do nothing" operation (the KEEP tag).

3.4 Common details

Existing Grammatical Error Correction datasets may contain 'k-to-l' edits with either $k > 1$ or $l > 1$. Since datasets differ in how they treat such multiword operations and GECToR model produces only 0-to-1, 1-to-1 and 1-to-0 edits, we split all such edits to elementary '0-to-1', '1-to-0' and '1-to-1' operations, treating as correct all elements of such splits. We also add the "do nothing" edit that returns the source sentence. It is treated as positive if the original sentence is already correct.

4 Model description

4.1 Edits classification

Given numerous successes of Transformer models in NLP, we decide to use Roberta (Liu et al., 2019) for edit classification. It takes as input the sequence

$$\mathbf{x} = \langle\text{BOS}\rangle\text{SOURCE}\langle\text{SEP}\rangle\text{EDITED\_SOURCE}\langle\text{EOS}\rangle$$

and outputs the probability of the edited source to be a plausible correction. Consider the sequence $\mathbf{x} = \text{BOS}\ x_1 \dots x_L\ \text{SEP} x'_1 \dots x'_{L+\delta}\ \text{EOS}$ and let $x_i \dots x_j$ and $x'_i \dots x'_{j+\delta}$ be the source and the target of the edit, respectively. Then our classification model M can be decomposed as

$$M(\mathbf{x}) = g(f(\text{READOUT}(\text{ENCODER}(\mathbf{x})))),$$

where

- ENCODER is the Transformer encoder that produces the embeddings[5] sequence $\mathbf{h} = h_{\text{BOS}} h_1 \dots h_L h_{\text{SEP}} h'_1 \dots h'_{L+\delta} h_{\text{EOS}}$.
- READOUT is the readout function that converts a sequence of embeddings to the vectorization of the whole input. We use the first embedding of the target span (x'_i in our notation) and consider other variants during ablation in Appendix D.1.
- f is a multilayer perceptron and g is the final classification layer with sigmoid activation.

4.2 Decoding

After classifying the edits we cannot simply apply all 'positive' operations as they may conflict each other (e.g., the edits *fall* $\rightarrow$ *fell* and *fall* $\rightarrow$ *falls* for the sentence *The boy fall on the floor yesterday*). The conflicts may also happen between adjacent edits (*boy* $\rightarrow$ *boys* and *fall* $\rightarrow$ *falls*) thus we consider as contradicting any two edits whose source spans either intersect or are adjacent and non-empty.[6] We test two decoding strategies:

parallel pick the edits whose probability is greater than the maximum of predefined threshold and "do nothing" edit score. Keep those that do not contradict any edits with higher scores.

stagewise if the most probable edit is "do nothing" or its probability is below threshold, stop. Otherwise pick the most probable edit, apply it to the current input sentence and remove all the edits with intersecting spans. Repeat this until reaching the maximal number of iterations.

[5]Through all the paper 'embedding' means the encoder output for current subtoken.

[6]Using all non-overlapping edits leads to worse empirical performance and is less correct linguistically.

Iter.	Source	Edits	Result
1	CLS Boy fall the floor	APPEND_The LOWER VBD KEEP KEEP	The boy fell the floor
2	CLS The boy fell the floor	KEEP KEEP KEEP APPEND_on KEEP KEEP	The boy fell on the floor

Table 1: An example of GECToR labeling and corresponding sentence edits.

S In there moment , I thought that my best friends was my parents and sister .
A 0 1|||R:PREP|||At|||REQUIRED|||-NONE-|||0
A 1 2|||R:OTHER|||that|||REQUIRED|||-NONE-|||0
A 10 11|||R:VERB:SVA|||were|||REQUIRED|||-NONE-|||0
S При новых законах , надо было держать женщин на работу .
A 0 1|||Предлог|||По|||REQUIRED|||-NONE-|||0
A 1 3|||Заменить|||новым законам|||REQUIRED|||-NONE-|||0
A 9 10|||Сущ.:Падеж|||работе|||REQUIRED|||-NONE-|||0

Figure 2: Examples of single sentence descriptions for English (above) and Russian (below). The second edit for Russian sentence ("новых законах" *novykh zakonakh* 'new+PL+LOC law+PL+LOC' $\mapsto$ "новым законам" *novym zakonam* 'new+PL+DAT law+PL+DAT') is multiword and should be split into two separate substitutions during edit generation.

The stagewise strategy is slower as it requires rerunning the scorer on the modified sentence on each iteration. However, it produces slightly better scores, thus we use it for all experiments in the paper. The optimal threshold is model-dependent and is optimized on development set. We investigate the effect of threshold selection and decoding strategy in Appendix D.2.

4.3 Scoring

All edit generators described above also return scores, corresponding to edit log-probabilities. By default we do not use them, taking only the scorer probabilities (the '**scorer-only**' variant). We also try to combine generator and scorer probabilities in a log-linear model (the '**combined**' method). Precisely, we set the score of edit $\mathbf{e}$ equal to $\log p_{\text{scorer}}(\mathbf{e}) + \alpha \cdot \text{score}_{\text{gen}}(\mathbf{e})$, where α is the tunable parameter.[7]

The 'scorer-only' variant is used by default for most experiments in the paper, the 'combined' method scores are reported only for the best selected models to compare with SOTA scores.

5 Data and models

5.1 Data

We test our approach on English (a high-resource language) and Russian with less resources and worse edit generators available. For English we use the BEA 2019 Shared Task data (Bryant et al., 2019). We use the same training data as in the previous works: Write&Improve and LOCNESS corpus (Bryant et al., 2019), First Certificate of English (FCE) (Yannakoudakis et al., 2011), National University of Singapore Corpus of Learner English (NUCLE) (Dahlmeier et al., 2013), Lang-8 Corpus of Learner English (Tajiri et al., 2012) and synthetic data (Awasthi et al., 2019). For experiments with pretraining on synthetic data we utilize PIE dataset (Awasthi et al., 2019). We test our models on BEA 2019 development and test sets and CoNLL 2014 (Ng et al., 2014) test data.

For additional experiments we also use cLang8 (Rothe et al., 2021) – the cleaned and extended version of Lang8 corpus. The characteristics of datasets are given in Table 2.

Dataset	Size	Usage
W&I+LOCNESS	34308	Train, finetune
FCE	28350	Train
NUCLE	57151	Train
Lang8	1037561	Train
PIE synthetic	9000000	Pretrain
BEA 2019 dev	4384	Development
BEA 2019 test	4477	Test
CoNLL14	1312	Test
cLang8	2372119	Train

Table 2: Training data for English GEC experiments.

For Russian we use the RULEC-GEC data (Rozovskaya and Roth, 2019). Due to its small size we generate our own synthetic dataset, cor-

[7] In all the experiments the optimal value was $\alpha = 0.1$.

rupting the source sentences with rule-based operations such as comma / preposition insertion/deletion/replacement or changing the word to another form of the same lexeme. The full list of operations is given in Appendix A.2.

Dataset	Sentences	Errors
RULEC-GEC train	4980	4383
RULEC-GEC dev	2500	2182
RULEC-GEC test	5000	5301
Synthetic data	213965	187122

Table 3: Data used for experiments on Russian GEC.

We follow the training procedure described in (Omelianchuk et al., 2020). Namely, after pretraining on synthetic data only we perform the main training on full BEA 2019 train set which is the concatenation of W&I+LOCNESS, FCE, NUCLE and Lang8 and afterwards finetune the model on W&I+LOCNESS. When using cLang8 instead of Lang8 we do not apply pretraining. For Russian we pretrain on the concatenation of real and synthetic data and finetune on RULEC-GEC train set.

5.2 Model architecture and training

For our scorer we use the Transformer model and initialize it using the weights of pretrained Roberta(Liu et al., 2019)[8] model. We represent the edit with the embedding of the leftmost word in the target span. This vector is then passed through a two-layer perceptron with intermediate ReLU and sigmoid output activation. We implement our models using PyTorch and use HuggingFace to work with pretrained transformer models.

The model is trained using total batch size of 3500 subtokens to fit into 32GB GPU memory. All the examples for a single sentence are placed into the same batch. Since the number of proposed negative edits is much larger than the number of positive ones, we independently average the binary cross-entropy loss for positive and negative examples inside each batch. We optimize the model with AdamW optimizer using default hyperparameters.

6 Experiments

In this section we describe our experiments. Note that our main contribution is the scorer and we claim that our method is not limited to a particular edit generator. Thus we do not train edit generators by ourselves and only adapt them to our pipeline as described in Section 3.

[8]By default we use the roberta-base variant.

Dataset	Rule-based	BERT-GEC	GECToR
BEA dev	45.8	55.5	54.9
W&I train	46.7	61.0	66.3
FCE	40.4	60.7	56.6
NUCLE	39.6	48.3	45.0
Lang8	33.0	50.2	43.3
BEA dev $F_{0.5}$	38.4	48.6	54.1

Table 4: Recall of different edit extraction methods for English. W&I is W&I+LOCNESS.

Our main experiments are conducted for English, in Subsection 6.3 we also present results for Russian. We compare the models by $F_{0.5}$ score using ERRANT (Bryant et al., 2019) for English BEA data and M2Scorer (Dahlmeier et al., 2013) for other datasets. Since the BEA 2019 test set does not contain correct answers, we do most of the comparisons on the development data.

6.1 Edit generators

We use three edit generators of different type: the rule-based one with GPT2-medium edit scorer (Subsection 3.1), the seq2seq BERT-GEC model[9] (Subsection 3.2) and the sequence labeler based on our extension of GECToR[10] (Subsection 3.3). For all edit generators we set the number of hypotheses ("beam width") to 15 and loss threshold θ to 3.0.

Before all we check that our edit generator has sufficient recall. As shown in Table 4, BERT-GEC and GECToR has similar recall on BEA data, while on other datasets BERT-GEC coverage is better despite lower quality of the corresponding model. Recall of the rule-based model is low because it cannot handle free text rewriting.

6.2 English

The first research question we ask is **RQ1: does our scorer achieve decent performance for all edit generators**. We answer it in the upper block of Table 5. For all three scorers our model outperforms the BERT-GEC model(Kaneko et al., 2020) on BEA2019 development set. It is still behind the state-of-the-art GECToR model, however, the latter

[9]https://github.com/kanekomasahiro/bert-gec

[10]We use the roberta-base model available from https://github.com/grammarly/gector, our extension code is available on https://github.com/AlexeySorokin/gector

Edit gen.	Scorer	PT	BEA 2019 dev			CoNLL 2014		
			P	R	$F_{0.5}$	P	R	$F_{0.5}$
Rule-based	'scorer-only'	NO	63.3	28.1	50.6	71.2	33.3	58.0
BERT-GEC	'scorer-only'	NO	62.1	33.9	53.2	70.2	38.0	60.0
GECToR	'scorer-only'	NO	60.4	34.1	52.5	73.6	34.9	60.2
BERT-GEC	'scorer-only'	YES	68.4	30.4	55.1	71.2	39.4	61.3
GECToR	'scorer-only'	YES	69.1	30.9	55.4	72.9	39.1	62.1
GECToR	'combined'	YES	68.4	34.5	57.2	79.1	38.3	65.2
BERT-GEC(Kaneko et al., 2020)		YES	53.0	36.5	48.6	69.2	45.1	62.5
GECToR, roberta(Omelianchuk et al., 2020)		YES	62.3	35.6	54.2	72.8	40.9	63.0
GECToR, XLNet(Omelianchuk et al., 2020)		YES	66.0	33.8	55.5	77.5	40.2	65.3

Table 5: Results for different GEC models on two English GEC datasets. Models in the second block are additionally pretrained on 9M synthetic data. Lower blocks contains results of external models. PT='YES' means additional PIE pretraining.

uses more training data being pretrained on PIE synthetic dataset.

Consequently, we ask the **RQ2: does our model outperform GECToR(Omelianchuk et al., 2020) when being trained in the same conditions**. The second block of Table 5 shows that our ranker outperforms GECToR variant based on the same Transformer with both model-based edit generators. If it takes into account the scores of GECToR edit generator by using 'combined' decoding, we additionally improve on BEA dev by 1.8 $F_{0.5}$ points. Notably, if BERT-GEC is used on the first stage of our pipeline, the scorer still shows solid performance being significantly better than its generator model. Thus SOTA performance is possible even for a weak generator model provided its recall is high.

The third research question is **RQ3: can our model further improve performance using larger language models and more training data**. In this setup we do two modifications: replace Lang8 with larger and better cLang8 dataset (Rothe et al., 2021) and utilize roberta-large model instead of roberta-base. For all the models we use GECToR edit generator. As shown in Table 6, roberta-large produces further improvement over roberta-base and outperforms current SOTA on BEA 2019 test set. However, the improvement on CoNLL-2014 is much smaller, we hypothesize that our models may overfit to BEA domain.

6.3 Russian

English is relatively simple from the point of its morphology. With 6 main cases and 3 genders, Russian is a more complicated case. Its rich nominal morphology drastically extends the space of possible errors even for the rule-based generator. There is no pretrained model for Russian GEC, thus we compare two generators: the rule-based one (analogous to English) and the finetuned ruGPT-large.[11] Their coverage statistics are given in Table 7. We initialize the scorer with ruRoberta-large[12] since there is no roberta-base for Russian. The results are given in Table 8.

Contrasting with English experiments, even scoring the rule-based edits outperforms the previous SOTA models. We also note the latter are of larger size and were trained with two magnitudes more synthetic data. When using the model-based generator, the score is 1.5 better; the combination of ranker and generator scores yields further $F_{0.5}$ improvement. Thus we answer positively to **RQ4: is the suggested approach applicable in case of little annotated data and languages other than English**.

7 Ablation studies

7.1 Joint generators

Our model is trained on edits from a particular generator. A natural question is whether it overfits to this generator or learns a model-independent notion of grammaticality. We check this by training a model with a single generator and applying it to the union of different generators output ('joint' generator in Table 9). We also investigate the effect of finetuning and full training on joint edit sets.

[11] https://huggingface.co/sberbank-ai/ruGPT3large_based_on_gpt2

[12] https://huggingface.co/sberbank-ai/ruRoberta-large

Model	Scorer	cLang8	BEA 2019 dev			BEA 2019 test			CoNLL 2014		
			P	R	$F_{0.5}$	P	R	$F_{0.5}$	P	R	$F_{0.5}$
roberta-base	‘combined‘	NO	68.4	34.5	57.2	82.4	54.5	74.7	79.1	38.3	65.2
roberta-base	‘scorer-only‘	YES	70.2	32.9	57.2	82.8	52.4	74.2	72.6	39.5	63.9
roberta-base	‘combined‘	YES	69.3	35.5	58.2	82.5	55.1	75.1	79.6	36.2	66.0
roberta-large♢	‘scorer-only‘	NO	70.2	33.1	57.3	83.8	52.0	74.7	77.3	36.3	63.0
roberta-large♢	‘combined‘	NO	69.6	35.6	58.5	83.5	54.4	75.5	79.3	39.5	66.0
roberta-large♣♢	‘scorer-only‘	YES	**71.0**	33.4	57.9	**86.2**	54.2	**77.1**	79.4	36.1	64.0
roberta-large♣♢	‘combined‘	YES	70.3	**35.9**	**59.0**	84.8	**56.3**	77.0	80.2	39.1	66.3
GECToR, ensemble		NO	NA	NA	NA	79.4	57.2	73.7	78.2	41.5	66.5
(Sun et al., 2021)♣♢		NO	NA	NA	NA	NA	NA	NA	71.0	52.8	66.4
T5-XXL, cLang8♣♢		YES	NA	NA	NA	NA	NA	75.9	NA	NA	**68.9**

Table 6: Results for different GEC models on different GEC datasets. Lower blocks contains results of external models. cLang8 column means whether the model was trained on cLang8 dataset, ♣ stands for large language models and ♢ for using additional training data.

Dataset	Coverage	
	Rule-based	ruGPT-based
RULEC-GEC train	54.4	81.5
RULEC-GEC dev	55.5	59.3
RULEC-GEC test	46.4	54.3
Synthetic data	78.0	95.8

Table 7: Coverage of edit generators for Russian data.

Results are given in Table 9. We note that the recall of joint generator on BEA development set is 69%, which significantly exceeds the coverage of individual generators, which is about 55% (see Table 4). Table 10 also illustrates the difference in edits produced by different generators.

Joining generators output produce minor improvements for GECToR-based model and has negative impact on BERT-GEC-based one. It proves that our models overfit to the edit generation algorithm, the most severe overfitting happens in case of BERT-GEC. As expected, full training on joint set of edits performs better than only on edits from GECToR generator. The same patterns hold for large models and ‘combined‘ decoding, in particular, the roberta-large model trained with joint edits achieves 76.2 $F_{0.5}$ on BEA test, reaching the highest score among the models trained without external data.

7.2 Decoding ablation

Our decoding algorithm has three hyperparameters: the decoding algorithm (‘scorer-only‘ or ‘combined‘), the threshold between positive and negative edits and the the maximal allowed number of edits. Detailed results of their ablation are in Appendix D.2, summarizing:

1. ‘Combined‘ decoding provides a stable improvement of $0.5 - 1\%$ over ‘scorer-only‘.
2. Optimal threshold is usually 0.7 before finetuning and 0.9 after finetuning.
3. $F_{0.5}$ score monotonically improves up to 8 allowed edits due to increased recall, after 5 edits the scores almost saturate.

8 Related work

The task of grammatical error correction has a long history. The main paradigm of recent years is to treat it as low-resource machine translation (Felice et al., 2014; Junczys-Dowmunt et al., 2018) using extensive pretraining on synthetic data (Grundkiewicz et al., 2019). Synthetic data is usually generated using random replacement, deletion, insertion, spelling errors and perturbations (Grundkiewicz et al., 2019; Kiyono et al., 2019; Náplava and Straka, 2019), other approaches include training on Wikipedia edits (Lichtarge et al., 2019) and backtranslation (Kiyono et al., 2019). Another trend is incorporating pretrained Transformer language models either as a part of system architecture (Kaneko et al., 2020) or for the initialization of model weights (Omelianchuk et al., 2020). The extreme case of the latter approach is the “brute force” when one simply uses large encoder-decoder Transformer that potentially is able to solve any text-to-text task (Rothe et al., 2021).

Another paradigm in GEC is to reduce grammar correction to sequence labeling (Omelianchuk

Model	Training data	P	R	$F_{0.5}$
Transformer (Náplava and Straka, 2019)	10M synth., RULEC-GEC train, dev	63.3	27.5	50.2
mT5-XXL (Rothe et al., 2021)	mC4 synth., RULEC-GEC train	NA	NA	51.6
ruGPT-large finetune (strong baseline)	200K synth., RULEC-GEC train	65.7	27.4	51.3
rule-based edits	200K synth., RULEC-GEC train	69.4	25.9	51.9
ruGPT-large edits, 'scorer-only'	200K synth., RULEC-GEC train	70.9	26.8	53.4
ruGPT-large edits, 'combined'	200K synth., RULEC-GEC train	73.7	27.3	55.0

Table 8: Results for Russian on RULEC-GEC data. Our results are in the lower block and baselines are in the upper.

Generator			Metrics		
Train	Finetune	Test	P	R	$F_{0.5}$
GECToR	GECToR	GECToR	69.1	30.9	55.4
	GECToR	joint	67.6	33.0	55.9(+0.5)
	joint	joint	64.8	35.5	55.7(+0.3)
BERT-GEC	BERT-GEC	BERT-GEC	68.4	30.4	55.1
	BERT-GEC	joint	63.4	34.2	54.2(−0.9)
	joint	joint	64.2	34.3	54.6(−0.5)
joint	joint	joint	64.5	38.2	56.7(+1.3)

Table 9: Effect of generator joining on different training stages. All models are trained on full BEA dataset with PIE pretraining and tested on BEA development set using 'scorer-only' decoding.

et al., 2020). However, it requires constructing a linguistically meaningful set of tags that could be hard to design for languages with complex morphology. Our work mainly follows the third approach that considers GEC as two-stage process including edit generation as the first stage and their ranking or classification as the second. Edits were usually generated by manually written rules and their scoring was performed by linear classifiers (Rozovskaya et al., 2014) or later by a pretrained language model (Alikaniotis and Raheja, 2019). A recent work of Yasunaga et al. (2021) generates edits using separate sequence-to-sequence Transformer and then filters them using a language model.

Our approach can be seen as a special case of **reranking**. Feature-based reranking was common in statistical machine translation before the advent of neural networks (Och et al., 2004), in the field of grammatical error correction it was applied by Hoang et al. (2016), Xie et al. (2016) used a feature-based binary classifier similar to ours to improve precision of the GEC model. Grundkiewicz et al. (2019) used a R2L language model scorer to rerank the output of the first stage seq2seq model. However, recent studies on machine translation (Lee et al., 2021) and summarization (Liu and Liu, 2021) benefit from a training a Transformer rescoring model, not choosing a fixed one. Our work is partially inspired by theirs, the key difference is that we use classification loss instead of ranking and rerank individual edits, not complete sentences. As far as we know, the only example of trainable reranking for GEC is Liu et al. (2021), but it uses a more complex architecture and focuses more on error detection than correction.

9 Conclusion

We have developed a two-stage algorithm for grammatical error correction based on edit classifications. Our main results are the following:

- Our model reaches state-of-the-art performance on English even without using the scores of edit generator. With 'roberta-base' backbone, it outperforms models of the same size and achieves SOTA scores using 'roberta-large'.
- It notably improves current state-of-the-art on Russian, proving that our model is also applicable to small datasets with weaker edit generators.
- Our approach works with different edit generators and their combinations.

Since our model shows competitive performance even with rule-based edit generators, it may be applied in settings that require control over possible corrections. One such field is language learning,

e.g., correcting error of particular type, such as verb tense or determiner choice. In the future work we plan to address this question in more details and test the applicability of our approach on additional languages, such as German or Czech. Last but not the least, the main idea of ranking individual edits can be applied not only to GEC, but to any task where the concept of elementary edit has meaning, for example, machine translation post-editing.

References

Dimitris Alikaniotis and Vipul Raheja. 2019. The unreasonable effectiveness of transformer language models in grammatical error correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 127–133, Florence, Italy. Association for Computational Linguistics.

Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, and Vihari Piratla. 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)*, pages 4260–4270, Hong Kong, China. Association for Computational Linguistics.

Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe. 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*, pages 52–75, Florence, Italy. Association for Computational Linguistics.

Daniel Dahlmeier, 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*, pages 22–31, Atlanta, Georgia. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Mariano Felice, Zheng Yuan, Øistein E. Andersen, Helen Yannakoudakis, and Ekaterina Kochmar. 2014. Grammatical error correction using hybrid systems and type filtering. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 15–24, Baltimore, Maryland. Association for Computational Linguistics.

Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Kenneth Heafield. 2019. Neural grammatical error correction systems with unsupervised pre-training on synthetic data. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 252–263, Florence, Italy. Association for Computational Linguistics.

Duc Tam Hoang, Shamil Chollampatt, and Hwee Tou Ng. 2016. Exploiting n-best hypotheses to improve an smt approach to grammatical error correction. *arXiv preprint arXiv:1606.00210*.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Shubha Guha, and Kenneth Heafield. 2018. Approaching neural grammatical error correction as a low-resource machine translation task. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 595–606, New Orleans, Louisiana. Association for Computational Linguistics.

Masahiro Kaneko, Masato Mita, Shun Kiyono, Jun Suzuki, and Kentaro Inui. 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*, pages 4248–4254, Online. Association for Computational Linguistics.

Shun Kiyono, Jun Suzuki, Masato Mita, Tomoya Mizumoto, and Kentaro Inui. 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)*, pages 1236–1242, Hong Kong, China. Association for Computational Linguistics.

Ann Lee, Michael Auli, and Marc'Aurelio Ranzato. 2021. Discriminative reranking for neural machine translation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7250–7264, Online. Association for Computational Linguistics.

Jared Lichtarge, Chris Alberti, Shankar Kumar, Noam Shazeer, Niki Parmar, and Simon Tong. 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)*, pages 3291–3301, Minneapolis, Minnesota. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis,

Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Yixin Liu and Pengfei Liu. 2021. SimCLS: A simple framework for contrastive learning of abstractive summarization. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 1065–1072, Online. Association for Computational Linguistics.

Zhenghao Liu, Xiaoyuan Yi, Maosong Sun, Liner Yang, and Tat-Seng Chua. 2021. Neural quality estimation with multiple hypotheses for grammatical error correction. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5441–5452, Online. Association for Computational Linguistics.

Jakub Náplava and Milan Straka. 2019. Grammatical error correction in low-resource scenarios. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 346–356, Hong Kong, China. Association for Computational Linguistics.

Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. 2014. The CoNLL-2014 shared task on grammatical error correction. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–14, Baltimore, Maryland. Association for Computational Linguistics.

Franz Josef Och, Daniel Gildea, Sanjeev Khudanpur, Anoop Sarkar, Kenji Yamada, Alex Fraser, Shankar Kumar, Libin Shen, David Smith, Katherine Eng, Viren Jain, Zhen Jin, and Dragomir Radev. 2004. A smorgasbord of features for statistical machine translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004*, pages 161–168, Boston, Massachusetts, USA. Association for Computational Linguistics.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. GECToR – grammatical error correction: Tag, not rewrite. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 163–170, Seattle, WA, USA → Online. Association for Computational Linguistics.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners.

Sascha Rothe, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. 2021. A simple recipe for multilingual grammatical error correction. *arXiv preprint arXiv:2106.03830*.

Alla Rozovskaya, Kai-Wei Chang, Mark Sammons, Dan Roth, and Nizar Habash. 2014. The Illinois-Columbia system in the CoNLL-2014 shared task. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 34–42, Baltimore, Maryland. Association for Computational Linguistics.

Alla Rozovskaya and Dan Roth. 2019. Grammar error correction in morphologically rich languages: The case of Russian. *Transactions of the Association for Computational Linguistics*, 7:1–17.

Xin Sun, Tao Ge, Furu Wei, and Houfeng Wang. 2021. Instantaneous grammatical error correction with shallow aggressive decoding. *arXiv preprint arXiv:2106.04970*.

Toshikazu Tajiri, 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)*, pages 198–202, Jeju Island, Korea. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Ziang Xie, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, and Andrew Y Ng. 2016. Neural language correction with character-based attention. *arXiv preprint arXiv:1603.09727*.

Helen Yannakoudakis, 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*, pages 180–189, Portland, Oregon, USA. Association for Computational Linguistics.

Michihiro Yasunaga, Jure Leskovec, and Percy Liang. 2021. Lm-critic: Language models for unsupervised grammatical error correction. *arXiv preprint arXiv:2109.06822*.

Kyra Yee, Yann Dauphin, and Michael Auli. 2019. Simple and effective noisy channel modeling for neural machine translation. 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)*, pages 5696–5701, Hong Kong, China. Association for Computational Linguistics.

A Rule-based transformations used for edit generation

A.1 English

Rule-based edit generator includes the following operations:

- Comma insertion and deletion.
- Preposition insertion, deletion and substitution. Insertion is allowed only before the first token of a noun group.
- Determiner insertion, deletion and substitution. Insertion is allowed only before the first token of a noun group.
- *to* insertion before infinitives.
- Spelling correction for OOV words using Hunspell.[13]
- Substitution a word with all its inflected forms, inflection is performed using Lemminflect.[14]
- Capitalization switching.
- Replacement of comma by period and capitalizing the subsequent word (*I have a dog, it is cute.* → *I have a dog. It is cute.*).

A.2 Russian

Rule-based edit generator for Russian includes the following operations:

- Comma insertion and deletion.
- Preposition insertion, deletion and substitution. Insertion is allowed only before the first token of a noun group.
- Conjunction substitution.
- Spelling correction for OOV words using Hunspell.[15]
- Joining of consecutive words using Hunspell (e.g. *ne bol'shoj* 'no+big' ↦ *nebol'shoj* 'small').
- Substitution a word with all its inflected forms, inflection is performed using PyMorphy.[16]
- Joint noun group inflection (e.g. *bol'shoj dom* 'large house' ↦ *bol'shikh domov* 'large+GEN+PL houses+GEN')
- Capitalization switching.
- Switching the order of consecutive words.

The rules take as input sentence dependency trees, parsing is done using DeepPavlov.[17]

[13] https://github.com/MSeal/cython_hunspell

[14] https://github.com/bjascob/LemmInflect/

[15] https://github.com/MSeal/cython_hunspell

[16] https://github.com/kmike/pymorphy2/

[17] http://docs.deeppavlov.ai/en/0.14.1/

B Data sources

English

- W&I-LOCNESS train, dev and test.
- FCE.
- Lang8.
- CLang8.
- CoNLL14.
- PIE synthetic data.

Russian

- RULEC-GEC.
- Synthetic data: not available yet.

C Examples of elementary edits

See Table 10.

D Ablation studies

D.1 Additional losses

The choice of model architecture and training parameters may seem arbitrary. Therefore in this section we study other possible variants of modern architecture. The architecture used in main experiments has the following key components:

1. The model is trained with cross-entropy classification loss without any additional objectives.
2. The loss is normalized separately for positive and negative instances.
3. Edit is represented using the first token embedding in the output span.
4. The classification module contains a single hidden layer.
5. Except for the classification module, no additional layers are added on the top of main Transformer encoder.

We test the following architecture modifications:

1. Adding an additional ranking objective. We do it adding standard margin loss:

$$L(x^+, x^-) = \max\left(g(x^-) - g(x^+) + \theta, 0\right),$$

$$L = L_{CE} + \alpha \frac{\sum\limits_{(x^+,x^-)\in P} L(x^+,x^-)}{|P|}.$$

Source Correct	*Until the dawn all of them go out , so they sacred until they find a refuge .* *By dawn all of them had got out , so they sacred until they found a refuge .*		
Edit	Target	Gain	Label
Rule-based edit generator			
(1, 2, the) → _	Until dawn all of them go out , so they sacred until they find a refuge .	1.33	True
(11, 11, _) → are	Until the dawn all of them go out , so they are sacred until they find a refuge .	0.95	False
(3, 3, _) → ,	Until the dawn , all of them go out , so they sacred until they find a refuge .	0.95	False
(11, 11, _) → were	Until the dawn all of them go out , so they were sacred until they find a refuge .	−1.73	False
(-1, -1, _) → _	Until the dawn all of them go out , so they sacred until they find a refuge .	0.00	False
BERT-GEC edit generator			
(11, 11, _) → are	Until the dawn all of them go out , so they are sacred until they find a refuge .	0.06	False
(1, 2, the) → _	Until dawn all of them go out , so they sacred until they find a refuge .	−0.06	True
(11, 11, _) → stay	Until the dawn all of them go out , so they stay sacred until they find a refuge .	−0.24	False
(0, 2, Until the) → Before	Before dawn all of them go out , so they sacred until they find a refuge .	−0.79	False
(12, 12, _) → themselves	Until the dawn all of them go out , so they sacred themselves until they find a refuge .	−2.95	False
(0, 2, Until the) → Up until	Up until the dawn all of them go out , so they sacred until they find a refuge .	−2.99	False
(-1, -1, _) → _	Until the dawn all of them go out , so they sacred until they find a refuge .	0.00	False
GECToR edit generator			
(0, 1, Until) → In	In the dawn all of them go out , so they sacred until they find a refuge .	5.35	False
(1, 2, the) → _	Until dawn all of them go out , so they sacred until they find a refuge .	4.59	True
(0, 1, Until) → _	The dawn all of them go out , so they sacred until they find a refuge .	4.01	False
(0, 1, Until) → As	As the dawn all of them go out , so they sacred until they find a refuge .	2.86	False
(12, 13, until) → _	Until the dawn all of them go out , so they sacred they find a refuge .	1.21	False
(15, 16, a) → _	Until the dawn all of them go out , so they sacred until they find refuge .	1.01	False
(7, 8, out) → _	Until the dawn all of them go , so they sacred until they find a refuge .	0.72	False
(0, 1, Until) → By	By the dawn all of them go out , so they sacred until they find a refuge .	0.71	True
(3, 3, _) → ,	Until the dawn , all of them go out , so they sacred until they find a refuge .	0.65	False
(8, 10, ,_so) → . So	Until the dawn all of them go out . So they sacred until they find a refuge .	0.48	False
(6, 7, go) → went	Until the dawn all of them went out , so they sacred until they find a refuge .	−0.55	False
(8, 9, ‘,‘) → _	Until the dawn all of them go out so they sacred until they find a refuge .	−0.81	False
(12, 12, _) → ,	Until the dawn all of them go out , so they sacred , until they find a refuge .	−1.18	False
(14, 15, find) → found	Until the dawn all of them go out , so they sacred until they found a refuge .	−3.76	True
(-1, -1, _) → _	Until the dawn all of them go out , so they sacred until they find a refuge .	0.00	False

Table 10: Output of different edit generators for the sentence *Until the dawn all of them go out , so they sacred until they find a refuge .* Gain column contains the first stage score.

Here g is the logit of positive class before sigmoid, P is the set of contrastive pairs of batch elements, θ is a margin hyperparameter and α is the additional loss weight [18]. We investigate 3 variants of defining P:

- All pairs of positive and negative instances (+*soft*),
- Only pairs of positive and negative instances whose spans intersect(+*hard*),
- All pairs of the form $(\mathbf{e}^+, \mathbf{e}_0)$ and $(\mathbf{e}^0, \mathbf{e}^-)$, where $\mathbf{e}^+, \mathbf{e}^-$ and $\mathbf{e}_0$ are positive, negative and "do nothing" edits, respectively(+*contrast*).

2. Removal of class normalization (*no_norm*).
3. Using the CLS token (*cls*), mean representation of output span (*mean*) and concatenation of output and source span (*origin*) as edit encodings.
4. Adding one more hidden layer in the classification block (*'2 layers'*).
5. Adding an additional Transformer layer between all the edit representations for the same sentence (+*attention*). That allows to potentially use information from other hypotheses.

We run all ablation experiments on the concatenation of W&I+LOCNESS train and FCE datasets using GECToR edit generator, results are given in Table 11. For all the models we select the best performing checkpoint and threshold according to the $F_{0.5}$ score and perform stagewise decoding. For those models that improve over the basic one on the small dataset, we run additional testing on full BEA train data without finetuning.

We observe that additional losses that are helpful in low-resource setting even decrease performance for larger data. Thus the variant used in the paper is the most effective despite being the simplest, however, a more detailed study is required.

D.2 Decoding ablation

In the first experiment in Table 12 we vary the decoding algorithm and the decision threshold. We

[18] We set $\alpha = 0.25, \theta = 2.0$.

Model	W&I+FCE			BEA 2019 train+finetune		
	P	R	$F_{0.5}$	P	R	$F_{0.5}$
Basic	55.5	26.7	46.1(+0.0)	60.4	34.1	52.5(+0.0)
+hard	55.1	26.4	45.8(−0.3)	NA	NA	NA
+soft	55.2	30.8	47.6(+1.5)	58.2	35.3	51.6(−0.9)
+contrast	55.1	31.1	47.7(+1.6)	60.9	30.1	50.5(−2.0)
no_norm	55.8	27.4	46.2(+0.1)	NA	NA	NA
CLS	57.7	22.0	43.5(−2.6)	NA	NA	NA
+mean	58.0	27.0	47.2(+1.1)	61.6	31.6	51.8(−0.7)
+origin	57.4	26.2	46.4(+0.3)	NA	NA	NA
2layers	55.6	27.7	46.3(+0.2)	NA	NA	NA
+attention	52.8	31.4	46.4(+0.3)	NA	NA	NA

Table 11: Comparison of different architecture modifications, the number in brackets is the difference with the 'Basic' model used in the paper. See the list above for a complete description.

provide the scores for the model trained with GECToR edit generation on full training data before and after finetuning on W&I-LOCNESS training data. Another notable pattern is that before finetuning the best $F_{0.5}$-score is achieved at threshold $0.6 - 0.7$, while afterwards the optimal threshold is $0.8 - 0.9$. These values are stable across datasets, so setting the threshold to 0.7 before finetuning and to 0.9 after it is nearly optimal, thus threshold tuning is almost unnecessary.

In Table 13 we also analyze how the quality of the model depends on the maximal number of edits allowed. We observe that recall and $F_{0.5}$ score are improved up to 8 edits per example. The difference between stagewise and parallel algorithms is about $0.5 - 0.7$ $F_{0.5}$ score. It follows the experience of (Omelianchuk et al., 2020), where iterative rewriting (the analogue of our stagewise decoding) improved performance even more significantly.

E Limitations and risks of the work

Our method relies on either the existence of a grammatical error correction model that can serve as model-based generator or a pretrained LM to be used with rule-based generator. With the existence of multilingual language models these requirements are fulfilled for most of high- or middle-resource languages. A more serious limitation is the existence of labeled corpus of grammatical errors and its quality.

Concerning practical applications of our work, we mentioned that it can be used for automatic correction of learner sentences, for example, in the field of Second Language Learning. However, we acknowledge that real-word learner errors differ from the ones in the academic datasets. It implies that before applying our model or its extension in any practical setting an additional study is required to check whether its precision is enough for practical usage. In particular, its corrections should be verified by a human in case of usage for automatic essay scoring and related tasks.

The model was trained on examples from academic datasets that may be biased towards students having particular mother tongue. Therefore an additional investigation is required, whether the model has equal quality for the sentences from English learners with different native languages and proficiency levels.

Threshold	Before finetuning			After finetuning		
	P	R	$F_{0.5}$	P	R	$F_{0.5}$
0.5	59.2	30.7	49.9	57.1	39.8	52.6
0.6	60.5	29.8	50.2	58.6	38.9	53.2
0.7	63.1	27.7	50.2	60.7	37.9	54.2
0.8	68.8	22.7	48.9	63.1	35.9	54.8
0.9	79.9	10.7	34.8	69.2	30.9	55.4

Table 12: Precision, recall and $F_{0.5}$ score on BEA 2019 development set with different decision thresholds with/without finetuning using parallel decoding. Models are trained on synthetic data and BEA 2019 full train set and finetuned on W&I-LOCNESS train set with GECToR edit generator.

		1	2	3	4	5	6	7	8
Parallel	Precision	72.9	70.6	69.6	69.5	69.4	69.4	69.4	69.4
	Recall	18.8	25.7	28.0	29.0	29.3	29.5	29.5	29.5
	$F_{0.5}$ score	46.2	52.4	53.7	54.3	54.5	54.6	54.6	54.6
Stagewise	Precision	72.9	71.0	70.1	69.4	69.2	69.1	69.0	69.0
	Recall	18.8	25.9	30.4	28.8	29.9	30.5	30.9	31.0
	$F_{0.5}$ score	46.2	52.6	54.5	54.9	55.2	55.3	55.4	55.4
	($F_{0.5}$ gain)	(+0.00)	(+0.2)	(+0.8)	(+0.6)	(+0.7)	(+0.7)	(+0.8)	(+0.8)

Table 13: Dependence of model performance from the maximal allowed number of edits. The last row is the difference between stagewise and parallel decoding algorithms.