CUNI System for the Building Educational Applications 2019 Shared Task: Grammatical Error Correction

Jakub Náplava and **Milan Straka**
Charles University,
Faculty of Mathematics and Physics,
Institute of Formal and Applied Linguistics
{naplava,straka}@ufal.mff.cuni.cz

Abstract

In this paper, we describe our systems submitted to the Building Educational Applications (BEA) 2019 Shared Task (Bryant et al., 2019). We participated in all three tracks. Our models are NMT systems based on the Transformer model, which we improve by incorporating several enhancements: applying dropout to whole source and target words, weighting target subwords, averaging model checkpoints, and using the trained model iteratively for correcting the intermediate translations. The system in the Restricted Track is trained on the provided corpora with oversampled "cleaner" sentences and reaches 59.39 F0.5 score on the test set. The system in the Low-Resource Track is trained from Wikipedia revision histories and reaches 44.13 F0.5 score. Finally, we finetune the system from the Low-Resource Track on restricted data and achieve 64.55 F0.5 score, placing third in the Unrestricted Track.

1 Introduction

Starting with the 2013 and 2014 CoNLL Shared Tasks on grammatical error correction (GEC), much progress has been done in this area. The need to correct a variety of error types lead most researchers to focus on models based on machine translation (Brockett et al., 2006) rather than custom designed rule-based models or a combination of single error classifiers. The machine translation systems turned out to be particularity effective when Junczys-Dowmunt and Grundkiewicz (2016) presented state-of-the-art statistical machine translation system. Currently, models based on statistical and neural machine translation achieve best results: in restricted settings with training limited to certain public training sets (Zhao et al., 2019); unrestricted settings with no restrictions on training data (Ge et al., 2018); and also in low-resource track where the training data should not come from any annotated corpora (Lichtarge et al., 2018).[1]

In this paper, we present our models and their results in the restricted, unrestricted, and low-resource tracks. We start with a description of related work in Section 2. We then describe our systems together with the implementation details in Section 3. Section 4 is dedicated to our results and ablation experiments. Finally, in Section 5 we conclude the paper with some proposals on future work.

2 Related Work

Transformer (Vaswani et al., 2017) is currently one of the most popular architectures used in machine translation. Its self-attentive layers allow better gradient flow when compared to recurrent neural models and the masking in decoder provides faster training. Junczys-Dowmunt et al. (2018) propose several improvements for training Transformer on GEC: using dropout on whole input words, assigning weight to target words based on their alignment to source words, and they also propose to oversample sentences from the training set in order to have the same error rate as the test set.

Majority of work in grammatical error correction has been done in restricted area with a fixed set of annotated training datasets. Lichtarge et al. (2018), however, show that training a neural machine translation system from Wikipedia edits can lead to surprisingly good results. As the authors state, corpus of Wikipedia edits is only weakly supervised for the task of GEC, because most of the edits are not corrections of grammatical errors and also they are not human curated specifically for GEC. To overcome these issues, the authors use iterative decoding which allows for incremental corrections. In other words, the model can re-

[1] Note that in this settings Wikipedia revisions are allowed

peatedly translate its current output as long as the translation is more probable then keeping the sentence unchanged. Similar idea is also presented in (Ge et al., 2018), where the translation system is trained with respect to the incremental inference.

3 Our System

In this section, we present our three systems submitted to each track of the BEA 2019 Shared Task. We start with the Restricted Track In Section 3.1, where we present a series of improvements to the baseline Transformer model. In Section 3.2, we describe our model trained on Wikipedia revisions which was submitted to the Low-Resource Track. Finally, in Section 3.3, we describe the model submitted to the Unrestricted Track.

All our models are based on the Transformer model from Tensor2Tensor framework version 1.12.0.[2]

3.1 Restricted Track

In the Restricted Track, we use the 5 provided datasets for system development: FCE v2.1 (Yannakoudakis et al., 2011), Lang-8 Corpus of Learner English (Mizumoto et al., 2011; Tajiri et al., 2012), NUCLE (Dahlmeier et al., 2013), Write & Improve (W&I) and LOCNESS v2.1 (Bryant et al., 2019; Granger, 1998). From Lang-8 corpus, we took only the sentences annotated by annotators with ID 0 (A0) and ID 1 (A1). All but the development sets from W&I and LOCNESS datasets were used for training. The simple statistics of these datasets are presented in Table 1. The displayed error rate is computed using maximum alignment of original and annotated sentences as a ratio of non-matching alignment edges (insertion, deletion, and replacement).

We use the *transformer_base* configuration of Tensor2Tensor as our baseline solution. The training dataset consists of 1 230 231 sentences. After training, beam search decoding is employed to generate model corrections and we choose the checkpoint with the highest accuracy on a development set concatenated from the W&I and LOCNESS development sets.

3.1.1 Transformer Big

The first minor improvement was to use the *transformer_big* configuration instead of *transformer_base*. This configuration has bigger capacity and as Popel and Bojar (2018) show, it reaches substantially better results on certain translation tasks.

[2] https://github.com/tensorflow/tensor2tensor

Dataset		Sentences	Average error rate
Lang8	A0	1 037 561	13.33 %
	A1	67 975	25.84 %
FCE v2.1	train	28 350	11.31 %
	dev	2 191	11.67 %
	test	2 695	12.87 %
NUCLE		57 151	6.56 %
W&I	train A	10 493	18.13 %
	train B	13 032	11.68 %
	train C	10 783	5.62 %
	dev A	1 037	18.32 %
	dev B	1 290	12.46 %
	dev C	1 069	5.91 %
LOCNESS	dev N	998	4.72 %

Table 1: Statistics of available datasets. The error rate is computed as a ratio of non-matching alignment edges.

3.1.2 Source and Target Word Dropout

Dropout (Srivastava et al., 2014) is a regularization technique that turned out to be particularly effective in the field of neural networks. It works by masking several randomly selected activations during training, which should prevent the neural network from overfitting the training data. In the area of NLP, it is a common approach to apply dropout to whole embeddings, randomly zeroing certain dimensions. As Junczys-Dowmunt et al. (2018) show, we can also apply dropout to whole source words to reduce trust in the source words. Specifically, full source word embedding vector is set to zero vector with probability p. We further note this probability as the *source_word_dropout*.

To make regularization even more effective, we decided to dropout also whole target word embeddings. We refer to the probability with which we dropout entire target word embeddings as the *target_word_dropout*.

3.1.3 Edited MLE

Compared to traditional machine translation task, whose goal is to translate one language to another, GEC operates on a single language. Together with the relatively low error rate, the translation system may converge to a local optimum, in which the

model copies the input unchanged to the output. To overcome this issue, Junczys-Dowmunt et al. (2018) propose to change the maximum likelihood objective to assign bigger weights to target tokens different from the source tokens. More specifically, they start by computing the word alignment between each source $x = (x_0, x_1, ..x_N)$ and target sentence $y = (y_0, y_1, ...y_M)$. Then they set the weight λ_t of the target word y_t to 1 if it is matched, and otherwise, if it is an insertion or replacement of a source token, λ_t is set to some predefined constant. Modified log-likelihood training objective then takes following form:

$$L(x, y) = -\sum_{t=1}^{M} \lambda_t \log P(y_t|x, y_0, \ldots, y_{t-1}).$$

3.1.4 Data oversampling

It is crucial to have training data from the same domain as the test data, i.e., training data containing similar errors with similar distribution as the test data. As we can see in the Table 1, the vast majority of our training data comes from the Lang-8 corpus. However, as it is quite noisy and of low quality, it matches the target domain the least. Therefore, we decided to oversample other datasets. Specifically, we add the W&I training data 10 times, all FCE data 5 times and NUCLE corpus 5 times to the training data. The oversampled training set consists of 1 900 551.

In Table 1, we can also see token error rate of each corpus. The development error rate in W&I and LOCNESS varies from 5.91% up to 18.32%. This gives us a basic idea how the test data looks like, and since the test data does not contain annotations from which set (`A`, `B`, `C`, `N`) it comes, we decided not to optimize the training data against the token error rate any further.

3.1.5 Checkpoint Averaging

Popel and Bojar (2018) report that averaging several last Transformer model checkpoints during training leads both to lower variance results and also to slightly better performance than the baseline without averaging. They propose to save checkpoints every one hour and average either 8 or 16 last checkpoints. Since we found out that the model overfits the oversampled dataset quite quickly, we save checkpoints every 30 minutes.

3.1.6 Iterative decoding

A system for grammatical error correction should correct all errors in the text while keeping the rest of the text intact. In many situations with multiple errors in a sentence, the trained system, however, corrects only a subset of its errors. Lichtarge et al. (2018) and Ge et al. (2018) propose to use the trained system iteratively to allow the system to correct certain errors during further iterations. Iterative decoding is done as long as the cost of the correction is less than the cost of the identity translation times a predefined constant. While Lichtarge et al. (2018) use the same trained model log-likelihoods as the cost function, Ge et al. (2018) utilize an external language model for it. Because the restricted track does not contain enough training data to train a quality language model, we adopted the first approach and utilize the trained system log-likelihoods as a stopping criterion.

The iterative decoding algorithm we use is presented in Algorithm 1. Note that when the resulting beam does not contain the identical (non-modified) sentence, the correction with the lowest cost is returned regardless of the provided threshold. We adopted this approach for two reasons – efficiently obtaining the log-likelihood of the identical sentence would require non-trivial modification of the Tensor2Tensor framework, and for

Data: *input_sent*; *max_iters*; *threshold*
for *iter in [1,2,..,max_iters]* **do**
  beam_results = decode(*input_sent*);
  identity_cost = $+\infty$;
  non_identity_cost = $+\infty$;
  non_identity_sent = None
  for *beam_item in beam_results* **do**
    text = *beam_item*["text"];
    cost = *beam_item*["cost"];
    if *text* == *input_sent* **then**
      identity_cost = *cost*;
    else if *cost* < *non_identity_cost* **then**
      non_identity_cost = *cost*;
      non_identity_sent = *text*;
  end
  if *non_identity_cost* ≤ *threshold* · *identity_cost* **then**
    input_sent = *non_identity_sent*;
  else
    break;
  end
end
return *input_sent*;

Algorithm 1: Iterative decoding algorithm

threshold $>$ 1 (i.e., allow generating changes which are less likely than identical sentence) the results are the same.

3.1.7 Implementation Details

Apart from the first experiment in which we use *transformer_base* configuration, all our experiments are based on *transformer_big* architecture. We use Adafactor optimizer (Shazeer and Stern, 2018), linearly increasing the learning rate from 0 to 0.011 over the first 8000 steps, then decrease it proportionally to the number of steps after that.[3] We also experimented with Adam optimizer with default learning rate schedule, however, training converged poorly. We hypothesise that this was caused by the higher learning rate.

All systems are trained on 4 Nvidia P5000 GPUs for approximately 2 days. The vocabulary consists of approximately 32k most common word-pieces, batch size is 2000 word-pieces per each GPU and all sentences with more than 150 word-pieces are discarded. Model checkpoints are saved every 30 minutes. We ran a grid search to find values of all hyperparameters described in the previous sections.

At evaluation time, we run iterative decoding using a beam size of 4. Beam-search length-balance decoding hyperparameter alpha is set to 0.6. This applies to all further experiments.

3.2 Low-Resource Track

The dataset for our experiments in the Low-Resource Track consists of nearly 190M segment pairs extracted from Wikipedia XML revision dumps. To acquire these, we downloaded all English Wikipedia revision dumps (155GB in size) and processed them with the *WikiRevision* dataset problem from Tensor2Tensor. The processing pipeline extracts individual pages with chronological snapshots, removes all non-text elements and downsamples the snapshots. With low probability, additional spelling noise is added by either inserting a random character, deleting a random character, transposing two adjacent characters or replacing a character with a random one. With the same low probability, a random text substring (up to 8 characters) may also be replaced with a marker, which should force the model to learn infilling. Finally, the texts from two consecutive snapshots are aligned and sequences between matching segments are extracted to form a training pair. Only 4% of identical samples are preserved.

Despite having an enormous size compared to 1.2M sentences in the Restricted Track, the training pairs extracted from Wikipedia are extremely noisy, containing a lot of edits that are in no sense grammatical correction. It is also worth noting that the identical data modified by the spelling and infilling operations form nearly 50% of the training pairs.

Since we want to re-use the system in other scenarios, we train the model on the original (untokenized) training data. To evaluate the model on the BEA development and test data, we detokenize the data using Moses,[4] run model inference and finally tokenize corrected sentences using spaCy.[5]

The training segments may contain newline and tab symbols; therefore, we applied additional post-processing in which we replaced both these symbols with spaces.

Because overfitting should not be an issue with the Wikipedia data, we decided to use *transformer_clean_big_tpu* configuration, following Lichtarge et al. (2018). This configuration, compared to *transformer_big*, performs no dropouts. The vocabulary consists of approximately 32k most common word-pieces, batch size is 2000 word-pieces per each GPU and all sentences with more than 150 word-pieces are discarded. We train the model for approximately 10 days on 4 Nvidia P5000 GPUs. After training, the last 8 checkpoints saved in 1 hour intervals are averaged. Finally, we run a grid search to find optimal values of *threshold* and *max_iters* in iterative decoding algorithm.

3.3 Unrestricted Track

Our system submitted to the Unrestricted Track is the best system from the Low-Resource Track finetuned on the oversampled training data as described in Section 3.1.4. Since our system in the Unrestricted Track was trained on detokenized data, the training sentences for finetuning were also detokenized. The tokenization and detokenization was done in the same way as described in Section 3.2.

We finetune the system with the Adafactor optimizer. The learning rate linearly increases from

[3] We use 8000 warmup steps and learning_rate_schedule=rsqrt_decay

[4] We use mosestokenizer v1.0.0 and its detokenizer.

[5] We use spaCy v1.9.0 and the en_core_web_sm-1.2.0 model.

Track	P	R	$F_{0.5}$	Best	Rank
Restricted	67.33	40.37	59.39	69.47	10 / 21
Unrestricted	68.17	53.25	64.55	66.78	3 / 7
Low Resource	50.47	29.38	44.13	64.24	5 / 9

Table 2: Official shared task $F_{0.5}$ scores on the test set.

System	A	B	C	N	Combined
Transformer-base architecture	39.98	32.68	23.97	14.49	32.47
Transformer-big architecture	39.70	35.13	26.22	20.20	34.20
+ 0.2 src drop, 0.1 tgt drop, 3 MLE	42.06	38.25	28.72	23.80	38.15
+ Extended dataset	45.99	41.79	32.52	27.89	40.86
+ Averaging 8 checkpoints	47.90	44.13	36.19	29.05	43.29
+ Iterative decoding	48.75	45.46	37.09	30.19	44.27

Table 3: Development combined $F_{0.5}$ score of incremental improvements of our system.

0 to 0.0003 over the first 20 000 steps and then remains constant. We employ source word dropout, target word dropout and weighted MLE. The training data for finetuning and the rest of the training scheme are identical to Section 3.1.7.

4 Results

We now present the results of our system. Additionally, we present several ablation experiments, which are evaluated on the concatenation of W&I and LOCNESS development sets (the *Dev combined*).

4.1 Shared Task Results

The official results of our three systems on the blind test set are presented in Table 2. All our systems have substantially higher precision than recall. It is an interesting observation that the system in the unrestricted track has similar precision as the model in the restricted track while having higher recall.

4.2 Restricted Track

The first experiment we conducted is devoted to the incremental enhancements that we proposed in Section 3.1. As Table 3 indicates, applying each enhancement results in higher performance on the development set. By applying all incremental improvements, total $F_{0.5}$ score on the development set increases by 11.8%.

We improved the $F_{0.5}$ score by adding *source_word_dropout*, *target_word_dropout* and MLE weighting by almost 4%. To find out optimal values of all three hyper-parameters, we ran a small grid search. The results of this experiment are presented in Table 4. The source-word dropout improves the results the most, MLE provides minor gains, while the influence of target-word dropout on the results is unclear.

Source word dropout	Target word dropout	MLE	Dev combined $F_{0.5}$
0	0	1	34.20
0.1			37.89
0.2			38.26
	0.1		35.43
	0.2		33.98
		2	34.56
		3	34.28
		4	34.17
0.2	0.1		37.89
0.2		3	38.68
0.2	0.1	3	38.15

Table 4: The effect of source word dropout, target word dropout, and MLE weight on development combined $F_{0.5}$ score.

In the next experiment, we examined the effect of checkpoint averaging. Table 5 presents results of the model without averaging and with averaging 4, 6, and 8 model checkpoints. The best results are achieved when 8 checkpoints are used and the results indicate that the more checkpoints are averaged the better the results are.

Finally, we inspect the effect of iterative decod-

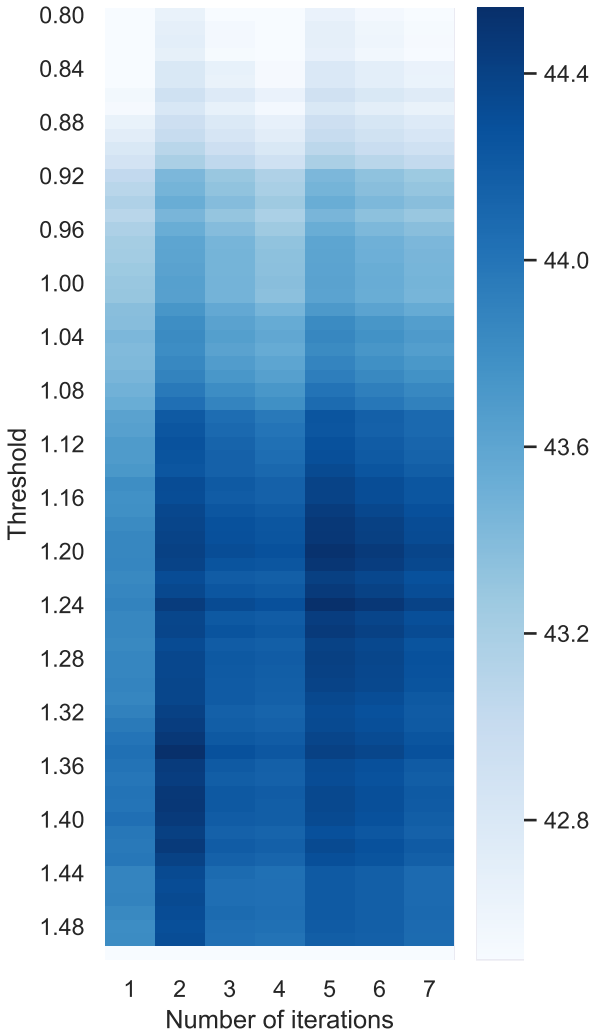

Figure 1: Performance of iterative decoding depending on number of iterations and threshold parameters.

ing. Specifically, we run an exhaustive grid search to find optimal values of *threshold* and *max_iters*. The results of this experimented are visualised in Figure 1. We can see that increasing *threshold* from 1 to values around 1.20 leads to substantially better results. Moreover, using more iterations also has a positive impact on the model performance. Both of these improvements are caused by the model generating more corrections which are deemed less likely to the model, i.e., we increase recall at the expense of precision.

4.3 Low-Resource Track

We train following models in the Low-Resource Track:

1. the *transformer_big* configuration with *input_word_dropout* set to 0.2 and *target_word_dropout* to 0.1 – settings similar to the best system in the Restricted Track but without edited MLE;

2. the *transformer_clean_big_tpu* configuration – this configuration uses no internal dropouts;

3. the *transformer_clean_big_tpu* configuration with *input_word_dropout* 0.2 and *target_word_dropout* 0.1;

4. the *transformer_clean_big_tpu* configuration trained on sentences extracted from Wikipedia revisions without introducing additional spelling errors and infillment marker.

All but the fourth model use the training data as described in Section 3.2 and the training scheme is in all models identical. The results of all models are presented in Table 6.

The best results are achieved with the second model which performs no dropouts. When we incorporate source and target word dropouts in the third experiment, the performance deteriorates by more than 1%. When we also add Transformer internal dropouts in the first experiment, the performance drops by additional 2.8%. This confirms

Checkpointing	Dev combined $F_{0.5}$
No checkpointing	41.55
Averaging 4 checkpoints	43.00
Averaging 6 checkpoints	43.13
Averaging 8 checkpoints	43.29

Table 5: Maximum development combined $F_{0.5}$ score achieved by averaging the given number of checkpoints.

ID	Model	Dev combined $F_{0.5}$
1	*transformer_big* 0.2 src drop, 0.1 tgt drop	22.03
2	*transformer_clean_big_tpu* no src drop, no tgt drop	26.05
3	*transformer_clean_big_tpu* 0.2 src drop, 0.1 tgt drop	24.80
4	*transformer_clean_big_tpu* no spelling or infillment errors	21.16

Table 6: Development combined $F_{0.5}$ score achieved with different models in the Low-Resource Track.

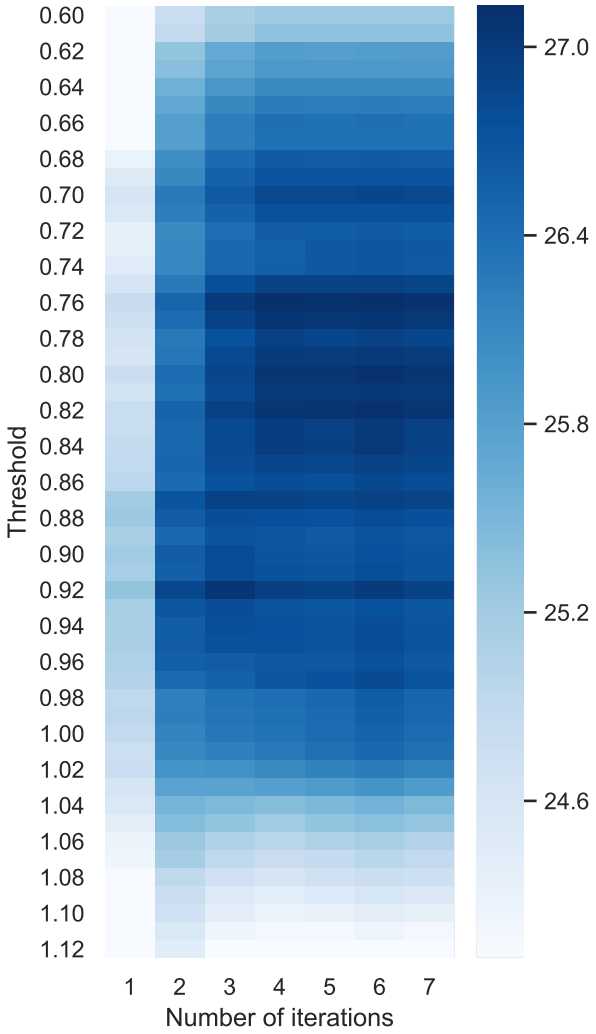

Figure 2: Performance of iterative decoding depending on number of iterations and threshold parameters.

our assumption that the enormous amount of data is strong enough regularizer and the usage of additional regularizers leads to worse performance.

The results of the fourth model, which was trained on data without additional spelling and infillment noise, are almost 5% worse than when training on data with this noise. It would be an interesting experiment to evaluate the effect of spelling and infillment noise separately, but this was not done in this paper.

We also run an exhaustive grid search to find optimal values of *threshold* and *max_iters* in iterative decoding. As we can see in Figure 2, the optimal value of *threshold* is now below 1 indicating that precision is now increased at the expense of recall. A performance gain in using more than one iteration is clearly visible.

4.4 Unrestricted Track

In the Unrestricted Track, we tried finetuning the pretrained system with two different learning rate schedules:

- linearly increase learning rate from 0 to 0.011 over the first 8000 steps, then decrease it proportionally to the number of steps after that – exactly same as while training system from scratch in the Restricted Track (see Section 3.1.7);
- linearly increase learning rate from 0 to 3e-4 then keep the learning rate constant as proposed by Lichtarge et al. (2018).

All other hyper-parameters and the training process remain the same as described in Section 3.3.

The first finetuning scheme overfitted the training corpus quite quickly while reaching score of 48.33. The second scheme converged slower and reached a higher score of 48.82.

5 Conclusion

We have presented our three systems submitted to the BEA 2019 Shared Tasks. By employing larger architecture, source and target word dropout, edited MLE, dataset extension, checkpoint averaging, and iterative decoding, our system reached 59.39 $F_{0.5}$ score in the Restricted Track, finishing 10^{th} out of 21 participants.

In the Low Resource Track, we utilized Wikipedia revision edits as a training data, reaching 44.14 $F_{0.5}$ score. Finally, we finetuned this model using the annotated training data, obtaining 65.55 $F_{0.5}$ score in the Unrestricted Track, ranking 3^{rd} out of 7 submissions.

As future work, we would like to explore iterative decoding algorithm more thoroughly. Specifically, we hope that allowing *threshold* parameter to change in each iteration might provide gains. We would also like to train systems on Wikipedia revisions in other languages.

Acknowledgements

The work described herein has been supported by OP VVV VI LINDAT/CLARIN project (CZ.02.1.01/0.0/0.0/16_013/0001781) and it has been supported and has been using language resources developed by the LINDAT/CLARIN project (LM2015071) of the Ministry of Education, Youth and Sports of the Czech Republic.

This research was also partially supported by SVV project number 260 453 and GAUK 578218 of the Charles University.

References

Chris Brockett, William B Dolan, and Michael Gamon. 2006. Correcting esl errors using phrasal smt techniques. In *Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics*, pages 249–256. 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 14th Workshop on Innovative Use of NLP for Building Educational Applications*. 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.

Tao Ge, Furu Wei, and Ming Zhou. 2018. Reaching human-level performance in automatic grammatical error correction: An empirical study. *arXiv preprint arXiv:1807.01270*.

Sylviane Granger. 1998. The computer learner corpus: A versatile new source of data for SLA research. In Sylviane Granger, editor, *Learner English on Computer*, pages 3–18. Addison Wesley Longman, London and New York.

Marcin Junczys-Dowmunt and Roman Grundkiewicz. 2016. Phrase-based machine translation is state-of-the-art for automatic grammatical error correction. *arXiv preprint arXiv:1605.06353*.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Shubha Guha, and Kenneth Heafield. 2018. Approaching neural grammatical error correction as a low-resource machine translation task. *arXiv preprint arXiv:1804.05940*.

Jared Lichtarge, Christopher Alberti, Shankar Kumar, Noam Shazeer, and Niki Parmar. 2018. Weakly supervised grammatical error correction using iterative decoding. *arXiv preprint arXiv:1811.01710*.

Tomoya Mizumoto, Mamoru Komachi, Masaaki Nagata, and Yuji Matsumoto. 2011. Mining revision log of language learning sns for automated japanese error correction of second language learners. In *Proceedings of 5th International Joint Conference on Natural Language Processing*, pages 147–155.

Martin Popel and Ondřej Bojar. 2018. Training tips for the transformer model. *The Prague Bulletin of Mathematical Linguistics*, 110(1):43–70.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. *arXiv preprint arXiv:1804.04235*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

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: Short Papers-Volume 2*, pages 198–202. 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.

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-Volume 1*, pages 180–189. Association for Computational Linguistics.

Wei Zhao, Liang Wang, Kewei Shen, Ruoyu Jia, and Jingming Liu. 2019. Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data. *arXiv preprint arXiv:1903.00138*.