Grammatical Error Correction with Denoising Autoencoder

Krzysztof Pajak[1]
LangMedia Sp. z o.o.,
Mariana Rapackiego 5, 53-021 Wrocław,
Poland

Adam Gonczarek[2]
Alphamoon Sp. z o.o.,
Grabarska 1, 50-079 Wrocław,
Poland

***Abstract*—A denoising autoencoder sequence-to-sequence model based on transformer architecture proved to be useful for underlying tasks such as summarization, machine translation, or question answering. This paper investigates the possibilities of using this model type for grammatical error correction and introduces a novel method of remark-based model checkpoint output combining. This approach was evaluated by the BEA 2019 shared task. It was able to achieve state-of-the-art F-score results on the test set 73.90 and development set 56.58. This was done without any GEC-specific pre-training, but only by fine-tuning the autoencoder model and combining checkpoint outputs. This proves that an efficient model solving GEC might be trained in a matter of hours using a single GPU.**

***Keywords*—Denoising autoencoder transformer; sequence-to-sequence; grammatical error correction; model ensembling; error remarks filtering; fine-tuning**

I. Introduction

Grammatical Error Correction (GEC) is a language processing task whose target is to detect and correct any mistake that could be found in input data, without changing the meaning intended by an author.

According to the British Council, English is spoken at a useful level by more than a quarter of the world's population [1]. Most of English users are not native speakers and posses different levels of proficiency. Therefore, all tools aimed for improving language correctness and assisting learning process would be of great importance.

There are two main approaches in solving Grammatical Error Correction task by neural models. First is to treat GEC as a form of Neural Machine Translation, where erroneous source texts are "translated" into correct ones (for example, [2] and [3]). The other way is to treat GEC as a sequence classification task, where model provides probability distribution over available corrections for every token ([4] and [5]).

From many approaches to create a GEC-solving system, so far the best results (BEA 2019 shared task [6] test set) have been reported by GECToR [4]. They propose a sequence tagging model that classifies input text tokens in a few iterations to identify the errors. They use pre-trained transformer-based encoders with dense layers on top that select one of possible token-level transformations. This architecture aids fast inference, since there is no need to sequentially decode output tokens as in NMT-like solutions. The model training was done in three phases, using a large amount of parallel synthetic data at first and then tuning on smaller higher quality sets (NUCLE, LANG8 [7], FCE, WI+LOCNESS).

In [2] was introduced the most recent sequence-to-sequence approach that uses a transformer-based encoder model as a base for sequence-to-sequence system. The base encoder model is pre-trained BERT ([8]), which is then fine-tuned on GEC data. This fine-tuning is perform on two tasks: MLM (Masked Language Model objective from [8]) and GED (Grammatical Error Detection). The encoder model adjusted this way is used to generate additional features in a sequence-to-sequence target model.

Problem of an inadequate amount of supervised training data was addressed in [9] and approached by using confusion sets to generate pseudo-data and pre-trains a sequence-to-sequence transformer. In [3] pseudo-data generation was performed via back-translation.

The main challenge of GEC is a very limited amount of annotated training data. It is relatively easy to acquire parallel texts for Machine Translation, while there are plenty of sources that provide texts in different languages. Corrected text, on the other hand, which are used for GEC, need to be proofread by human annotators. Preferably, every text should be reviewed multiple times, as in a test set for [6] and in [10].

Another aspect of this GEC task that might need closer attention is making better use of quickly improving language models. Both [2] and [4] include knowledge from models like BERT or XLNet in their approaches, but they also require quite complex pre-training phases with generated pseudo-data. The main advantage of relying more on a general-purpose model is that the target GEC system will get better together with constantly improving language models.

This paper investigates the possibilities of applying pre-trained sequence-to-sequence models for grammatical error correction and proves that fine-tuning is sufficient for achieving an efficient error correction model. This approach enables developing such models relatively quickly, with limited computational resources and limited data. Furthermore, after applying remark combination, it is possible to improve state-of-the-art results for GEC.

II. Model

In this section we describe our design decisions regarding model architecture, training and processing model output.

A. Architecture

In our approach we treat GEC as a sequence-to-sequence text transformation task, similar to machine translation. We choose the Transformer architecture ([11]) for our model because of plenty of successful applications of this model type in NLP problems (for example, machine translation [12], summarization [13] or question answering [14]). The most imporant transformer-based sequence-to-sequence models are GPT-2 ([15], T5 ([16]) and BART ([17]). Therefore, we use the pre-trained transfomer-based sequence-to-sequence model. Unlike encoder-models like BERT, XLNet, etc. they were pre-trained on full text-to-text tasks. We choose BART as our base model because of text denoising as its pre-training objective. This method makes it a natural candidate to solve GEC which may be seen as reconstructing correct text from some erroneous input. Our best results were achieved through BART large, which contained 12 encoder and 12 decoder layers and embedding dimension size equal to 1024.

B. Training

In contrast to other text generation tasks, in GEC difference between input and output text is relatively small. That impacts training and inference of the model. During training we try to set up configuration that would lead model to copy an input to the output with corrected language as the only adjustment. We noticed that both the amount of training data and training time need to be small and accurately selected to meet this goal. In Section IV we will describe the impact of data set source and size.

C. Inference

As for inference, an important distinction from other NLP tasks is the type of a decoding method. For example, in Machine Translation a common approach is to use Beam Search heuristic or such methods as Sampling, Sampling with Temperature ([18]) to aid diverse and human-like, natural output. However, in our model, applying Beam Search or Sampling led to very noisy output, and the best results required greedy selection of elements in the generated sequence (so the Beam parameter was set to 1).

D. Ensembling Method

The final output of the GEC task might be considered a set of text remarks that transforms the original text into the target one. It might be beneficial from the educational point of view, but also might be used to achieve performance improvement by combining remarks from multiple model instances. ERRANT [19] grading and annotation tool enables one to extract atomic remarks from parallel texts. Fig. 1 show examples of annotated sentences. The lines starting with S contain original sentences, following the lines starting with A which contain remarks. Every remark describes the annotation span, type and value. For example, in the sentence from Listing 1: *I think that the public transport will always be in the future* the first remark suggests removing the definite article, by defining the span from the 3rd to 4th token and empty replacement text. The second remark suggests replacing the infinitive *be* with *exist.*

We propose a simple and effective algorithm of combining remarks. Every model returns remarks for input text. Same remark may be produced by multiple models. Let us define output of a model i as:

$$M_i = \{r_1, .., r_N\} \tag{1}$$

where r is a single text remark. Then multi-set of model ensemble output M is defined as a tuple:

$$M = \left(\bigcup_i M_i, m\right) \tag{2}$$

where m is a function that gives every remark its number of occurrences:

$$m : \bigcup_i M_i \to \mathbb{N} \tag{3}$$

In practice, for simple texts, all sets of remarks are exactly the same. For more ambiguous texts, different model checkpoint outputs will differ. To combine different remark sets, we define the parameter R, which stands for *required remark frequency*, so the ensemble output M_e will be:

$$M_e = \{r : r \in M \land m(r) \geq R\} \tag{4}$$

Only the remark present in at least R model outputs will be chosen to the combined output. For example, if $R = 1$, we take all remarks from all models, and if $R = N$, where N equals a number of model checkpoints, only the remarks present in all model outputs are used in the target output. Increasing R forces only highly probable remarks to be selected for the target set; decreasing R results in selecting more remarks for the target set (see Fig. 2 that displays impact of R on Precision and Recall).

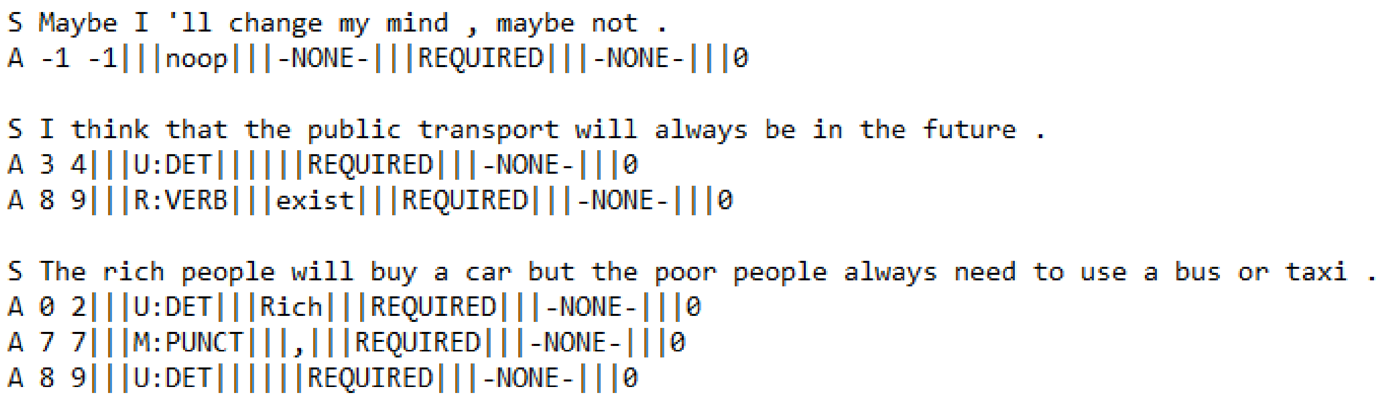
TABLE I. Training Data Sets. Sentence Count before and after Filtering Sentences without Errors

Name	Sentences	After filtering
WI+Locness	34308	34230
FCE	28350	28330
NUCLE	57113	21314
LANG-8	1041409	574180

III. Data

Four publicly available data sets were used for training experiments (listed in Table I), all of them having been described for the BEA 2019 Workshop shared tasks [6].

During the pre-processing, all sentences whose byte-pair representation was longer than 400 were removed from the training set (it was no more than 0.5
% of all data), which allow for using bigger batches during the training. This, in turn, sped up model convergence. Furthermore, we tried the approach introduced in [2] and removed sentence pairs without any corrections. We achieved the best results after all the correct sentence pairs were removed from the NUCLE and LANG-8 datasets.



```
S Maybe I 'll change my mind , maybe not .
A -1 -1|||noop|||-NONE-|||REQUIRED|||-NONE-|||0

S I think that the public transport will always be in the future .
A 3 4|||U:DET|||||REQUIRED|||-NONE-|||0
A 8 9|||R:VERB|||exist|||REQUIRED|||-NONE-|||0

S The rich people will buy a car but the poor people always need to use a bus or taxi .
A 0 2|||U:DET|||Rich|||REQUIRED|||-NONE-|||0
A 7 7|||M:PUNCT|||,|||REQUIRED|||-NONE-|||0
A 8 9|||U:DET|||||REQUIRED|||-NONE-|||0
```

Fig. 1. Examples of Sentences Annotated by ERRANT. Every Annotation Line (Starting with `A`) Describes Source Text Adjustment (Starting with `S`). Every Annotation Defines the Span, Remark Type and Replacement Text.

Multiple evaluation sets is an important criterion to prevent domain overfitting. The model is evaluated on Write and Improve development and test datasets introduced in [6], the CONLL2014 test set introduced in [10] and the FCE test dataset adjusted for the BEA workshop.

Evaluation on WI+Locness was performed by ERRANT [19]. The FCE and CONLL dataset results were measured by the M2 scorer [20]. However, except Table VI, all the results were reported by the ERRANT score. The M2 scorer was used only to allow for comparison with other reported results.

ERRANT and M2 evaluation method is based on text edits comparison. For every input sentence, measured system outputs some hypothesis. This hypothesis might be considered as a set of text edits E.

$$E = \{e_1, .., e_n\} \tag{5}$$

Every sentence has some gold standard edits G.

$$G = \{g_1, .., g_n\} \tag{6}$$

[20] defines precision and recall of system hypothesis as:

$$P = \frac{\sum_{i=1}^{n} |e_i \bigcap g_i|}{\sum_{i=1}^{n} |e_i|} \tag{7}$$

$$R = \frac{\sum_{i=1}^{n} |e_i \bigcap g_i|}{\sum_{i=1}^{n} |g_i|} \tag{8}$$

ERRANT and M2 display system edits and the gold standard in a format defined in Section II-D. ERRANT additionally generates results indicating specific error categories (such as `M:PUNCT`, which stands for *missing punctuation*).

IV. Experiments

During our experiments we measured the impact of different factors on model performance on GEC task. In Section II we emphasise the specifics of GEC. We noticed that our model setup was very sensitive for the quality and type of training data. On the other hand, a small amount of training data required precise selection of training time and learning rate to prevent overfitting. We reduced a dropout to 0.05 - higher values slowed down the model convergence and did not give any long-running benefits.

TABLE II. Training Configurations

Model size	base	large
Number of epochs	2	2
Max sentences in batch	256	16
Max tokens	512	512
Max updates	8216	8772
Warm-up	411	414
Learning rate	7e-05	3e-05
Dropout	0.05	0.05

The model was trained using the Fairseq toolkit [21], adopting the general configuration designed for translation tasks. This setting requires providing dictionaries, which in our case were the same for both the source and target language. The baseline model was BART in two versions: base (140M parameters) and large (400M parameters). BART requires text pre-processed by the byte-pair encoder, introduced in [15].

All experiments were performed on single GPU (Geforce RTX 2080 11GB), on Python version 3.7.6.

A. *Configuration*

An optimizer used for training was Adam [22] with label-smooth cross-entropy loss function [23]; the learning rate was set according to a polynomial schedule. All the training and learning rate schedule parameters, except those listed in Table II, were left unchanged from their default values. The polynomial schedule in its default configuration (a polynomial degree equals 1) basically increases the learning rate from 0 to the max value during warm-up phase and then linearly decreases. Token and sentence limits were set to facilitate a single batch fit into the available GPU memory.

B. *Data Set Impact*

In our approach, the base model is already trained on reconstructing noisy text. During the fine-tuning phase, we show pairs of correct/incorrect text, which alters model behavior to precisely fix a specific set of text modifications. We investigated the impact of including different data sets into the training set. Table III shows detailed results achieved on three development sets, while adding data to the training set. It proved to be important that a training data includes high-quality corrected texts (WI, FCE), and adding texts from other sources may degrade model performance.

Training only on the WI dataset yields average results of 53.65 for WI and 54.38. These results are almost as good as achieved by bigger training sets, but on FCE it gets only 48.49, which is significantly worse than further results. Adding the FCE training set improves score on the FCE test set to 53.22, without degrading results on other test sets. After adding $NUCLE$, average results on FCE increases slightly to 53.73 and on CONLL-14, to 55.75. However, the models trained only on WI and FCE, without $NUCLE$, achieve better result when multiple model output is combined. Data from the LANG-8 dataset caused quite a significant drop on all the test sets, which might be caused by difference in annotation quality between the training and test data. The LANG-8 annotations were created by native speakers - collaborative users of the LANG-8 learning service. The WI test set was created from selected *Write and Improve* service submissions, mixed with parts of the *LOCNESS* essay corpus and annotated 5 times by *Write and Improve* annotators.

C. *Model Size Impact*

For comparison, Table IV shows results for a smaller version of pre-trained BART containing 140M parameters. Both model types were trained from 10 different random initialization points. Results reported in the table are: the best checkpoint result, an average of all 10 checkpoints, an ensemble containing 3 models (an average of 10 random combinations of size 3) and an ensemble containing all 10 models. A detailed description of the ensembling method is provided in Section IV-D.

The smaller model achieves an average of 41.96 F-score, which, comparing to the LARGE version score of 53.36, is significantly worse, but its inference time is 2 times better, which might be an important quality when considering the production use.

D. *Combining Output*

Table V shows a change of $F_{0.5}$ on BEA-Dev dataset while changing values of R and N (parameters of the output combination algorithm, see Section II), and Fig. 2 showcases a trade-off between precision and recall for an ensemble of 10 checkpoints and a changing value of R.

The different model checkpoints are trained using the same train sets and configuration, but are initialized with different random values. Adding models to an ensemble allows for better overall correction-quality but requires longer inference time.

Thanks to the method described in Section II, the overall reported performance for BEA19 can increase by almost 4%, where a single model achieves 69.80, and after ensambling, it reaches 73.90.

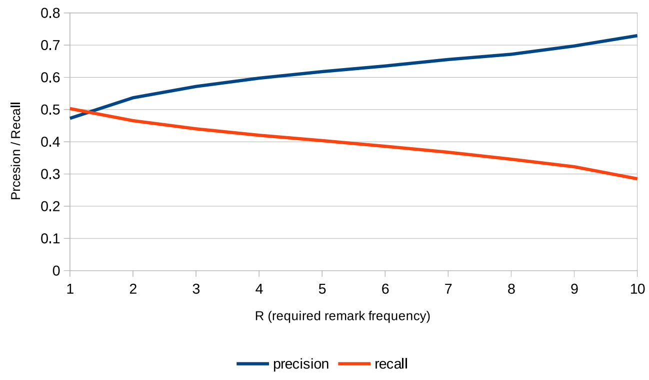


Fig. 2. Impact of Changing Remark Frequency R (see Section II-D) for Ensemble of 10 Models. Measurements were Performed on BEA-dev. For Detailed Results on Different Ensembles see Table V.

E. *Results Summary*

Single models were selected by comparing results from development sets, so the value reported on the BEA19 test comes from the checkpoint that achieved the best result on the BEA19 development set. In the case of CONLL14, where there is no development set available, the reported value is an average of 10 randomly initialized checkpoints. Table VI comprises results reported in current research papers and those achieved by our model. We report the best score on the BEA19 test and development sets. The scores on CONLL2014 and FCE are not far from the best reported results. These results, achieved by relatively low-resource fine-tuning, suggest that the GEC models might greatly benefit from a pre-trained model. A sequence-to-sequence denoising pre-training objective uses similar text transformations, as commonly required in GEC. [17] uses token masking, token deletion, token infilling, sentence permutation and document rotation. After fine-tuning, the model identifies a subset of these transformations specific to GEC. Remark-based ensambling proved to be a reasonable method to increase the correction precision, which improves the overall score, but it is also important for further model applications, where false positive remarks might be very misleading, especially in educational systems.

V. Conclusion and Future Work

A fine-tuned sequence-to-sequence transformer model is very effective in solving the GEC task. It was able to achieve state-of-the-art results on the BEA19 test set 73.90 and development set 56.58. It proves that an efficient model solving GEC might be trained in a matter of hours using a single GPU. Only a limited amount of human-annotated data was required.

What is also beneficial in our approach is that it facilitates leveraging future progress in general-purpose language models. Single language models followed by multi-lingual systems will enable solving GEC for languages other than English. Constantly expanding transformer-based models extend the limits of text comprehension and may improve GEC performance without costly data annotation and only with low-resource and fast fine-tuning.

TABLE III. Results Achieved for Training Sets Containing Data from Different Sources. Column `max` Displays Best Checkpoint, `avg` Average Result, `ens 10` Result after Combining Checkpoints from All Remarks

Train data	Sentence count	**BEA-dev (ERRANT)**		
		max	avg	ens 10
W&I	34K	53.65	53.34	56.74
W&I, FCE	63K	54.07	53.36	56.58
W&I, FCE, NUCLE	84K	53.29	52.73	55.42
W&I, FCE, NUCLE, LANG8	658K	48.87	35.79	41.03
Train data	Sentence count	**FCE-test (ERRANT)**		
		max	avg	ens 10
W&I	34K	49.35	48.49	51.61
W&I, FCE	63K	53.72	53.22	57.48
W&I, FCE, NUCLE	84K	54.5	53.73	57.94
W&I, FCE, NUCLE, LANG8	658K	52.58	52.18	56.46
Train data	Sentence count	**CoNLL-14 (ERRANT)**		
		max	avg	ens 10
W&I	34K	54.95	54.38	58.82
W&I, FCE	63K	55.48	54.68	58.81
W&I, FCE, NUCLE	84K	56.28	55.75	57.94
W&I, FCE, NUCLE, LANG8	658K	55.58	54.63	57.04

TABLE IV. Results for Different Model Sizes

Model size	parameters	**BEA-dev (ERRANT)**			
		max	avg	ens 3	ens 10
BASE	140M	44.34	41.96	43.32	45.66
LARGE	400M	54.07	53.36	56.19	56.58
Model size	parameters	**FCE-test (ERRANT)**			
		max	avg	ens 3	ens 10
BASE	140M	45.29	42.20	45.97	47.08
LARGE	400M	53.72	53.22	57.04	57.48
Model size	parameters	**CoNLL-14 (ERRANT)**			
		max	avg	ens 3	ens 10
BASE	140M	48.23	46.30	48.36	50.13
LARGE	400M	55.48	54.68	58.70	58.81

TABLE V. BEA-dev Results for Ensembles Containing Different Model Count (Parameter N on Horizontal Axis) and Changing Remarks Frequency Parameter R (Vertical Axis, See Section II-D)

R/N	2	3	4	5	6	7	8	9	10
1	52.83	51.72	51.03	50.12	49.59	48.99	48.60	48.22	47.85
2	55.87	55.46	54.76	54.18	53.74	53.21	52.79	52.42	52.07
3		56.19	56.09	55.82	55.37	55.04	54.66	54.28	53.95
4			56.15	56.47	56.25	56.02	55.67	55.39	55.10
5				56.10	56.49	56.44	56.30	56.05	55.85
6					56.04	56.62	56.57	56.50	56.27
7						55.94	56.60	56.56	56.66
8							55.86	56.57	56.52
9								55.73	**56.58**
10									55.62

TABLE VI. Results Summary (measured by ERRANT Except Two Columns for CONLL14 and FCE-Test Labeled Accordingly as Measured by M^2 Scorer)

	Results				
	BEA-test	**BEA-dev**	**CoNLL14** (M^2)	**CoNLL14**	**FCE-Test** (M^2)
Single model result					
GECToR [4]	72.40	55.50	65.30	-	-
Transformer + Pseudo data[3]	64.20	45.90	61.30	56.00	-
BERT-fuse[2]	-	-	-	-	**61.20**
Model ensembles					
GECToR [4]	73.63	-	**66.50**	-	-
Combined systems [24]	73.18	-	-	-	-
Transformer + Pseudo data[3]	70.20	-	65.00	**60.90**	-
BERT-fuse[2]	69.80	-	65.20	-	59.40
Transformer + Pseudo data based on confusion sets[9]	69.47	53.00	64.16	-	55.81
Sequential transfer learning [25]	69.06	52.79	-	-	-
Ours – single model	69.80	54.36	61.92	55.27	59.11
Ours – 3 models	73.13	56.19	62.48	58.70	59.32
Ours – 10 models	**73.90**	**56.58**	62.48	58.81	59.54

Acknowledgment

This work was supported by the Polish National Centre for Research and Development and is a part of the project ”System automatycznego wykrywania bledow jezykowych i sugerowa-

nia poprawek w tekstach pisanych w jezyku angielskim".

REFERENCES

[1] Howson, Paul, John Dubber, John Knagg, Mona Lotten, Mike Waldron, John Worne, Hannah Connell, Paul Howson, and John Dubber. 2013. The english effect.

[2] Kaneko, Masahiro, 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. *ArXiv*, abs/2005.00987.

[3] Kiyono, Shun, Jun Suzuki, Masato Mita, Tomoya Mizumoto, and Kentaro Inui. 2019. An empirical study of incorporating pseudo data into grammatical error correction. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

[4] Omelianchuk, Kostiantyn, Vitaliy Atrasevych, Artem N. Chernodub, and Oleksandr Skurzhanskyi. 2020. Gector - grammatical error correction: Tag, not rewrite. *ArXiv*, abs/2005.12592.

[5] Awasthi, Abhijeet, 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, Association for Computational Linguistics, Hong Kong, China.

[6] Bryant, Christopher, 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, Association for Computational Linguistics, Florence, Italy.

[7] 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)*, pages 198–202, Association for Computational Linguistics, Jeju Island, Korea.

[8] Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

[9] Grundkiewicz, Roman, 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, Association for Computational Linguistics, Florence, Italy.

[10] Ng, Hwee Tou, 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, Association for Computational Linguistics, Baltimore, Maryland.

[11] Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, L. Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *ArXiv*, abs/1706.03762.

[12] Edunov, Sergey, Myle Ott, M. Auli, and David Grangier. 2018. Understanding back-translation at scale. *ArXiv*, abs/1808.09381.

[13] Yan, Yu, Weizhen Qi, Yeyun Gong, Dayiheng Liu, Nan Duan, Jiusheng Chen, Ruofei Zhang, and Ming Zhou. 2020. Prophetnet: Predicting future n-gram for sequence-to-sequence pre-training. *arXiv preprint arXiv:2001.04063*.

[14] Yang, Z., Zihang Dai, Yiming Yang, J. Carbonell, R. Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*.

[15] Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

[16] Raffel, Colin, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *ArXiv*, abs/1910.10683.

[17] Lewis, Mike, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

[18] Ippolito, Daphne, Reno Kriz, João Sedoc, Maria Kustikova, and Chris Callison-Burch. 2019. Comparison of diverse decoding methods from conditional language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3752–3762, Association for Computational Linguistics, Florence, Italy.

[19] 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)*, pages 793–805, Association for Computational Linguistics, Vancouver, Canada.

[20] 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*, pages 568–572, Association for Computational Linguistics, Montréal, Canada.

[21] Ott, Myle, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*.

[22] Kingma, Diederik P. and Jimmy Ba. 2015. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

[23] Szegedy, Christian, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826.

[24] Kantor, Yoav, Yoav Katz, Leshem Choshen, Edo Cohen-Karlik, Naftali Liberman, Assaf Toledo, Amir Menczel, and Noam Slonim. 2019. Learning to combine grammatical error corrections. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 139–148, Association for Computational Linguistics, Florence, Italy.

[25] Choe, Yo Joong, Jiyeon Ham, Kyubyong Park, and Yeoil Yoon. 2019. A neural grammatical error correction system built on better pre-training and sequential transfer learning. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 213–227, Association for Computational Linguistics, Florence, Italy.