Creating and evaluating Grammatical Error Correction models with arbitrary error correction profiles

TIM KLAMPE, University of Twente, The Netherlands

Generating high-quality synthetic datasets with use case-specific error frequencies can boost the performance of Grammatical Error Correction models substantially. In this paper, I propose a system in which datasets are created according to specific error frequencies with a tagged grammatical corruption model. The effect of these frequencies is then evaluated in error-specific accuracy testing.

The system can be used to flexibly generate synthetic datasets and then train a grammatical error correction model. The accuracy of said model is analyzed and then can be iteratively improved by changing error frequencies in the dataset and comparing the effects on the accuracy.

I will demonstrate the generation and evaluation of a grammatical error correction model that takes the expected error profile of a native English speaker into consideration.

Additional Key Words and Phrases: grammatical error correction, tagged corruption model, synthetic datasets

1 INTRODUCTION

Grammatical Error Correction (GEC) is an established subfield of Natural Language Processing (NLP) [2]. As such it has seen many different developments, but it was the proposal of the Transformer model architecture in 2017 [11] that provided a substantial boost to NLP and subsequently also GEC-related projects. Recent developments have reached a human level of performance in some GEC benchmarks, with the Transformer model architecture playing a central role [5].

But even with a highly optimized model architecture, the performance of the resulting model depends heavily on the quality of the dataset. Good accuracy can only be achieved if both quantitative and qualitative sufficient training data is available [4]. While there are high-quality datasets for grammatical error correction tasks are publicly available, such as the CoNLL-2014 shared task [1], the BEA-2019 Corpus [2] and the C4_200M dataset [3], these datasets provide little information on the distribution of error types. This can lead to a bias towards specific grammatical error types, which in turn can hurt the overall correction accuracy of the GEC model. But it can also improve correction accuracy if the model bias matches the error profile of the user.

Identifying such a bias towards specific grammatical errors, and modeling this error bias in a dataset, can lead to an increased correction accuracy and more realistic suggestions [8, 10]. Modeling a specific error bias in a dataset can be done in two different ways: Filtering an existing dataset, or generating a new synthetic dataset.

Filtering and especially classifying errors in an existing dataset can quickly become very compute intensive with larger dataset sizes. This approach is therefore more viable for small datasets. Instead of using exisiting sequences with grammatical corruptions, a dataset can also be created synthetically.

Previous work has shown a tagged grammatical corruption model to be highly efficient in generating synthetic datasets for GEC tasks [8]. A grammatical corruption model here is the inverse of a grammatical correction model: A grammatically correct input sequence will be returned with grammatical corruptions. A tagged corruption model is an extension of a simple corruption model, the exact error type present in the output sequence can here be controlled through the use of tags in the input sequence.

Building a synthetic dataset with a corruption model requires defining the exact distribution of error types as well as the overall size of the dataset [8]. Often these questions can only be answered through field-testing. A validation mechanism that provides accuracy numbers per error tag can be a helpful tool to help fine-tune a dataset for a specific use case. Iteratively refining the error tag distribution in the underlying dataset can create a use case-specific dataset. As shown in previous work [8], this can create human levels of accuracy. But only a reproducible evaluation mechanism allows to systematically refine the error distribution per use case.

I therefore pose the following research question:

RQ: How can we efficiently create Transformer GEC models that are skewed toward arbitrary grammatical error frequencies?

For this, we need to answer the following two sub-questions:

RQ1: How can we efficiently create synthetic datasets that are skewed towards specific error frequencies?
RQ2: How can we validate and compare the performance of models that were trained on the created synthetic datasets?

2 RELATED WORK

The paper 'Attention is all you need' proposed a new machine learning model architecture, the Transformer [11]. This proved to be groundbreaking for the field of Natural Language Processing (NLP) and subsequently also for the field of Grammatical Error Correction (GEC).

What sets the Transformer apart from other machine learning model types is the use of *attention* throughout the model. Attention mechanisms allow the modeling of dependencies without regard to their distance in the input and output sequences [11]. This means

[1] https://www.comp.nus.edu.sg/ nlp/conll14st.html
[2] https://www.cl.cam.ac.uk/research/nl/bea2019st/
[3] https://github.com/google-research-datasets/C4_200M-synthetic-dataset-for-grammatical-error-correction

TScIT 37, July 8, 2022, Enschede, The Netherlands

that complex grammatical structures and interdependencies can be learned by the model.

Using attention throughout the complete model allowed the researchers to develop a highly parallelizable model. This enables a highly efficient learning process, during which all positions of an input can be related to each other. Pre-training the model on a more general dataset been has shown to be beneficial to the accuracy of the model. The work 'Attention is all you need' produced Transformer architecture is also referred to as the Text-to-Text-Transfer-Transformer (T5) model.

The potential of the transfer architecture was further explored in the paper *'The Unreasonable Effectiveness of Transformer Language Models in Grammatical Error Correction'*[2].

The idea of using a tagged grammatical corruption model for dataset creation has been explored in the paper *'Synthetic Data Generation for Grammatical Error Correction with Tagged Corruption Models'* [8]. Their work has produced valuable insights into the design of a corruption model.

Previous research has shown that grammatical error correction can benefit from synthetic datasets, and that the accuracy of machine learning models can be improved when user-specific error frequencies are taken into consideration [8, 10]. Native and non-native English speakers have different error frequencies, and also different English proficiency levels produce different error frequencies [8]. Research has also shown that this phenomenon is not exclusive to the English language, error frequency modeling has also proven to be beneficial for Russian [10].

3 ENSURING REPRODUCIBILITY

Making academic work reproducible is important to establish credibility and allow the replication of results [9]. To facilitate the reproducibility of the results of this project, machine learning-related code was structured according to the conventions of the library PyTorch Lightning (PL)[4].

PL structures raw PyTorch code according to library-defined structures. This makes elements reusable, allows for the automation of standard machine learning workflows, and also ensures reproducibility. Further, PL allows for easy deployment of machine learning systems in a SLURM cluster for example.

4 MODEL ARCHITECTURE

During this project, two different model types are to be implemented: A *tagged grammatical error corruption model* and a *grammatical error correction model*.

To work time effectively and provide an efficient solution, a reusable model architecture makes sense. Therefore, for both the *corruption* and the *correction* model, as many parts are to be re-used as possible.

The Transformer architecture has been designed for a use-case like the one present. First, our dataset contains information that is spread out throughout the input sequence. Grammatical errors might be of complex structure and stretch out over multiple sentences. The self-attention mechanism of the Transformer allows the modeling of dependencies without regard to their distance [11]. Previous work has also shown that Transformer models can learn complex structures such as error tags [8].

Figure 1 shows the architecture of the Text-To-Text-Transfer-Transformer model (T5) as proposed in the original paper:[11]:

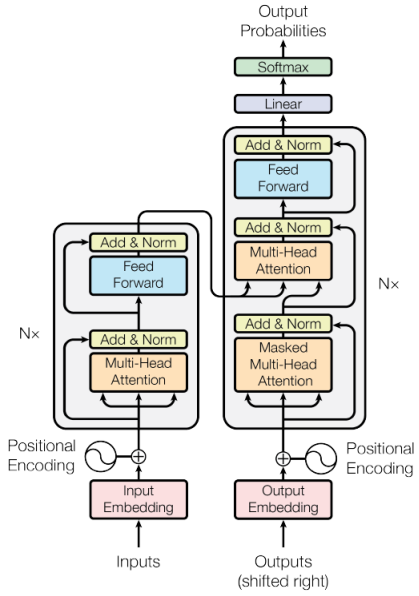

Fig. 1. The T5 Transformer model architecture [11]

The T5 architecture follows an encoder-decoder structure, as visualized in Figure 1. Both encoder and decoder use stacked self-attention and point-wise, fully connected layers. In the case of the T5 model, *N=6* layers are used for both the encoder and decoder.

While the encoder only has two sub-layers, a multi-head attention, and a fully connected feed-forward network, the decoder inserts a third layer. This additional layer performs multi-head attention over the output of the encoder [11].

Encoder or decoder-only transformer model architectures also exist, but in order to keep complexity down, the originally proposed encoder-decoder structure was used in this project.

An alternative model at this point would be a recurrent neural network (RNN) with an attention layer to still allow modeling of dependencies over any distance. But RNNs come with a downside: Their inherently sequential structure does not allow for parallelization within training examples, which becomes an important factor with longer sequence lengths [11].

Transformer models do not suffer from this issue: They can make use of highly optimized matrix multiplication code by using scaled dot-product attention [11]. This, together with other optimizations, make the Transformer an extremely scalable and high-performing machine learning model, even for longer sequence lengths [2]. New groundbreaking accuracy numbers in translation tasks have been

[4]https://lightning.ai/docs/pytorch/stable/

achieved with relatively little training time [11], which further emphasizes the potential of the Transformer architecture.

4.1 ERRANT

ERRANT is a grammatical ERRor ANnotation Toolkit, designed to compare two input sequences and classify grammatical errors according to a dataset-agnostic, rule-based framework [3]. In this project, this toolkit plays a central role during the dataset generation for the corruption model as well as the evaluation process.

Table 1 shows the different error tags supported by ERRANT.

Type	Missing	Unnecessary	Replacement
Adjective	M:ADJ	U:ADJ	R:ADJ
Adverb	M:ADV	U:ADV	R:ADV
Conjunction	M:CONJ	U:CONJ	R:CONJ
Determiner	M:DET	U:DET	R:DET
Noun	M:NOUN	U:NOUN	R:NOUN
Particle	M:PART	U:PART	R:PART
Preposition	M:PREP	U:PREP	R:PREP
Pronoun	M:PRON	U:PRON	R:PRON
Punctuation	M:PUNCT	U:PUNCT	R:PUNCT
Verb	M:VERB	U:VERB	R:VERB
Contraction	M:CONTR	U:CONTR	R:CONTR
Morphology	R:MORPH		
Orthography	R:ORTH		
Other	M:OTHER	U:OTHER	R:OTHER
Spelling	R:SPELL		
Word Order	R:WO		
Adjective Form	R:ADJ:FORM		
Noun Inflection	R:NOUN:INFL		
Noun Number	R:NOUN:NUM		
Noun Possessive	M:NOUN:POSS	U:NOUN:POSS	R:NOUN:POSS
Verb Form	M:VERB:FORM	U:VERB:FORM	R:VERB:FORM
Verb Inflection	R:VERB:INFL		
Verb Agreement	R:VERB:SVA		
Verb Tense	M:VERB:TENSE	U:VERB:TENSE	R:VERB:TENSE

Table 1. Error tags supported by ERRANT

The 54 different error tags are distributed among 24 different grammatical error categories. If applicable, each grammatical error category contains a tag for a *missing*, *unnecessary*, or *replacement* grammatical error structure.

The toolkit was chosen because it has been used in other related work [6, 8] and provides a user-friendly python package[5].

5 IMPLEMENTATION

5.1 CORRUPTION MODEL

We want to create a *tagged* grammar corruption model. This means, that we not only want a model that can convert grammatically correct sentences into corrupt sentences, but also to learn to associate error tags with specific grammatical error types.

Like all machine learning models, we first need to find or create a dataset. This dataset needs to contain sufficient information for the model to learn about the tags and related grammatical errors.

[5]https://pypi.org/project/errant/

Finding such an existing dataset has proven to be hard, so I decided to create my own. The C4_200M dataset [6] contains pairs of grammatically corrupt and correct sentence pairs, but no error tags were present. To add this essential piece of information, I used the tool ERRANT to generate error tags, representing the different error types in the sequences.

The correct/corrupt/error tags were combined into a dataset according to the structure in Table 2:

Input	Output
corrupt: [R:OTHER U:OTHER] I love swimming	I lve swim

Table 2. Dataset structure of grammatical corruption model

In the **Input** column, *corrupt:* is a prefix corresponding to our task of corrupting the input sequence. With this prefix the model can learn to differentiate between different tasks [11], in our case, we are just training for the corruption task. This means no further task prefixes will be defined.

The *error tags*, surrounded by brackets, describe the error types that are present in the corrupt sentence.

Following the error tags is the correct input sequence.

In the **Output** column of the dataset then stands the grammatically corrupt version of the input sequence.

This way, the model learns to associate the error tags with the various types of grammatical errors and should be able to transfer the learned error types to new inputs.

The sentence pair *(tags+correct, corrupt)* is tokenized using a T5Tokenizer[7] and then fed into the T5 model[8]. The implementation can be found in the corresponding repository of this paper.

A demonstration of the corruption capabilities of the model can be found below:

Input:
corrupt: [M:ADV] *There were a lot of sheep*
Output:
There were a sheep.

5.2 CORRECTION MODEL

The correction model shall be trained to be able to take an input sequence and return a grammatically corrected version of said input sequence. A dataset for this is structured as a simple tuple of *(corrupt, correct)* sequences. We can take the corruption model implementation and just change the dataset structure to represent the information needed for our correction model. Table 3 shows the dataset structure of the correction model.

[6]https://github.com/google-research-datasets/C4_200M-synthetic-dataset-for-grammatical-error-correction
[7]https://huggingface.co/docs/transformers/v4.30.0/en/model_-doc/t5#transformers.T5Tokenizer
[8]https://huggingface.co/docs/transformers/model_doc/t5

Input	Output
There were lots sheep of.	There were a lot of sheep.

Table 3. Dataset structure of grammatical correction model

A demonstration of the correction capabilities of the model can be found below:

Input:
There were a sheep
Output:
There were sheeps.

6 EVALUATION

6.1 Introduction

The above sections describe how one can first create a tagged grammatical error corruption model, use said model to corrupt arbitrary datasets according to specific error frequencies, and then train a GEC model on the resulting dataset.

But to answer the main research question, which asks for an *efficient* process, a validation and comparison mechanism needs to be developed.

A validation mechanism can be used in a process outlined in Figure 2, in which the error tag distribution in a synthetic dataset is tested and then iteratively refined until the highest-performing error distribution is found.

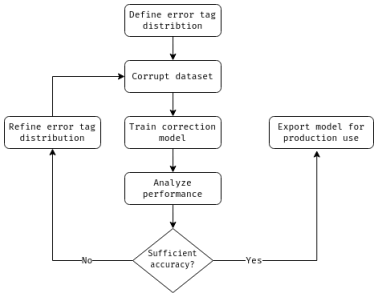

Fig. 2. Refinement workflow of the error tag distribution

It is to be emphasized that the accuracy numbers produced in the process show the correction accuracy for sequences generated by a corruption model. To gain generally comparable numbers to other GEC systems, more commonly used benchmarks like for example the CoNLL-2014 Shared Task[9] should be considered.

[9]https://www.comp.nus.edu.sg/ nlp/conll14st.html

6.2 Architecture

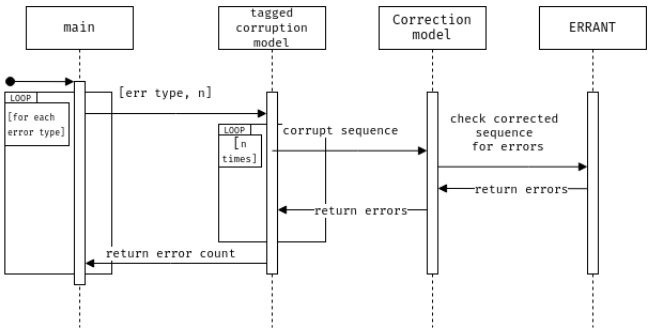

Fig. 3. Sequence diagram of evaluation process

Fig. 1 shows the architecture of the evaluation system. The list of error types gets iterated over and fed into the *corruption model*. The corruption model iteratively generates a set of *n* corrupted sequences, all with the supplied error type. The sequences are corrected using the *correction model* and then analyzed with ERRANT. By counting how many sequences have been corrected successfully an accuracy statistic is generated for each error type.

6.3 Dataset

To generate an evaluation dataset with the corruption model, first a grammatically correct corpus needs to be present. Here I used the *Wikipedia* dataset packaged by Huggingface[10], purely for simplicity reasons and the general quality of the dataset content.

Many other datasets could alternatively be used here, for example, the Common Crawl Dataset[11]. Another alternative would be to generate sequences with a Language Generation Model, like for example GPT-4[12]. With these approaches, attention to the grammatical correctness of the sequences needs to be paid.

6.4 CORRECTION ANALYSIS

So, the corruption model generates a grammatically corrupt sequence, which is in turn then fed into the correction model. Now, how do we know if the correction has been performed accurately?

As described in Figure 3, we are iterating over the various error types. This means that we know which errors are present in the corrupted sequence. The system feeds the corrupt/corrected sequence pair into ERRANT, which returns a list of the classified errors. If this list contains the error types that we are currently evaluating, we know that the correction was performed accurately.

If not all present errors have been corrected, the result is marked as invalid. This simple mechanism allows for error tag-specific correction analysis of the GEC model.

[10]https://huggingface.co/datasets/wikipedia
[11]https://commoncrawl.org/
[12]https://openai.com/gpt-4

7 EXPERIMENT

7.1 Representing a specific error frequency

Sources like [8] have already produced high-quality data for native and non-native error distributions, and as they are also working with ERRANT tags it would allow the direct reproduction of their work here. But to not just reproduce their efforts but also demonstrate the flexibility of the tag-based system, I use a different source: A research paper by Lastres-Lopez [7] has researched the different error distributions of native and non-native L1 and L2 English speakers. Their summarized results can be found in Table 4:

Metacategory of errors	Native	Non-Native	Total
Grammatical errors	21 (38.2%)	34 (61.8%)	55 (100%)
Spelling errors	19 (59.4%)	13 (40.6%)	32 (100%)
Punctuation errors	47 (53.4%)	41 (46.6%)	88 (100%)
Total number of errors	87 (49.7%)	88 (50.3%)	175 (100%)

Table 4. Frequency of errors made by native and non-native speakers, summarized per metacategory

Now, if we want to use this data to produce a GEC system for *native* English speakers, this would mean that the GEC model should show a lower overall accuracy for *grammatical errors*, but a higher sensitivity and accuracy for *spelling* and *punctuation errors*.

One quickly obvious problem is that our data source only groups errors in three meta categories, but our tag-based corruption system offers 54 different error types. If we assume that in this case we only have Table 4 as our available data, we can choose to divide the 54 tags into three categories as can be seen in Table 5.

Metacategory	Tags
Spelling Errors	R:SPELL
Punctuation Errors	M:PUNCT U:PUNCT R:PUNCT
Grammatical errors	all tags not included in the other categories

Table 5. Tag distribution between metacategories, see Table 1 for all tags

Now, we still need to choose the concrete error type distribution of our example. Grammatical errors can be of a very complex nature, so machine learning models might require a larger subset of a specific error type to achieve accuracy numbers compared to other error types. This means that we can only really do an estimation of the required tag distribution to generate a model that has a high accuracy for all error types. The here proposed mechanism allows for reproducible results and direct comparisons between different datasets. This means, that iteratively the error distribution in the dataset can be adjusted to find the one yielding the highest accuracy.

The initial distribution of the three different error tags for the *native English speaker* correction model can be seen in Table 6.

The frequency of the individual tags in the meta category *grammatical errors* has been distributed evenly.

According to this error distribution I generated a dataset of 40.000 example errors. Purposefully this is chosen to be a small dataset, to show the effectiveness of the solution.

Error type	Amount
Grammatical errors	60%)
Spelling errors	20%
Punctuation errors	20%
Total number of errors	100%

Table 6. Target distribution of metacategory of errors in synthetic dataset

As outlined in the previous sections, the resulting synthetic dataset was then used to train a grammatical error correction model and evaluated as outlined in Figure 3.

7.2 Results

The accuracy numbers as seen in Figure 4 represent the average accuracy of all tags inside the meta categories (see Table 5 for the tag distribution).

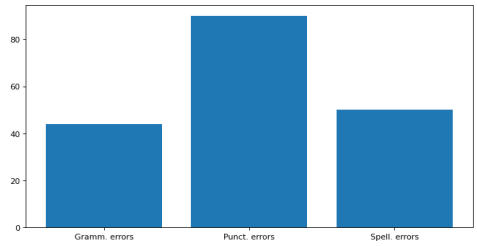

Fig. 4. Accuracy of the trained correction model

We can observe that the model has a modest overall accuracy for the general group of *grammatical errors*, a very high accuracy of *punctuation errors*, and a decent accuracy at *correcting spelling errors*.

The in Table 4 error frequency has been approximately modeled according to the evaluation results in Figure 2. Increased sensitivity towards spelling and punctuation errors can be observed, while grammatical errors have lower general accuracy.

Even though the same amount of spelling and punctuation errors were used in the training dataset (see Table 6), the correction accuracy is not equal. This can be explained by the difficulty of the underlying error type: The set of all possible permutations of punctuation errors on an arbitrary sequence is smaller than the set of all possible spelling errors. This means that the model needs fewer examples of punctuation errors in the dataset to achieve a high accuracy than for example spelling errors.

If we were to continue our experiment to find the most effective dataset for native English speaker GEC tasks, we could use the found information to refine the error tag distribution. Iteratively the most ideal tag distribution can be found.

8 DISCUSSION

The previously defined research question asks for an *efficient* process to create Transformer GEC models with arbitrary grammatical error frequencies. As observable in the Results section, a process that

creates Transformer GEC models with arbitrary error frequencies was implemented. But can it be classified as an efficient process?

According to Merriam-Webster [1], efficient can be defined as *productive of desired effects*, especially *capable of producing desired results with little or no waste (like time or materials)*.

The demonstrated results here showcase that the outlined process indeed is productive of desired effects. The process has also not produced waste of either time or materials. The corruption model was re-used for the analysis of the correction model, and can even be re-used in following projects.

But clearly, the weak point of this proposed system is the corruption model: If this model does not have the necessary accuracy, any created synthetic dataset will have low quality and the resulting correction model as well will not be able to produce high-quality corrections. Manual and automated testing is necessary here to validate the quality of the corruption model.

9 FUTURE WORK

I hope that future work can benefit from this project, the outlined approach has a lot of potential if applied correctly.

Choosing the correct size of the resulting dataset can have large implications on not only the accuracy and training time of the model, but also on the carbon footprint of the project [12]. A small but high-quality dataset can potentially deliver similar performance to a model trained on a larger dataset. This principle can be applied to generate potentially smaller datasets that would have typically been used. Future research could explore the option of incorporating more awareness of the carbon footprint in a project.

Finally, I think that using a generative language model in this context could be a very powerful extension. Next to generating highly specific error frequencies, a generative language model can also generate highly specific sequences. These could be matched to patterns in the expected target group, so that accuracy can be further improved.

10 CONCLUSIONS

In this paper, an efficient process of generating synthetic datasets with a tagged corruption model is presented. Training and evaluating the performance of GEC models trained on said synthetic datasets allows for iterative refinements of grammatical error distributions.

Therefore, the original research question, together with the two sub-questions, was successfully answered.

REFERENCES

[1] 2023. efficient. https://www.merriam-webster.com/dictionary/efficient

[2] Dimitris Alikaniotis and Vipul Raheja. 2019. The Unreasonable Effectiveness of Transformer Language Models in Grammatical Error Correction. *2019* (2019). https://doi.org/10.18653/v1/W19-4412

[3] Christopher Bryant, Mariano Felice, and Ted Briscoe. 2017. Automatic annotation of error types for grammatical error correction. *2017* (2017). https://doi.org/10.18653/v1/P17-1074

[4] Giuseppe Fenza, Mariacristina Gallo, Vincenzo Loia, Francesco Orciuoli, and Enrique Herrera-Viedma. 2021. Data set quality in Machine Learning: Consistency measure based on Group Decision Making. *2021* (2021). https://doi.org/10.1016/j.asoc.2021.107366

[5] Tao Ge, Furu Wei, and Ming Zhou. 2018. REACHING HUMAN-LEVEL PERFORMANCE IN AUTOMATIC GRAMMATICAL ERROR CORRECTION: AN EMPIRICAL STUDY. *2018* (2018). https://doi.org/10.48550/arXiv.1807.01270

[6] Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Kenneth Heafield. 2019. Neural Grammatical Error Correction Systems with Unsupervised Pre-training on Synthetic Data. *2019* (2019). https://doi.org/10.18653/v1/W19-4427

[7] Cristina Lastres-López and Gianpaolo Manalastas. 2017. Errors in L1 and L2 University Students' Writing in English: Grammar, Spelling and Punctuation. *2017* (2017).

[8] Felix Stahlberg and Shankar Kumar. 2021. Synthetic Data Generation for Grammatical Error Correction with Tagged Corruption Models. *2021* (2021). https://doi.org/10.48550/arXiv.2105.13318

[9] Victoria Stodden, Peixuan Guo, and Zhaokun Ma. 2013. Toward Reproducible Computational Research: An Empirical Analysis of Data and Code Policy Adoption by Journals. *2013* (2013). https://doi.org/10.1371/journal.pone.0067111

[10] Yujin Takahashi, Satoru Katsumata, and Mamoru Komachi. 2020. Grammatical Error Correction Using Pseudo Learner Corpus Considering Error Tendency of Learners. *2020* (2020). https://doi.org/10.18653/v1/2020.acl-srw.5

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *2017* (2017). https://doi.org/10.48550/arXiv.1706.03762

[12] Carole-Jean Wu, Ramya Raghavendra, Udit Gupta, Bilge Acun, Newsha Ardalani, Kiwan Maeng, Gloria Chang, Fiona Aga, Jinshi Huang, Charles Bai, Michael Gschwind, Anurag Gupta, Myle Ott, Anastasia Melnikov, Salvatore Candido, David Brooks, Geeta Chauhan, Benjamin Lee, Hsien-Hsin Lee, Bugra Akyildiz, Maximilian Balandat, Joe Spisak, Ravi Jain, Mike Rabbat, and Kim Hazelwood. 2022. Sustainable AI: Environmental Implications, Challenges and Opportunities. 2022 (2022). https://proceedings.mlsys.org/paper_files/paper/2022/hash/462211f67c7d858f663355eff93b745e-Abstract.html