Grammatical Error Correction at the Character Level

Adam Viola **Rushikesh Dudhat** **Shubham Shetty** **Pranjali Parse**
{aviola, rdudhat, pparse, shubhamshett}@umass.edu

1 Problem statement

Grammatical error correction (GEC) is the task of detection and correction of grammatical errors in ungrammatical text. Grammatical errors include errors such as spelling mistakes, incorrect use of articles or prepositions, subject-verb disagreement, or even poor sentence construction. GEC has become an important NLP task, with applications ranging from auto-correct for email or text editors to use as a language learning aid.

Over the past decade, neural machine translation-based approaches to GEC, which generate the grammatical version of a sentence from its ungrammatical form, have dominated the field. However, their slow inference speed, large data requirement, and poor explainability has led to research toward text-edit approaches, which make modifications to ungrammatical sentences to produce their grammatical form.

In our work, we model GEC as a two-step problem that involves edits on the character level. The first subtask is deletion, where we fine tune a Reformer, an efficient character-level transformer, to predict whether or not to delete each character. The second step is insertion, where we predict which characters to insert between existing characters to produce a grammatical sentence. These operations can be trained independently, using the longest common subsequence between the ungrammatical and grammatical sentence as an intermediary label.

2 What you proposed vs. what you accomplished

- ~~Create labels for the deletion and insertion subtasks using the Longest Common Subsequence between the ungrammatical and grammatical versions of each sentence~~
- ~~Implement models for the deletion and insertion subtasks based on the Reformer~~
- *Fine-tune the deletion and insertion models on the C4_200M, Lang-8, NUCLE, and WI-LOCNESS datasets*

  Due to constraints on our computational resources, we fine-tuned both models using the FCE training set, which contains 34,490 annotated sentences.
- ~~Perform an error analysis to determine the types of data cases where our approach~~
- ~~Maintain a repository for our code.~~

  Code associated with our project can be found at https://github.com/shubham-shetty/Reformer_GEC
- *Beat the performance of the LaserTagger baseline model*

  Our approach failed to perform well. See the results and error analysis sections for additional details.

3 Related work

Existing works commonly approach the task of grammatical error correction from two points of view – as a sequence-to-sequence task or a text-edit task.

3.1 Sequence-to-sequence

Under the sequence-to-sequence formulation, an ungrammatical sentence is given as input, and its grammatical counterpart is generated. These approaches borrow models originally created for neural machine translation (NMT) (Junczys-Dowmunt and Grundkiewicz, 2016). More recently, transformer-based (Vaswani et al., 2017) sequence-to-sequence models developed for NMT

have achieved state-of-the-art GEC performance (Grundkiewicz et al., 2019).

The literature has highlighted several difficulties that make large sequence-to-sequence models difficult to train and deploy for real-world use (Omelianchuk et al., 2020; Wiseman et al., 2018). These models suffer from slow inference speed, poor interpretability and explainability, and require a large amount of training data.

However, recent research in synthetic data generation has been shown to boost the performance of sequence-to-sequence models (Stahlberg and Kumar, 2021); these methods produce ungrammatical sentences from existing grammatical sentences (ordinary text data). Grundkiewicz et al. (2019) pretrain a large transformer exclusively on synthetic data and fine-tune it on manually-annotated ungrammatical sentences to achieve state-of-the-art GEC performance.

3.2 Text-edit

Grammatical error correction is alternatively viewed as a text-edit task, where an ungrammatical sentence is given as input, and a sequence of deletions, insertions, or other modifications are predicted to create the corresponding grammatical sentence.

Malmi et al. (2019) propose LaserTagger, a sequence tagging approach that corrects ungrammatical sentences using three edit operations. For each token in the ungrammatical sentence, LaserTagger predicts whether to (1) keep or (2) delete the token as well as whether or not to (3) insert a common phrase (from a predetermined phrase vocabulary) before the token. The authors find that predicting each tag simultaneously (keep/delete and/or insert) performs worse than assigning tags in an autoregressive, left-to-right fashion.

Stahlberg and Kumar (2020) propose Seq2Edits, a text-edit approach that predicts a sequence of span-based edit operations to modify text in a way that applies to multiple NLP tasks. Seq2Edits achieves state-of-the-art performance for the text normalization and sentence splitting tasks, and it achieves near-state-of-the-art performance for grammatical error correction.

Omelianchuk et al. (2020) propose GECToR, a sequence tagging approach that assigns token-level transformations which map each token of the ungrammatical input to its corresponding token in the grammatical target. Since GECToR predicts transformations instead of tokens, each edit is explicitly explainable. GECToR is trained using both synthetic and annotated data, and it achieves state-of-the-art performance on the CoNLL-2014 (Ng et al., 2014) and BEA-2019 (Bryant et al., 2019) benchmarks.

4 Your dataset

4.1 FCE dataset

The Cambridge Learner Corpus First Certificate in English (CLC FCE) dataset, introduced by Yannakoudakis et al. (2011), consists of a set of sentences written by English language learners at the upper-intermediate proficiency level in response to exam prompts. The texts have been manually annotated to mark errors based on approximately 80 error types.

The FCE released dataset contains anonymised texts which are annotated using XML and linked to meta-data about the question prompts, the candidates' grades, native language and age. It consists of 1244 annotated scripts, from years 2000-01, which are marked and corrected. These texts can be processed further to get original and corrected data. Statistics for the FCE dataset are detailed in Table 1.

4.2 Data Preprocessing

Since the FCE dataset is provided in an XML format, we first extract the original text and the corresponding corrected text and place the sentences into two separate files.

The core principle for our text-editing approach is based on the Longest Common Subsequence (LCS) between the ungrammatical and grammatically correct text. As the majority of text-edits are minor and usually involve a few edited characters, there is a significant overlap between each ungrammatical sentence and its corrected form; this overlap is exactly the LCS. We use the LCS as an intermediate label between the two subtasks of our approach: deletion and insertion. The deletion subtask involves deleting characters from the ungrammatical sentence that are not in the grammatical sentence, resulting in the LCS. The insertion subtask involves inserting characters into the LCS to produce the grammatical sentence. Additional details are described in the Approach section.

We first produce the labels for the deletion step. To do so, we compare each character from the ungrammatical sentence input to that of the LCS

	# Sentences	*# Correct*	*% Correct*	*# Errorful*	*% Errorful*
Train Set	30935	15045	48.63	15890	51.37
Validation Set	3555	1719	48.35	1836	51.65
Total	34490	16764	48.61	17726	51.39

Table 1: Statistics for FCE Dataset

between the ungrammatical and grammatical sentences. We label each character as 'delete' (1) or 'keep' (0), creating a tensor of length equal to that of the ungrammatical sentence.

For the insertion transformer, the LCS string is compared to the grammatically correct string. At each position in the LCS where characters could be inserted, we create a label. This label corresponds to the set of characters that need to be inserted into the LCS at that position in order to create the grammatical sentence. The label is 'None' if no characters need to be inserted. Otherwise, the value is the set of characters that need to be inserted at that position. Note that the number of insertion positions is one greater than the length of the LCS because characters can be inserted before each character and after the last character.

The deletion and insertion labels for a sample sentence are displayed in Figure 1.

5 Baselines

Our approach views GEC as a text-edit task, so we compare our approach to the text-edit approaches described in the related work section.

LaserTagger (Malmi et al., 2019) is our primary baseline because it was trained using the Low Resource track of the 2019 Building Educational Applications workshop, which contains only 4,384 training sentences. This is dataset is the closest in size to our training set of 34,490 sentences.

For informational purposes, we also present the results for Seq2Edits (Stahlberg and Kumar, 2020) and GECToR (Omelianchuk et al., 2020), which represent the state-of-the-art in text-edit grammatical error correction. It is worth noting that these models were trained on datasets totalling two million and ten million sentences, respectively.

The published work of each model describes its performance on the public CoNLL-2014 and/or BEA-2019 benchmarks, enabling direct comparisons to our approach. We evaluated our model on CoNLL dataset comprising of 1312 sentences and BEA dataset consisting of 4834 sentences.

6 Your approach

We propose a text-editing approach for GEC with two important differences from existing text-edit approaches: (1) text-edits at the character level and (2) the division of GEC into two independent sub-tasks.

6.1 Character-level edits

Sequence-to-sequence approaches to GEC treat the problem of correcting an ungrammatical sentence as a text-generation task. As a result, these sequence-to-sequence models need to both remember the input sentence and generate the corrected input in a left-to-right fashion.

This difficult formulation of the task motivates the use of word-level text-edit approaches, which can instead dedicate their capacity entirely toward predicting more interpretable word-level operations, such as insertions and deletions.

However, we observe that the most grammatical errors typically require only a small number of character-level edits to correct. For this reason, we propose to predict text-exits at the character level.

To enable character-level text edits, we extend the Reformer model, an efficient transformer with several innovations that enable computation on long input sequences: (1) locality-sensitive hashing attention in $O(L \log L)$ time and (2) reversible residual layers that reduce the memory footprint during training to only the activations of a single layer. The Reformer is publicly available on Huggingface, and there exists a version that was pretrained as a character-level language model on enwik8.

6.2 Sub-tasks

We propose to divide GEC into two independent sub-tasks performed sequentially: (1) character

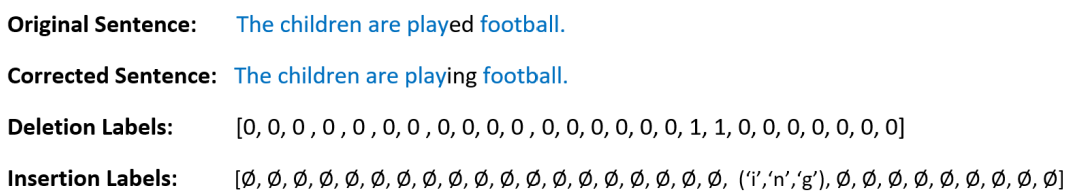

Figure 1: Labels for an example sentence. LCS is marked in blue, Ø represents None.

deletion and (2) character insertion.

The basis for this sub-task breakdown is the longest common subsequence (LCS), which given two input sentences A and B, is the longest sequence of (not necessarily contiguous) characters that are in A and B. It is worth noting the mapping from sentence A to the LCS is precisely a deletion of all characters in A that are not in B. Similarly, the mapping from the LCS to the sentence A is the insertion of all characters in A that are not in B.

Applying this concept to grammatical error correction, we can map an ungrammatical sentence to its grammatical counterpart using two learned mappings that involve the LCS between an ungrammatical sentence and its corrected form. First, we learn to delete characters from the ungrammatical sentence to produce the LCS. Then, we learn to insert characters to the LCS to produce the grammatical version of the ungrammatical sentence.

The LCS between the ungrammatical and grammatical forms of a sentence, which we can compute programmatically using dynamic programming, serves as the ground-truth for the character deletion sub-task and the input to the character insertion sub-task. Since there is no dependence between tasks, we train them independently.

At inference time, the grammatical sentence and resulting LCS are not given. To produce a prediction, we provide the ungrammatical sentence to the deletion component, which deletes all characters predicted to not be in the grammatical sentence. We provide the output as input to the insertion model, which repeatedly inserts characters to produce the corrected sentence.

6.3 Models

Since we treat the character deletion and character insertion subtasks as two independent problems, we implement and train two different models. Both the deletion and insertion models extend the Reformer, an efficient, character-level transformer.

Deletion Model Given an input sentence $X = \{x_1, ..., x_T\}$, we model the probability to delete each character $P_\theta(d_i|X)$.

To accomplish this, we remove the last softmax layer of the Reformer that maps each last layer representation to the probability of each character in the vocabulary. We replace this component with a a linear layer $l \in \mathbb{R}^d$ that maps the last layer representation to a single scalar value, where d is the size of the last layer representation. We apply the logistic/sigmoid function to map the real value to the set $[0, 1]$. This value corresponds to the probability of character deletion. More formally, given an input sentence $X = \{x_1, x_2, ..., x_T\}$ and its last layer representations $R = \{r_1, r_2, ..., r_T\}$, we estimate the probability to delete each character:

$$P_\theta(d_i|X) = \frac{1}{1 + e^{-l^T r_i}} = \sigma(l^T r_i)$$

We compute the binary cross-entropy loss on each deletion prediction and average the loss across all sequences in the batch.

Since the Reformer is traditionally trained as a left-to-right language model, we remove the self-attention mask so that all positions can attend to every other position. The implementation of the deletion model is in `code/models/deletion_reformer.py` of our GitHub repository.

Insertion Model Given an input sentence $X = \{x_1, ..., x_T\}$ of length T, we model the discrete distribution over the insertion of each character $c \in \mathcal{V}$ at each position $p \in [0, T]$ as $P_\theta(c, p|X)$.

To accomplish this, we implement the approach introduced by the Insertion Transformer (Stern et al., 2019). We first remove the last softmax layer of the Reformer that maps each last layer representation to the probability of each character in the vocabulary. We replace this component with a concatenation layer, which concatenates adjacent last layer representations. The output generated by the concatenation between adjacent inputs will correspond to a character insertion between those

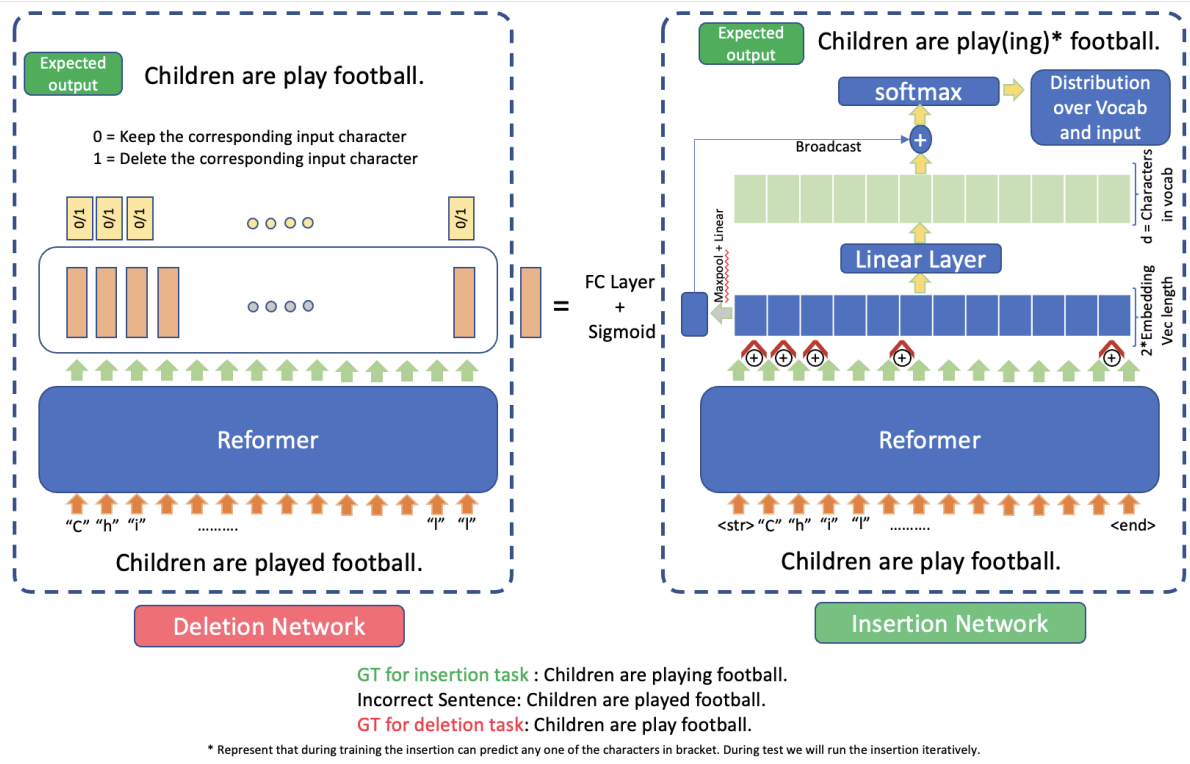

Figure 2: Insertion and deletion network architecture.

two inputs. In order to model insertions before the first character and after the last character, we start and end each sequence with new start and end tokens, respectively. The result of the concatenation layer is $T+1$ hidden representations $h \in \mathbb{R}^{2d}$, each corresponding to a unique insertion position. We denote the matrix of stacked hidden representations as $H \in \mathbb{R}^{(T+1)\times 2d}$. We then apply a linear layer with weight matrix $W \in \mathbb{R}^{2d\times|\mathcal{C}|}$ to map each hidden representation to character logits for each position.

To increase information sharing across positions, Stern et al. (2019) propose an additional bias vector $b \in \mathbb{R}^{|\mathcal{C}|}$ derived from a max pooling operation over each class of the hidden representations. The result of the max operation $g \in \mathbb{R}^{2d}$ is projected using a new weight matrix $V \in \mathbb{R}^{2d\times|\mathcal{C}|}$.

$$g = \text{maxpool(H)}$$
$$b = gV$$

Finally, we add the bias vector b to the character logits for each position and apply the softmax function across the character logits of each position to produce the distribution over the insertion of character c at each position p:

$$P_\theta(c, p|X) = \text{softmax}(HW + b)$$

For each insertion position, we compute a negative log-likelihood loss based on the set of possible characters $\mathcal{C}$ that could correctly be inserted at that position. If no character should be inserted, $\mathcal{C} = \{\text{NI}\}$; note that there exists a no-insert (NI) character in the vocabulary. The loss at each position p is defined as:

$$\frac{1}{|\mathcal{C}|}\sum_{c\in\mathcal{C}} -\log P_\theta(c, p|X)$$

Like in the case of the deletion subtask, we remove the self-attention mask so that all positions can attend to every other position. The implementation of the insertion model is in `code/models/insertion_reformer.py` of our GitHub repository.

7 Experiment

7.1 Training

Before training, we divided the FCE dataset into training and validation sets using a 9:1 split: 30,935 training examples and 3,555 validation examples.

We trained the deletion and insertion models completely independently on the FCE training set using only Google Colab. We leveraged PyTorch Lightning, a wrapper around PyTorch, to

help structure our project and simplify our training code. We evaluated each model on the validation set every half epoch, and stored checkpoints of the models with the lowest validation loss in Google Drive. We stopped training when no improvements in validation loss were made for five consecutive validation checks.

Since Google Colab was our only method for training the deletion and insertion models, we were unable to perform an extensive hyperparameter search. However, we were able to experiment with several learning rates for each model.

The best deletion model was trained for eight epochs with a learning rate of `1e-4`. The best insertion model was trained for nineteen epochs with a learning rate of `1e-3`. Both models were trained with a batch size of 32 using the Adam optimizer.

7.2 Model Variants

We experiment with two different decoding techniques to generate grammatical sentences: greedy decoding and parallel decoding.

The first technique, which we refer to as greedy decoding, is similar to traditional language model greedy decoding. After the deletion model produces its intermediate prediction, the insertion model greedily selects the next insertion by examining the character-position pair with the largest estimated probability:

$$\arg\max_{c,p} P_\theta(c, p|X)$$

The second technique, which we refer to as parallel decoding, performs a greedy decoding step for each insertion position. For each position p, the model inserts the character given by:

$$\arg\max_{c} P_\theta(c, p|X)$$

This approach enables the model to insert multiple characters at each decoding step.

8 Results

We evaluate both variants of our trained models on the CoNLL 2014 (Ng et al., 2014) and BEA 2019 (Bryant et al., 2019) test sets.

To evaluate the quality of the predicted grammatical sentences, both benchmarks employ an automated system to detect differences between the ungrammatical sentence and its predicted correction. These edits are extracted and compared against a set of annotated ground-truth edits for each sentence. These edits are evaluted using the precision, recall, and $F_{0.5}$ scores metrics. The CoNLL 2014 benchmark uses the M^2 scorer (Dahlmeier and Ng, 2012) to perform this process. The BEA 2019 benchmark uses the ERRANT scorer (Bryant et al., 2017), which offers better performance than M^2, to perform this process. The precision, recall, and $F_{0.5}$ scores for both benchmarks computed using their respective scorers are displayed in Table 2.

Both variants of our approach produce very poor results. They fail to match the performance of LaserTagger, which is the only baseline trained on a similarly small number of data examples. We consider possible sources of this poor performance in our error analysis.

9 Error analysis

9.1 Deletion errors

In Table 3, we provide examples from the validation set of the only error type experienced by the deletion model. An error of type A corresponds to the case where the deletion model failed to delete character(s) that should have been deleted. A type B error occurs when the model deletes a character that should not have been deleted. However, all errors made by the deletion model on the validation set are of type A. This indicates that under uncertainty, the model has a tendency not to delete characters.

9.2 Insertion errors

In Table 4, we show several examples of the two types of errors made by the insertion model on the validation set. An error of type C corresponds to the case where the insertion model failed to insert any characters. An error of type D occurs when the insertion module inserts characters where an insertion is not required.

Examining the validation set errors, we find that errors of type C are the most common for the insertion model; they occur in 94.9% of all validation data cases where insertion errors occurs. This indicates that under uncertainty, the model tends to not insert characters. The type D errors occur in 6.9% of validation data cases with errors. The examples in Table 4 suggest that type D errors tend to occur in cases where there are multiple letters that could be each be inserted that result in a valid English word.

Approach	CoNLL-14			BEA-19		
	P	R	$F_{0.5}$	P	R	$F_{0.5}$
LaserTagger	-	-	-	47.46	25.58	40.52
Seq2Edits (Single Model)	63.0	**45.6**	58.6	-	-	-
Seq2Edits (5 Ensemble + Rescoring)	69.9	44.4	62.7	72.7	**62.9**	70.5
GECToR (XLNet)	77.5	40.1	65.3	**79.2**	53.9	72.4
GECToR (3 Ensemble)	**78.2**	41.5	**66.5**	78.9	58.2	**73.6**
Our Approach (Parallel)	0.13	0.001	0.005	4.79	0.22	0.93
Our Approach (Greedy)	0.13	0.001	0.005	4.91	0.22	0.93

Table 2: Comparison between the baselines and our approach using precision, recall and $F_{0.5}$ scores on the CoNLL-14 and BEA-19 test sets

9.3 Sources of error

The poor performance of our approach may be attributed to several unavoidable sources of error.

First, the only character-level, transformer-based decoder or encoder (not encoder-decoder) pretrained primarily on English data that we could find is the Reformer. Huggingface does not provide any other character-level transformers pretrained primarily on English data. We explored the literature and found CharBERT (Ma et al., 2020), but CharBERT doesn't explicitly provide character-level representations that we can use for the deletion and insertion models as described in our approach.

The Reformer was pretrained as a left-to-right language model. As a result, it was trained using masked self-attention, which prevents the model from attending on any tokens to the right of the current token. The deletion and insertion subtasks do not have this attention limitation, so we removed the self-attention mask for fine-tuning. However, the use of masked self-attention during the pretraining step may have served as a source of error.

Although the Reformer was trained on the enwik8 dataset (a dump of English Wikipedia), the Reformer paper describes that the purpose of training on enwik8 was to examine the Reformer's ability to compress data. It is unclear from the paper how this purpose affects the training of the model, so this may also serve as a source of error.

Another issue is the large class imbalance in the training set. Only about 51.37% of the data cases in our training set contain errors. Among the error-ful sentences, there are typically a few deletions and insertions required to correct the sentence. Only 5.23% of labels for the deletion subtask are character deletions, and only 1.96% of labels for the insertion subtask are character insertions. As a result, there exists a large class imbalance; the deletion and insertion models are both trained to respectively delete and insert characters very infrequently. From our error analysis, it appears that our models prefer to almost always predict the majority class (no deletion or no insertion). We experimented with loss-weighting schemes to provide a larger weight to losses associated with character deletions/insertions, but these changes had no measurable effect.

There also exists a fundamental flaw with our approach. Recall that the insertion transformer is trained to predict the grammatical sentence given the longest common subsequence (LCS). If there are significant differences between the ungrammatical sentence and its corrected version, it is often the case that the LCS contains too little information to generate the grammatical sentence. Consider this example from our training set:

UG: `yours sincerely ,`
LCS: `yours faithfully ,`
G: `yours ily ,`

One way to fix this issue is to use an encoder-decoder model for the insertion subtask. The ungrammatical sentence can be provided as input to the encoder, and the LCS (or output of the deletion model) can be given as input to the decoder.

Error Type		Sentences	Notes
A1	Input	they decided to return to his father because he was old and he needed help .	's' in 'his' was supposed to be deleted from the input sentence
	GT	they decided to return to their father because he was old and he needed help .	
	Output	they decided to return to his father because he was old and he needed help .	
A2	Input	i went as a voluntier staff in order to go for free .	'i' in 'voluntier' was supposed to be deleted from the input sentence
	GT	i went as a volunteer in order to go for free .	
	Output	i went as a voluntier staff in order to go for free .	
A3	Input	on the hall of our classroom we have seen an advertisement for the london fashion and leisure show and we would like to go to the show is it possible .	First character 'o' was supposed to be deleted
	GT	in the hall outside our classroom we have seen an advertisement for the london fashion and leisure show and we would like to go to the show if is that possible .	
	Output	on the hall of our classroom we have seen an advertisement for the london fashion and leisure show and we would like to go to the show is it possible .	
A4	Input	first tell to your parents .	Characters 't' and 'o' were supposed to be deleted
	GT	first tell your parents .	
	Output	first tell to your parents .	
A5	Input	i am writting composition about `` how has modern technology changed your daily life '' , this is our question .	Extra character 't' from writting was supposed to be deleted
	GT	i am writing a composition about `` how modern technology has changed your daily life '' . this is our subject .	
	Output	i am writting composition about `` how has modern technology changed your daily life '' , this is our question .	

Table 3: Observed errors for Deletion Model

10 Contributions of group members

- **Adam**: Implemented the deletion and insertion models, set up training in Colab with checkpointing in Google Drive, trained the insertion model, computed $P/R/F_{0.5}$ scores, and wrote a lot.

- **Rushikesh**: Implemented training and evaluating code. Evaluated the trained insertion and deletion reformer on CoNLL-14 and BEA-19 dataset and compared with baselines. Created architecture diagram and other tables.

- **Shubham**: Set-up development environment, wrote code to pull and clean FCE data. Implemented label generation code for deletion step. Trained and fine-tuned the deletion model.

- **Pranjali**: Implemented insertion label generation code using the Longest Common Subsequence (LCS) strategy. Wrote data loader functions for model training and evaluation. Performed error analysis.

11 Conclusion

In this paper, we propose a novel approach to grammatical error correction that operates on the character level. Overall, our approach failed to produce a successful system for grammatical error correction. We found that even with clean data and a novel idea, it can be surprisingly difficult to beat the performance of existing models.

11.1 Future work

We still feel that a character-level GEC model should be able to provide good results, but with significant changes from the approach we followed. Some potential ideas which could be explored include -

- *Experiment with other character-level transformers:* We chose to use the Reformer model in our project because it was the only character-level encoder or decoder (not encoder-decoder) available pretrained primarily on an English dataset. As stated in the sources of error section, the Reformer may have been a significant reason for our poor

Error Types		Sentences	Notes
C1	Input	what i have to pay by myself ?	'do' was supposed to be inserted. No insertion observed in LCS.
	LCS	what i have to pay myself ?	
	Output	what i have to pay myself ?	
C2	Input	i went as a voluntier staff in order to go for free .	'e' was supposed to be inserted in word 'volunter'. No insertion observed in LCS.
	LCS	i went as a volunter in order to go for free .	
	Output	i went as a volunter in order to go for free .	
D1	Input	what other i would like ask is that why it is said this is the london 's newest and best musical show .	'w' & 't' inserted instead of 't' & 'e' in LCS. Wrong characters inserted.
	LCS	h other i would like ask is why it is said this is london 's newest and best musical show .	
	Output	wht other i would like ask is why it is said this is london 's newest and best musical show .	
D2	Input	fashion became a part of our life and some people think that what clothes you wear is most important thing .	'o' inserted instead of 'a' in becme. Wrong character inserted.
	LCS	fashion becme a part of our life and some people think that what clothes you wear is most important thing .	
	Output	fashion become a part of our life and some people think that what clothes you wear is most important thing .	

Table 4: Observed errors for Insertion Model

performance, and we may find more success with insertion using an encoder-decoder model pretrained on English data.

- *Train model with augmented synthetic data:* The C4_200M Synthetic Dataset for Grammatical Error Correction (Stahlberg and Kumar, 2021) has been recently been released which can be used to generate a large volume of synthetically generated ungrammatical sentences. Although this process would be computationally intensive and the synthetic data may not exactly reflect actual real-world common grammatical errors, the resulting pretrained model could provide a better starting point for deletion and insertion fine tuning.

- *Try a tagging & pointing model:* As one of the causes for error was that grammatical sentence and LCS may not have significant overlap, a tagging and pointing model can be used instead of the deletion and insertion model we proposed. The tagging and pointing based model is explored in FELIX (Mallinson et al., 2020). Pointing allows FELIX to reorder word inputs instead of deleting spans of text. A similar approach can be tried out at the character level.

References

Bryant, C., Felice, M., Andersen, Ø. E., and Briscoe, T. (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.

Bryant, C., Felice, M., and Briscoe, E. (2017). Automatic annotation and evaluation of error types for grammatical error correction. Association for Computational Linguistics.

Dahlmeier, D. and Ng, H. T. (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.

Grundkiewicz, R., Junczys-Dowmunt, M., and Heafield, K. (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.

Junczys-Dowmunt, M. and Grundkiewicz, R. (2016). Phrase-based machine translation is state-of-the-art for automatic grammatical error correction. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1546–1556, Austin, Texas. Association for Computational Linguistics.

Ma, W., Cui, Y., Si, C., Liu, T., Wang, S., and Hu, G. (2020). Charbert: Character-aware pre-trained language model. *arXiv preprint arXiv:2011.01513*.

Mallinson, J., Severyn, A., Malmi, E., and Garrido, G. (2020). FELIX: Flexible text editing through tagging and

insertion. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1244–1255, Online. Association for Computational Linguistics.

Malmi, E., Krause, S., Rothe, S., Mirylenka, D., and Severyn, A. (2019). Encode, tag, realize: High-precision text editing. *arXiv preprint arXiv:1909.01187*.

Ng, H. T., Wu, S. M., Briscoe, T., Hadiwinoto, C., Susanto, R. H., and Bryant, C. (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.

Omelianchuk, K., Atrasevych, V., Chernodub, A., and Skurzhanskyi, O. (2020). Gector–grammatical error correction: Tag, not rewrite. *arXiv preprint arXiv:2005.12592*.

Stahlberg, F. and Kumar, S. (2020). Seq2edits: Sequence transduction using span-level edit operations. *arXiv preprint arXiv:2009.11136*.

Stahlberg, F. and Kumar, S. (2021). Synthetic data generation for grammatical error correction with tagged corruption models. In *Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 37–47, Online. Association for Computational Linguistics.

Stern, M., Chan, W., Kiros, J., and Uszkoreit, J. (2019). Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pages 5976–5985. PMLR.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Wiseman, S., Shieber, S. M., and Rush, A. M. (2018). Learning neural templates for text generation. *arXiv preprint arXiv:1808.10122*.

Yannakoudakis, H., Briscoe, T., and Medlock, B. (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.