Grammatical Error Correction with Transformer Models

Sahith Vavilala, Syed Fuzail Ali & Sumit Raut
Donald Bren School of Information and Computer Sciences
University of California - Irvine
Irvine, CA 92697, USA
{svavila1,sfali1,scraut}@uci.edu

Abstract

Text correction is a very important task within natural language processing (NLP), with applications in grammar correction, paraphrasing, and style refinement. Within this paper, we evaluated multiple different transformer-based architectures, which include the pre-trained T5 model, a pre-trained GPT model, and a custom-built transformer model using the CoEdit dataset. We assess their performance across different text correction tasks (mainly grammar correction), comparing the effectiveness of each model to generate accurate outputs. Our results will highlight the strengths and limitations of each model, which will provide further insights into the best tasks to provide to each model. These findings can contribute to the vast ongoing research involving text correction and shows the efficacy of non pre-trained custom transformer models that can be built by anyone.

1 Introduction

This project works upon trying to leverage transformer-based models to be able to complete grammatical error correction. This will be done using multiple different transformer-based models such as Google's T5 model, OpenAI's GPT2 model, and a custom-built transformer (built in PyTorch). We use T5 tokenizers for the T5 model and the custom-built transformer model and GPT2 tokenizer for the GPT2 model. We plan to see how each model performs and see if we can create a custom model that can provide results similar to the pretrained models. The CoEdit dataset provided by grammarly is used for all of the models and multiple metrics are used to identify what tasks certain models are good at versus not good at.

1.1 Why is this important?

This project is important because grammatical error correction is something that is widely used in the real-world. We can see this in applications used my millions of people such as Grammarly, Gmail, and many other services. Live-time grammatical error correction has widely benefited many people such as students, professionals, and non-native speakers of languages. By evaluating multiple transformer-based models (T5, GPT, custom) on the CoEdit dataset, we can get valuable insights into the strengths and limitations of each model specifically within text correction. Unlike other types of models, transformer models are able to provide context-aware text corrections (due to attention heads) which make them much more effective in real-world applications like AI-powered writing assistants. This project also contributed to NLP studies as it compares multiple different architectures (models) and fine-tuning strategies, which can help to offer insights on model optimizations and computational efficiency.

1.2 Proposed Approach

To solve this problem, we propose to train multiple transformer models as they utilize attention heads to help capture longer-range dependencies and contextual relationships between words in a given text. This is very useful for text correction tasks, where understanding the overall context of a sentence is essential for making accurate grammatical or stylistic changes.

Our approach involves evaluating and comparing three transformer architectures. The first being the T5 transformer model which is a pre-trained model which handles all natural language processing tasks as text-to-text problems (Hugging Face, 2025). We fine tuned the T5 transformer model on the CoEdit and part of the C4 200M datasets. The second model is the GPT transformer model which will learn to generate corrected versions of sentences of inputted sentences on word at a time. It will work by predicting the next word of a sentence by referencing the words that it has already generated. The final model is the custom transformer, which we developed from scratch. This custom model is implemented using PyTorch and using the ideas from the ”Attention is All You Need” paper by Google. By creating this model from scratch, we are able to control all of the architectural choices. This extends from the amount of layers to be used, the number of attention heads, embedding dimension, sequence length and more. Both the GPT model and the custom transformer model are trained and tuned with only the CoEdit dataset.

To evaluate these models, we observe the training and validation losses. We also use a wide range of metrics such as BLEU, ROUGE, METEOR, and BERT scores. Each score is better at evaluating different tasks, which will be explained later.

2 Dataset Description

2.1 CoEdit Dataset Description

In this project, we used the CoEdit dataset which is provided by grammarly, which we found on both HuggingFace [1] and Kaggle [2]. This dataset consistens of real-world, human-edited English text that covers a wide range of example corrections. It is one of the most robust datasets for training grammar correction models. The dataset consists of two files: validation.csv and train.csv. Each of the files consist of three columns: task, src, and tgt. The task column has a single tag (word) that basically describes the fix that should be made. These can vary from gec (grammatical error correction) to paraphrase. The src column contains the original (incorrect) text that should be fixed according to the task. Each data point will start with a prompt like ”Remove all grammatical errors from this text:”, and the text to be considered will be placed after it. The models will ignore this part of the text and will only fix the sentence which is shown after it. The tgt column consists of the correct output. Although, in some instances there can be multiple correct outputs (such as in paraphrasing). The training dataset contains approximately 69,071 examples and the validation dataset contains 1712 examples.

2.2 C4 200M Description

The C4 200M Grammar Error Correction [3] dataset is a synthetic dataset created by Google for the purpose of grammar error correction. Google created this dataset because they believed that one of the main problems with grammar error correction is that there is a sparse amount of data, meaning that they believed that there was not enough data available to train models for this task (Stahlberg & Kumar, 2021). So, they created this dataset using a new method called tagged corruption models. Basically, this method would begin with grammatically correct sentences and then used ’corruption’ to add errors to the grammatically correct sentences. By using this, they were able to create errors that were more consistent with errors that actual people may make when writing.

The total dataset has approximately one-hundred eighty-five million examples split across many files (for the sake of file size). Overall, the dataset is quite simple. There are two fields which are input and output. The input has the corrupted (incorrect) sentence to be corrected and the output contains the corrected version of the sentence. Part of this dataset was only used in the T5 model (to be explained and shown later) to expand the total dataset size. It was planned to be used in the

[1]HuggingFace Dataset Link: https://huggingface.co/datasets/grammarly/coedit

[2]Kaggle Dataset Link: https://www.kaggle.com/datasets/thedevastator/coedit-text-editing-dataset

[3]C4 200M Kaggle Link: https://www.kaggle.com/datasets/dariocioni/c4200m

other models as well, but due to the overall training time of the transformer models with such a large amount of data, it was only kept to the pretrained T5 model.

2.3 Preprocessing of Data

In terms of preprocessing of data, there were no extreme measures that needed to be taken. We simply loaded the tokenizer that was to be used for each specific model (T5 tokenizer for T5 and custom models, GPT2 tokenizer for GPT2 model). The tokenizer is used to convert the text into token IDs which can be understood by the models at hand. We then implemented a preprocessing function that goes through each example within both the training set and validation set and then converts the source and target columns into the tokenized form. The tokenized form for bothe the source and target columns are the same. It consists of making the max length of token sequences are at most 128 tokens long, makes sure the padding is equal to the max length (meaning that if a sequence is shorter than the max length, then it would be padded to 128 tokens long with padding tokens), truncates sequences that are longer than 128 tokens long to 128 tokens, and adds special tokens such as start and end tokens. The tokenized sequences are then converted to PyTorch Tensors. This is to enable the use of CUDA with Nvidia's GPU for faster training and validation times. The preprocessing function will then return the inputs (source text), attention mask (which indicates whether tokens are actual words or padding), and labels (target text) as tensors.

3 Related Works

Grammar Error Correction (GEC) has evolved significantly over the decades, progressing through various approaches. Early rule-based methods relied on handcrafted linguistic rules and parse trees but struggled with unseen errors. Classifier-based models, popular in the early 2000s, trained discriminative classifiers like SVMs and Naïve Bayes using contextual features but were limited to predefined error types. Statistical Machine Translation (SMT) models later emerged, leveraging parallel corpora to learn error corrections but relied heavily on memorized patterns. Neural Machine Translation (NMT) models improved upon SMT by capturing structural patterns and handling unseen errors better. However, NMT-based GEC still faces challenges like limited annotated data and high lexical similarity between input and output, leading to ongoing research in enhancing these models (Mina Naghshnejad, 2020).

Recent advancements in transformer-based models have significantly improved GEC performance. Models like COEDIT by Grammarly demonstrated substantial gains in error correction and broader language tasks. Large Language Models (LLMs), such as GPT and T5, have further advanced GEC by leveraging large-scale pretraining on diverse text corpora, enabling them to generalize across a wider range of grammatical errors. LLMs benefit from in-context learning, allowing them to correct errors more effectively without extensive task-specific fine-tuning. Their ability to model long-range dependencies and syntactic structures has led to state-of-the-art performance in GEC, making them the dominant paradigm in recent research.

4 Project Approach

4.1 T5-Small Transformer

We fine-tune a T5-small model to perform multiple sequence-to-sequence tasks, including Grammar Error Correction (GEC), neutralization, simplification, paraphrasing, coherence, and clarity enhancement. The model is trained using the Hugging Face Trainer API with mixed precision (fp16) for efficiency. The T5Tokenizer is used to tokenize input and target sequences, with a maximum token length of 128 for both. The training dataset is a combined corpus from Grammarly's CoEdit and Google's C4-200M, resulting in approximately 150,000 sentence pairs. The training configuration includes a batch size of 8, learning rate of 5e-5, 5 training epochs, gradient accumulation steps of 16, and evaluation every 500 steps. Optimization is handled by the AdamW optimizer, combined with a linear learning rate scheduler to dynamically adjust learning rates. To mitigate overfitting, label smoothing loss is applied. By leveraging a diverse dataset and structured training, the model effectively enhances text fluency, coherence, and grammatical accuracy across multiple language refinement tasks.

4.2 GPT-2 Transformer

The GPT-2 transformer model leverages the same Hugging Face dataset libraries to train the transformer and set up the GPT-2 tokenizer in the preprocessing phase. Specifically, the tokenizer setup involved defining end-of-sequence, beginning-of-sequence, and padding tokens to ensure consistency during batch processing. This allows the GPT-2 transformer model to detect tokens more accurately. That said, many users in various forums online, including the Hugging Face forum (Chand, 2021), have pointed out that GPT-2 tends to struggle with tasks that involve labeling or classifying individual words or tokens. This calls for unintended behavior which is discussed in the results section regarding the GPT-2 model's ability to correct grammar correctly consistently.

The GPT-2 transformer model is initialized using the GPT2LMHeadModel class from Hugging Face and comprises approximately 124 million parameters arranged in a decoder-only transformer architecture. The model utilizes CUDA on Kaggle's T4 GPUs to maximize compute power and efficiency. It employs the AdamW Optimizer with a learning rate of $5e^{-5}$ and weight decay of 0.01 for regularization across 5 epochs. During the forward pass, the model computes the output and the cross-entropy loss within an $autocast()$ context block using mixed precision to help speed up the training since the epochs prior took an hour each. The backward pass is scaled and handled using mixed precision to compute the gradients without losing accuracy. Before updating the model, the gradients are unscaled and clipped to avoid the exploding gradient issue. The updates are then finalized via the AdamW optimizer and trigger a checkpoint to save the model at the end of each epoch.

4.3 Custom Transformer

Our custom transformer model was created from scratch using PyTorch, creating separate classes for each block of the model which when combined together create the full transformer model. We use the T5 tokenizer for the vocabulary set for this model.

4.3.1 Embedding Layer

The first block of the model is the Embedding Layer. The embedding layer's role is to transform the input token IDs into vector representations that can help capture semantic relationships between words. With a vocabulary size V and an embedding dimension of d, we can initialize an embedding matrix E with V rows and d columns that are filled with numbers. This enables the model to represent words in a matrix where words with similar meanings are nearby one another. This will help the model to understand relationships between words.

4.3.2 Positional Encoding Layer

The positional encoding layer provides (as the name suggests) helps to provide positional information of words. This works by creating a matrix of positional patterns where each row will represent a position in a sequence, going up to the maximum sequence length. Each position in the matrix will receive a unique combination of sin and cosine values across the embedding dimension. So, when the model receives an input sentence, these positional values will provide a representation of where exactly these words belong in the sentence. This is critical for transformer models which view the whole sentence at once, so this will allow for the model to understand position of words. It is also critical for the grammar error correction task where the order of words is extremely important.

4.3.3 Multi-Head Attention

Multi-head attention is one of the most important, if not the most important part of the transformer model. This allows for the transformer model to focus on multiple parts of the sentence at the same time, which allows for the model to capture relationships between words regardless of where exactly they lie in the input sentence. The model takes the input embedding and multiplies each embedding using a linear layer to turn it into three vectors: query, key, and value vectors. These vectors are split into multiple heads that help the model to learn different types of relationships between words. A score is then calculated, called the attention score (shown in the equation below). This score (a value between 0 and 1) will provide a score for each word to every other word. The higher the score, the more related one word is to another (in the eyes of the model). The heads are then concatenated together and put through a linear layer where the final attention output is provided.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_h}}\right)V$$

4.3.4 Transformer Encoder Block

Next, is the encoder module (block) where it learns the relationships between words using the multi-head attention. It then passes the information through a feedforward neural network consisting of two layer normalization layers, two dropout layers (dropout = 0.2), with ReLU activation. The forward pass of the encoder module is shown in the equations below.

$$\begin{aligned}\text{AttentionOut} &= \text{MultiHead}(Q, K, V)\\ x_1 &= \text{LayerNorm}(Q + \text{AttentionOut})\\ x_2 &= \text{LayerNorm}(x_1 + \text{FFN}(x_1))\\ \text{Output} &= \text{Dropout}(x_2)\end{aligned}$$

4.3.5 Transformer Encoder

This block completes the entire encoder for the transformer. This will take the sequence of input token IDs and will process them through multiple layers including, embedding, positional encoding, and attention blocks. Overall, there is not anything new added within this block. It brings together previous blocks described above to complete the logic of the encoding mechanism.

4.3.6 Transformer Decoder Block

This is a single decoder layer in the transformer model. The role of this block is to generate corrected output text word by word. It does this attending to both the decoder's previous outputs (in a method called self-attention) and from the encoded input provided from the encoder. It first does self-attention with a mask, meaning that it can only see its current input and not anything further than that, which means that it can only learn from what it has outputted itself so far. Then the output of the attention is added to the original input and is then normalized and further regularized with a dropout layer. The decoder then goes to the encoder output (which is the key, query, and value vectors) using the value vector after the dropout step.

$$\begin{aligned}\text{SelfAttnOut} &= \text{MultiHead}(x, x, x) \quad \text{(masked)}\\ x_1 &= \text{LayerNorm}(x + \text{SelfAttnOut})\\ \text{CrossAttnOut} &= \text{TransformerModule}(K, Q, x_1)\\ \text{Output} &= \text{CrossAttnOut}\end{aligned}$$

4.3.7 Transformer Decoder

The transformer decoder is similar to the transformer encoder from earlier, where previous logical blocks are connected together to create the total decoder logic. This will take in the partially generated output and will pass it through multiple decoder layers (specified later). It uses word embedding which will convert token IDs from the target vocabulary set into vectors. Then it will apply positional encoding to that. Then, it will pass through a sequence of decoder blocks where self-attention will be used as described previously. Finally, the inputs will be passed through a output linear layer where the output of the decoder will be mapped to the size of the target vocabulary. This is used to produce probabilities during inference (with softmax function) during the inference step.

4.3.8 Final Transformer Model

The final transformer class combines the encoder and decoder modules to create a full encoder-decoder architecture, which will be used for the grammar error correction task. The most important part of this block comes in the inference (decoding) phase, where multiple methods are used such as temperature scaling, top-k sampling, and top-p sampling. Temperature scaling allows for us to control how confident the model's predictions are when generating the next word. If the value is less than 1, the predictions may become more random while if it is greater than 1, the predictions become more deterministic (choosing most likely token). Top-k sampling will limit the model to the top k (some real value) tokens. For example, if k was equal to 5, it would limit the model to choose from the top 5 most likely tokens from the probabilities provided. Top-p sampling depends on some value p (between 0 and 1) which is seen as a probability. It will sort the tokens by probability and choose the top tokens until their probabilities equal to p, it will then ignore all other tokens after. Our model uses a combination of these methods. After the decoding step, the model will return the output labels.

4.3.9 Parameters

We chose the embedding dimension to be equal to 128, the initial and target vocab size equal to 32000 (from T5 tokenizer), there to be 6 layers, sequence length to be 128, expansion factor to be equal to 4, and 8 attention heads.

5 Project Results - Evaluation

5.1 Metrics Used

We used multiple metrics for this project to evaluate the model performance when testing: BLEU, ROUGE, METEOR, BERT. BLEU score measures the precision of n-grams (phrases of one or more words) in the predicted text when compared to the reference text. The BLEU score is seen to be better for machine translation. ROUGE score measures the recall of n-grams in the predicted text when compared to the reference text. There are subsets of ROUGE scores such as ROUGE-1 (overlap of single words), ROUGE-2 (overlap of two-word sequences), ROUGE-L (measures longest common subsequence). ROUGE is seen to be better for summarization and paraphrasing. METEOR score measures the precision, recall, synonyms, and word order. It is seen to better for grammar and paraphrasing. BERT measures its score based on the semantic similarity of sentences, meaning that it bases its score on how close the meaning of two sentences are. It is seen to be best for meanings of sentences.

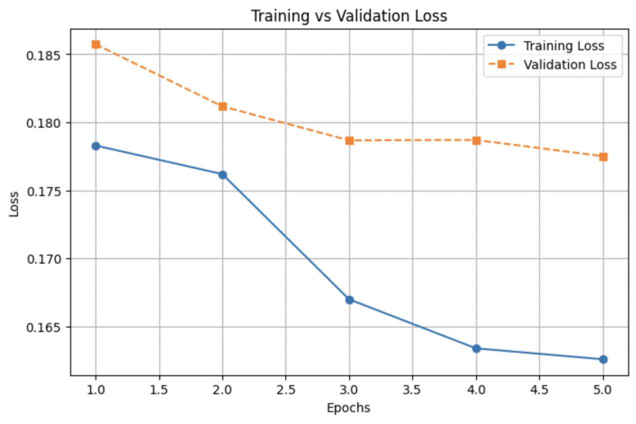

Figure 1: Training and Validation Loss for T5 Model.

5.2 Evaluation: T5-Small Transformer

The model was evaluated based on both loss trends and performance across six tasks using BLEU and ROUGE metrics. The training loss consistently decreased over five epochs, indicating stable learning. Meanwhile, the validation loss initially declined but plateaued after the third epoch, with only a slight reduction, suggesting that further gains might require additional fine-tuning or data augmentation.

For task-specific evaluations, the model performed exceptionally well on coherence (BLEU: 0.8882, ROUGE1: 0.9576) and neutralization (BLEU: 0.8587, ROUGE1: 0.9334), demonstrating strong text understanding and transformation capabilities. The clarity and GEC tasks also achieved high BLEU scores (0.7596 and 0.7175, respectively), reflecting effective grammar correction and readability improvements. However, paraphrasing (BLEU: 0.1496) and simplification (BLEU: 0.2808) showed weaker performance, likely due to the complexity of rephrasing while preserving meaning. Overall, the model demonstrated strong generalization on structured tasks but may benefit from additional fine-tuning for tasks requiring diverse rewording strategies.

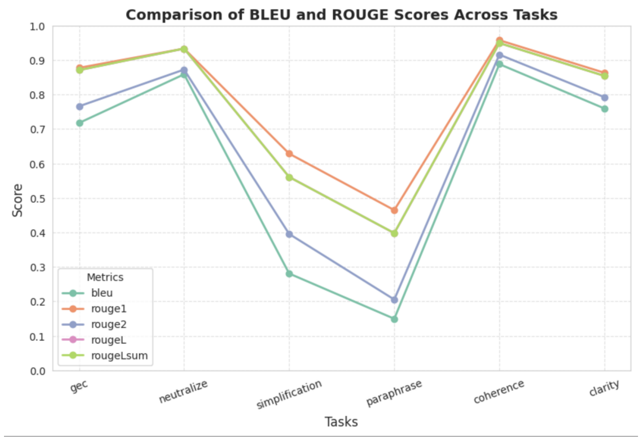

Figure 2: Metric Scores by Task For T5 Model.

5.3 Evaluation: GPT-2 Transformer

The GPT-2 Model was evaluated using both training and validation metrics. As shown in Figure 3, after training and validation error over 5 epochs, the loss is seen to consistently decrease from around 6.2 to 4.2. The model's inability to generalize patterns is seen with the poor loss regardless of its decreasing nature. For task-specific evaluations, the GPT2 Model measured (BLEU: 0.22, ROUGE-1: 0.2, ROUGE-2: 0.06, ROUGE-L: 0.2). Overall, these results indicate the GPT2 model's inability to detect tokens as precisely as the other models, and suggests further fine-tuning and preprocessing of the data.

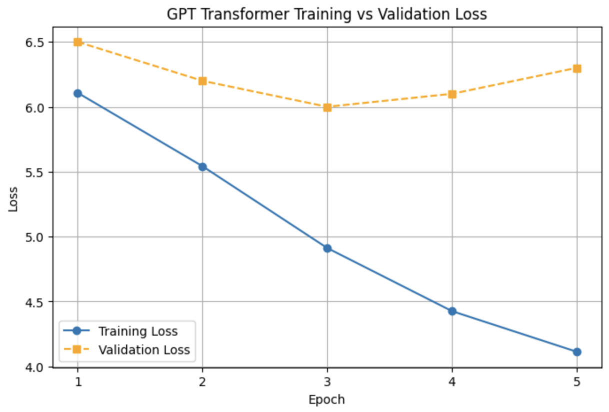

Figure 3: GPT-2 Training and Validation Loss Over 5 Epochs.

5.4 Evaluation: Custom Transformer

The model was evaluated based on both loss trends and performance using BLEU, ROUGE, METEOR, and BERT metrics. The training loss consistently decreased over 10 epochs, indicating stable learning. However, the validation loss declined for about 6 epochs before increasing again which indicates that the custom transformer started to overfit to the training data.

When evaluated using the testing data, we achieve a BLEU score of 0.66, a ROUGE-1 score of 0.83, a ROUGE-2 score of 0.73, a ROUGE-L score of 0.83, a METEOR score of 0.84, and BERT score of 0.8139. For the custom transformer model, these scores are quite good (although they are not as high as the T5 model scores).

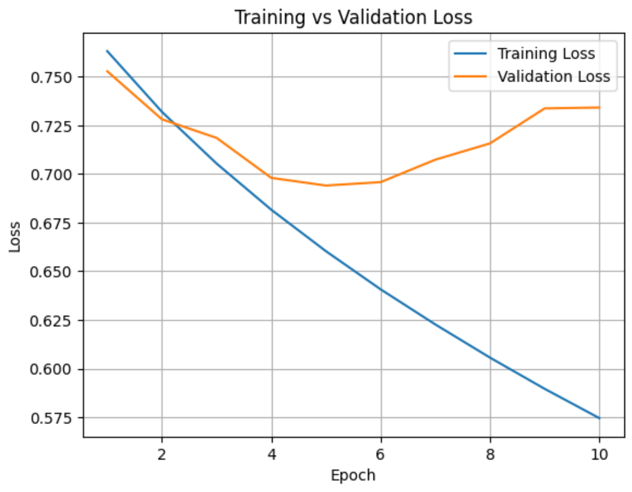

Figure 4: Training and Validation Loss for Custom Transformer.

6 Conclusion/Discussion

Overall, as we can see the pre-trained T5 model performed the best as it was trained on the most amount of data and is already pre-trained on a vast amount of data making it more versatile than the custom transformer. However, the custom transformer did not perform too far behind as it was able to score decently high on the metrics listed above. The GPT-2 model did perform the worst in terms of loss and metrics when compared to the other two models. This is because the GPT-2 model is not as good at detecting tokens, especially when compared to the T5 model. We were able to show that a custom transformer can perform somewhat on par with state of the art pre-trained models.

In terms of future work, we can extend the C4 200M dataset to all of the models to observe if that increases the functionality of the grammar error correction of the GPT-2 and custom transformer models. We also would want to look into increasing the functionality of the models when doing other tasks, such as paraphrasing or simplification. Another avenue for future work would be to train our own tokenizer based off of the datasets used to see if that would increase the accuracy of grammar correction in relation to these datasets. Finally, we would want to extend inputting custom inputs into the custom transformer for grammar correction.

7 Acknowledgments

All members of this group (Sahith, Syed, Sumit) contributed to all facets of this project. We each came together and worked on pre-processing the data, research, creating the custom transformer, training pre-trained models, slides for presentation, and the project report as a group. Each one of us ran a version of each of the models locally to get the best outputs. There was no unequal distribution of work and everyone committed valuable time and effort to this project. Everyone was always on the same page and was committed to delivering their part of the project.

REFERENCES

Vineeth Chand. Fine-tuned gpt2 model performs very poorly on token classification task. https://discuss.huggingface.co/t/fine-tuned-gpt2-model-performs-very-poorly-on-token-classification-task/14271/1, 2021. Accessed: 2025-03-20.

Hugging Face. T5 model documentation. https://huggingface.co/docs/transformers/en/model_doc/t5, 2025. Accessed: 2025-03-20.

Vijayan N. Nair Mina Naghshnejad, Tarun Joshi. Recent trends in the use of deep learning models for grammar error handling. September 2020. doi: https://doi.org/10.48550/arXiv.2009.02358. URL https://doi.org/10.48550/arXiv.2009.02358.

Felix Stahlberg and Shankar Kumar. The c4 200m synthetic dataset for grammatical error correction. August 2021. URL https://research.google/blog/the-c4_200m-synthetic-dataset-for-grammatical-error-correction/.