Faculty of Natural and Mathematical Sciences
Department of Informatics

Bush House, King's College
London, Strand Campus, 30
Aldwych, London WC2B 4BG
Telephone 020 7848 2145
Fax 020 7848 2851

7CCSMPRJ

Individual Project Submission 2023/24

Name: Shivang Chaudhary

Student Number: K23037541

Degree Programme: MSc Artificial Intelligence

Project Title: Simple methodology for building an effective GEC

Supervisor: Dr. Zheng Yuan

Word count: 13,780

RELEASE OF PROJECT

Following the submission of your project, the Department would like to make it publicly available via the library electronic resources. You will retain copyright of the project.

☒ I **agree** to the release of my project

☐ I **do not** agree to the release of my project

Signature: 

Date: 20th July 2024

Department of Informatics
King's College London
United Kingdom

7CCSMPRJ Individual Project

Simple methodology for building an effective GEC

Name: Shivang Chaudhary

Student Number: K23037541

Degree Programme: MSc Artificial Intelligence

Supervisor: Dr. Zheng Yuan

This dissertation is submitted for the degree of MSc in Artificial Intelligence.

Acknowledgements

I would like to express my sincere gratitude to all those who contributed to the successful completion of this project to test a grammatical error correction (GEC) system on the English datasets and make the required experiments possible.

Firstly, I would like to thank my project supervisor, Dr. Zheng Yuan, for her invaluable guidance, support, and encouragement throughout the duration of this project. Their insightful feedback and constructive criticism were instrumental in refining my approach and enhancing the quality of this work.

I am extremely grateful to the supervisor and the King's College London research department for providing the necessary resources and infrastructure to conduct this research. Access to computational facilities and relevant literature was crucial for the project's successful execution. The respective institution provided the necessary computational resources, such as a high-power computing resource, 'CREATE', so I could carry out successful testing and analysis for the project. I would also like to thank the creators of online software 'Overleaf' due to which the successful completion of this sophisticated LaTeX report was possible [3].

Lastly, I would also like to thank the creators and maintainers of the English datasets, CoNLL-2014 and lang-8, due to which results were made possible in this project. Their dedication to curating and providing high-quality linguistic data enabled a comprehensive evaluation of the GEC system.

Abstract

The project is focused on the purpose of providing a methodology that is simpler to apply and the purpose for the whole project can be understood in reasons which are essential to produce the grammatical error correction or GEC model in the paper, which will be called 'gt5' that aims to improve the accuracy of grammar detection by reducing the steps of fine-tuning thus allowing the ease in production of algorithm and its application. The model is expected to reduce the steps of fine-tuning due to its nature of being an optimised version of the base model and utilising the appropriate datasets. The clang-8 dataset [5] is a cleaner version of the noisy lang-8 dataset, and therefore, the focus relies greatly on generating the dataset for the application of the model, followed by the experiment on the CoNLL-2014 dataset [12] for its popularity among model's benchmark comparisons. Next, the model gt5, whose base model is a T5 (pre-trained) model, will be produced so that the results can be compared to the performances of relevant GEC systems made in the past. The benchmark is expected on language English for this paper. The base-model (t5-base) consists of consists of 220M parameters. Future experiments can be done on rest of the languages given the data is sufficient for each language. Through the mentioned stages of processing the algorithm, the aim converges at successfully formulating an easy but efficient method to build a GEC model that can further contribute to the field as a capable baseline model.

Nomenclature

This section serves as a reference list for the key terms, abbreviations, metrics, and models used in this NLP project. The relevant sections of the document provide detailed explanations for the mentioned concepts.

Mathematical annotations

$5e, 2e$ Learning rate represented scientifically as 0.00005, 0.00002

η Learning rate.

λ Regularization parameter.

∇L Gradient of the loss function.

B Batch size

i Index.

L' Derivate of loss function.

w_t Weight at time 't'

x_i Input data at index 'i'

y_i Output data at index 'i'

Code and functions

AdamW An optimizer that implements the Adam algorithm with weight decay.

Auto_tokenizer Tool to automatically load the appropriate tokenizer for a given pre-trained model without specifying the exact tokenizer class.

DataCollatorWithPadding A data collator that pads batches of data to the same length.

gec_pipeline The pipeline used for GEC tasks.

Gecmodel.py Main file for running the GEC model.

Gecmodel_errEval.py File for evaluating the results of GEC model.

introduce_errors Function to implement data augmentation.

pred_texts Variable to store generated sentences by the GEC.

preprocess_function Function to preprocess data before training.

run_prog Named virtual environment on HPC for training tasks.

subprocess A module to spawn new processes, connect to their input, output or error pipes, and obtain their return codes.

trainer.train() Method to begin the training loop in Hugging Face's Trainer class.

virtualenv Tool to create isolated Python environments.

Definitions

FN True negative.

FP False positive.

GEC A subfield of NLP that aims to correct grammatical errors in text.

HPC High-performance computing

MLM Stands for Masked Language Model, used in training models like BERT where the model learns to predict the words that are masked in the input.

NLP Natural Language Processing, the field of artificial intelligence focused on training machines to understand human language.

TN True negative.

TP True positive.

$C4$ Large-scale dataset used for pretraining language models, called Collosal Cleaned Crawled Corpus.

$cLang-8$ Cleaned version of lang-8 dataset.

$CoNLL-2014$ Benchmark dataset for grammatical error correction.

$lang-8$ Dataset for learner language used for language learning.

$mC4$ Multilingual C4, a dataset derived from Common Crawl for multiple languages.

Evaluation metrics

$BLEU$ It is a metric for evaluating the quality of text which is machine-translated from one language to another, stands for Bilingual Evaluation Understudy.

$F0.5, F-beta, F1$ Metrics combining precision and recall, with different emphasis.

$n-gram$ A contiguous sequence of n items from a given sample of text or speech.

$Precision$ Ratio of correctly predicted positive observations to the total predicted positives.

$Recall$ Ratio of correctly predicted positive observations to all observations in actual class.

$training-loss$ A measure of how well the model is learning during training.

Libraries/Frameworks

$datasets$ A library by Hugging Face for accessing and sharing datasets.

$HuggingFace$ Community and company that is engaged in creating tools for NLP, including transformers.

Models and techniques

$BART$ Model for text-generation and other tasks, stands for Bidirectional and Auto-Regressive Transformers.

$BERT$ Transformer-based model for various NLP tasks, stands for Bidirectional Encoder Representations from Transformers.

$GECTOR$ Transformer based model specifically for GEC.

$RoBERTa$ A robustly optimized BERT pre-training approach.

SMT Statistical machine translation.

SOTA State-Of-The-Art meaning best performing model in the field.

*T*5 Text-to-text transformer model.

Neural network components

batch_size Number of training examples utilized in one iteration.

$Dropout - rate$ A regularization technique to prevent overfitting.

Epochs One complete pass through the training dataset.

GRU Gated Recurrent Unit, another type of RNN.

LSTM Long Short-Term Memory, a type of RNN.

max_length Maximum sequence length for processing inputs.

RNN Recurrent Neural Network

steps Training iterations.

subwords Units of text smaller than words used in tokenization.

truncation Shortening sequences to a maximum length.

$weight - decay$ A regularization technique to reduce the complexity of a model.

Tools and formats

CSV Format of file for tabular data, stands for Comma Seperated Variables.

*m*2 Format of file for evaluation of results produced by GEC.

Contents

1 Introduction

1.1 Aims and objectives

In this paper, the step-by-step methodology for building a GEC is explored that can contribute to the respective field of NLP and following this basic principle to any further development can contribute substantially to any new coming-to-age model in the same field with substantially improved efficiency from the start itself. Through the optimal hyper-parameters in the algorithm and applying data augmentation, the goal is to improve the accuracy of grammatical error detection in the English language and thus produce the trained grammar error correction model which will generate grammatically correct versions of the input sentences.

To further the development, next is building the GEC or grammar error correction model that is called gt5 on the base model of T5-transformer. The current model is only focused on 1 language, with more data more languages can be dealt with in future experiments. The performance of the optimised model or gt5 will be later compared to past performances and essentially among both the datasets that the experiment aims to provide an analysis on, along with an analysis on the techniques applied in the algorithm that resulted in the production of the desired system. The dataset aimed for the project to provide an analysis of the implemented model is CoNLL-2014 and a shorter eligible version of cLang-8 datasets for training. The later has fewer experiments available in the research field currently, being the newest and potentially efficient.

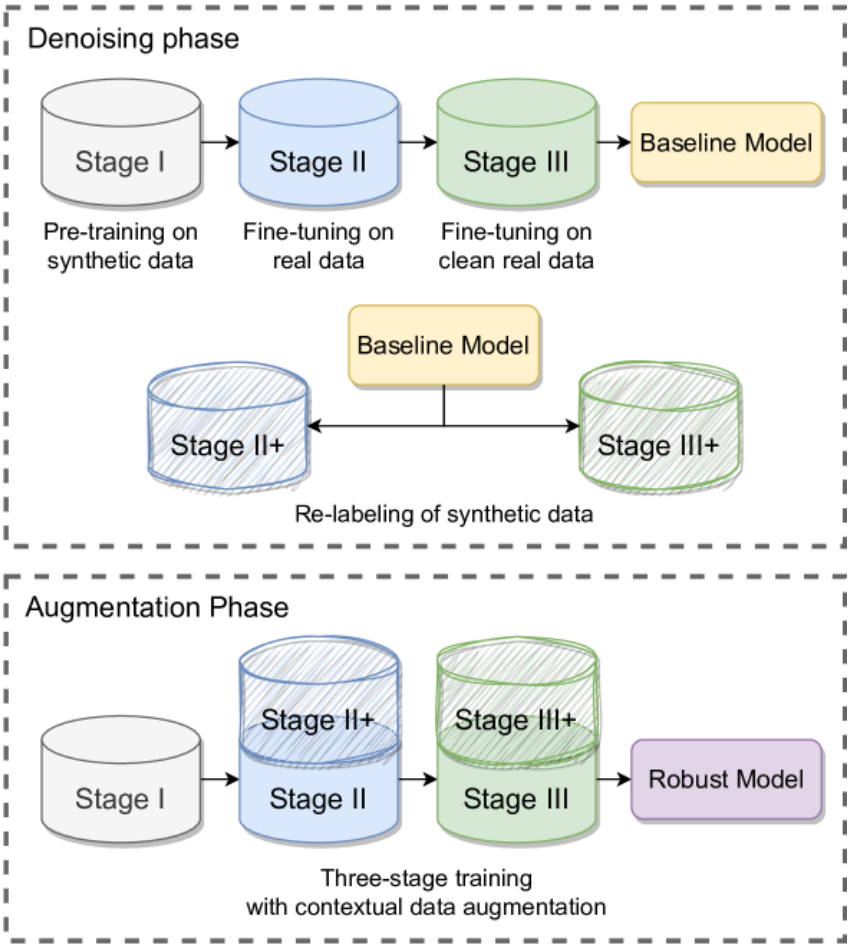

Figure 1: Fine-tuning data [8]

Therefore, the gt5 model which is to be developed in this project aims to provide convenient results when the comparison is performed to put itself as the better system as a new checkpoint for any future GEC experimentation. The final analysis will provide a clear gap between the aims and conclusion and specify the degree to which the model has performed well also along the process, discussion will be made regarding how well the applied techniques have either resulted in improvement or those which did not yield desired results but could

help in chasing the right path for the model development.

1.2 Background and literature survey

In the field of natural language processing, the quest for flawless and efficient communication has become increasingly necessary and thus the intricate correction of grammar, punctuation, and style often comes about to be challenging and therefore there are immediate requirements for such to proficient writers and speakers. In the face of this, the computational field has strived to put forward methods that can provide solutions to such challenges which has resulted in the innovative algorithms capable of automatically detecting and correcting grammatical errors in text.

The grammatical error correction (GEC) is a state-of-the-art technology built upon the ideology of combining the prowess of artificial intelligence and linguistic algorithms for the sole purpose of enhancing written communication. The need for reliable and intuitive grammar correction tools has become increasingly pronounced and this is all in correspondence with the expansion of digital landscape.

The idea of generating synthetic data to improve the model's performance has been prevalent in the field as well [43, 29]. The seq-2-seq tasks that can be spotted when researching methodologies for the purpose showcase that the method of leveraging self-supervised pre-training and thereby increasing model size can yield substantial results [47, 57, 20].

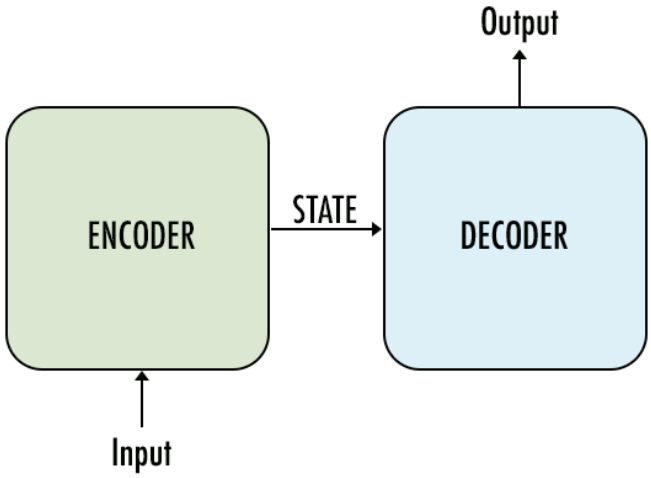

Figure 2: Enc-dec [42]

2 Literature review

2.1 GEC models

The past developments towards the models that are built on base models of t5 and mt5 have contributed significantly in tokenising the language sentences and thus performing the necessary grammatical error detection. The mt5 model was trained on mC4 corpus which is the subset of Common Crawl covering over 101 languages and composed of about 50 billion documents. But the span-prediction objective of the model does not allow it to perform GEC without much needed further fine-tuning. Another constraint is that the respective model is trained only on paragraphs and not sentences [57].

The most recent work is being done in the direction of enriching the entire model through pre-training on the data-rich tasks. Therefore, the models acquire some general-purpose abilities and knowledge that can be passed to downstream tasks. The rapid progress and diversity of techniques applied in this burgeoning field make it quite difficult to compare different algorithms, conclude the effects of new contributions, and thereby understand the space of existing methods for transfer learning. "Motivated by a need for more rigorous understanding, we leverage a unified approach to transfer learning that allows us to systematically study different approaches and push the current limits of the field."[47]

Even though such approaches and many similar [57, 51, 48] are being developed rapidly, applying such techniques to the GEC has not yet reached a concrete level of developmental stage.

Though the models implemented have contributed significantly on improving the accuracy, there comes completely new set of challenges that can be neutralised with careful approach. Whenever there is expansion in terms of application of GEC to wide range of languages, some challenges like language specific tuning for synthetic datasets and inability of synthetic data to capture complete data distribution of targeted evaluation sets result in the development of multi-lingual GEC at the cost of multi-stage fine-tuning process. Further challenges that are inevitable such as the careful consideration of learning rates and deciding the epochs make the further development of models on the pre-existing ones for bettering the overall error detection quite tedious and difficult. Therefore, GECs that can be attain the capability of reducing the steps to fine-tuning will contribute heavily to evolution of GEC models.

2.2 cLang-8 dataset

The cLang-8 dataset is much cleaner version of lang-8 which has approximately 1.16M annotated sentences available for training. But there are some erroneous and incomplete corrections in the lang-8 that can be corrected and those corrections can help in building better GEC which are much more efficient and easy to optimise for usage. Therefore, with the right kind of methods, clang-8 dataset can be built from the raw lang-8 which will help in achieving the aim. The common experiments that are possible are supervised and unsupervised approach. For this project specifically, supervised approach will be utilised in hope of achieving required up-to-the-mark results. Thus, new target sentences will be generated along with the appropriate file to generate corpora for combining the new target generated file and lang-8 for corrected English corpora [10].

2.3 CoNLL-2014 dataset

The CoNLL-2014 Shared Task on Grammatical Error Correction (GEC) introduced a benchmark dataset that has significantly propelled research in this field. The dataset comprises a collection of English sentences with corresponding error annotations, providing a valuable resource for developing and evaluating GEC systems.

The dataset is characterized by its diversity in error types, sentence complexity and language proficiency levels. Key challenges associated with the dataset include [27] : -

1. Error complexity: The dataset encompasses a wide range of error types, from simple spelling mistakes to complex grammatical errors, requiring robust models capable of handling diverse error patterns.

2. Data sparsity: Certain error types may be underrepresented, leading to challenges in model training and evaluation.

3. Annotation inconsistencies: Variations in annotation guidelines and human annotator judgments can introduce noise into the dataset.

Though, the respective dataset has played pivotal role in much advancing the GEC research [45]. Some effective approaches in the field considering the dataset includes: -

1. Neural Sequence-to-Sequence Models: These models have achieved state-of-the-art performance on the CoNLL-2014 dataset. Approaches such as Encoder-Decoder architectures with attention mechanisms have been extensively explored, demonstrating superior capabilities in capturing complex sentence structures and generating accurate corrections.

2. Pretrained Language Models: Leveraging large-scale pretrained language models, such as BERT and RoBERTa, has further improved GEC performance. These models provide rich contextual representations, enabling a better understanding of the sentence and facilitating error detection and correction.

3. Data Augmentation and Transfer Learning: Techniques like backtranslation, synonym replacement, and noise injection have been employed to augment the training data and improve model generalization. Additionally, transfer learning from related tasks, such as machine translation or text summarization, has shown promising results.

2.4 More attepts on GEC improvements

The transformer architecture has prevailed the disciplinary development of GEC models and some have excelled but usually in English language and thus lots of developments are in place for lot of other languages [53]. The SOTA model for English GEcTOR [46] is capable of applying multiple transformer-based encoders like BERT [24], ROBERTa [41] and XLNet [40]. This is essentially to develop a sequence tagger that can be capable of applying pre-defined corrections to the given input text.

Development in the multilingual approach has provided for possibilities to come up with single model that can work on multiple languages. Though this development is accelerated since possibilities have displayed themselves, it's not as advanced as the GECs for English language itself. But the architectures that are produced in the process have proven themselves to catalyse the error detection on English as well, improving the accuracy at a higher rate than before.

BERT is trained on raw corpora of large size in order to learn words and sentences which have further facilitated lot of development in various tasks [53]. MLMs have been utilized for not just classification and sequence labelling tasks but its utilization was for language generation as well by combining it with 'EncDec' models [38]. Common methods among them are infusion and initialization. In the 'init' method, downstream task model is to be initialized with the parameters of a pre-trained MLM which is then trained over a task-specific training set only [38, 48]. The approach was faulted, meaning the high-degree of training that such tasks require is not possible in the case of MLMs. Thus, sequence-to-sequence language generation here is rendered useless [65, 44]. Further, in the fusion method, the pre-trained representations of MLMs are utilized as additional features for task-specific model training [65]. The flaw in the approach is that MLM is capable of preserving the pre-training though it is unable to adapt to task-specific distribution of inputs, resulting in a failed attempt to utilise the complete potential of MLM by the GEC model.

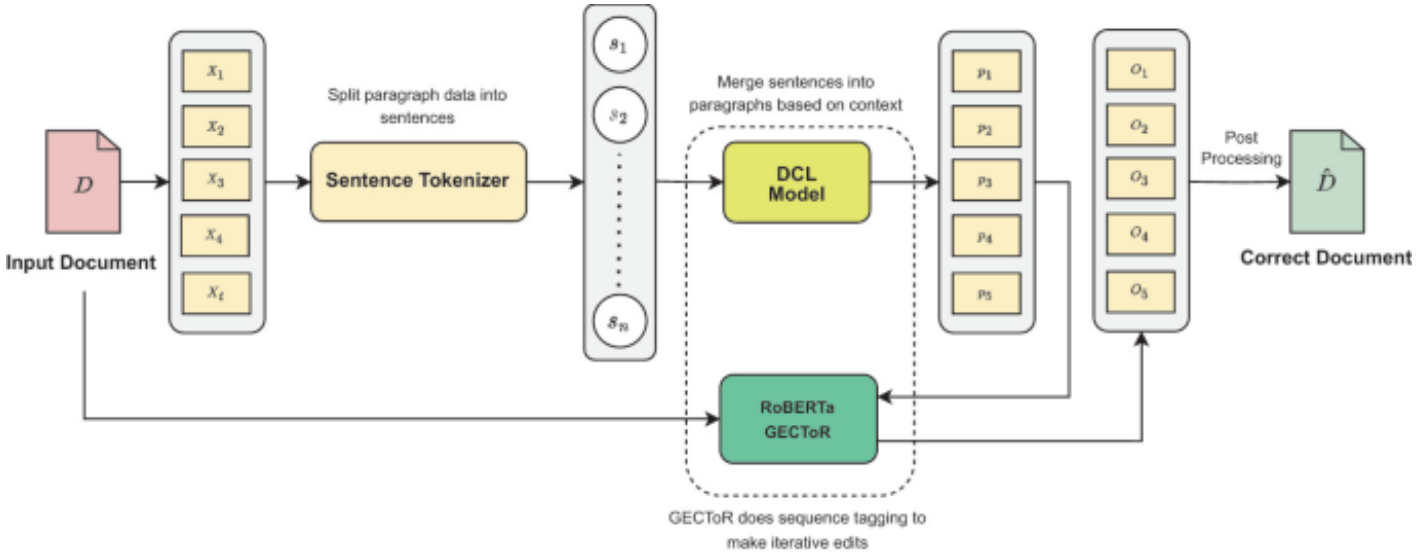

Figure 3: Improvements in GEC architecture [16]

The improvement in GEC systems can be made by addressing the gap between grammatically correct raw corpora and GEC corpora. There often exists a mismatch problem between contextual knowledge from pre-trained models and target bilingual machine translation [55].

2.5 Alternate basemodel: BERT

BERT stands for 'Bidirectional encoder representations from transformers'. It was launched by Google in 2018 and since then has served as a powerful language model. Some challenges in the field of NLP or 'natural language processing' include high-priority ones like shortage of data, and research is constantly being carried out to solve the problem and expand the field to make it more easily available to a lot more sources. One of the techniques that help in bridging the gap for the shortage of data is by training large language models on large amounts of unannotated text on the web, also known as pre-training. This process has been proven to be efficient in improving the accuracy rather than training the given datasets from scratch by fine-tuning them on individual NLP tasks.

The 'masked language model' by BERT can be a substantial help in improving the procedure of pre-training in terms of quality by the process followed in building such models. Usually, it includes making some of the tokens from the input, and then achieving its objective of predicting the original vocabulary id of the masked word based only on its context. The MLM's objective can help in effectively pre-training the deep bidirectional transformer [24].

Since BERT strives to address the limitations of NLP by leveraging transformers because the transformer architecture has helped in significantly understanding the entire sentences simultaneously unlike the preceding models that were good at processing individual words in the sentences respectively. This holistic approach captures the relevant relationships between the words at different positions [1].

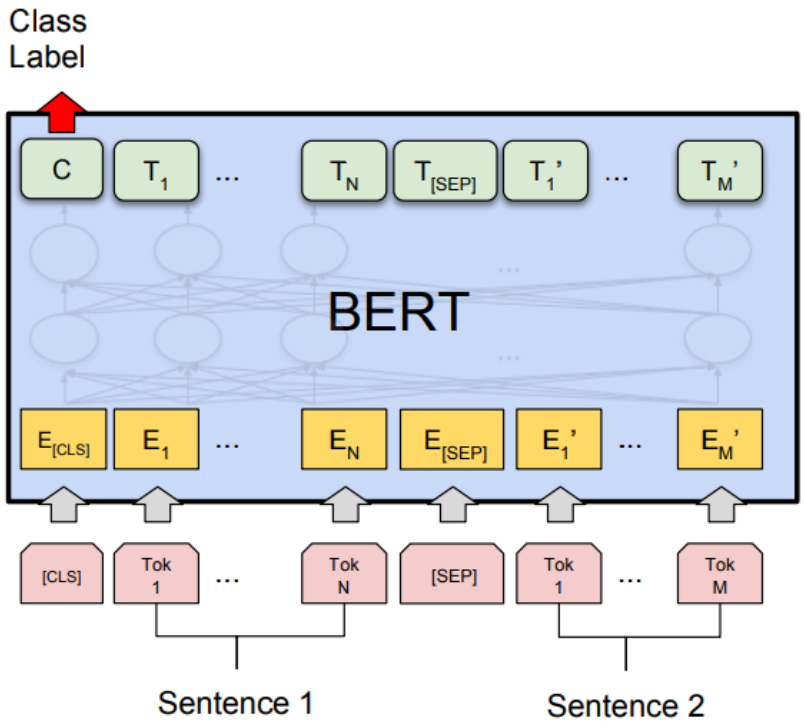

Figure 4: BERT architecture [14]

2.6 Text-to-Text transfer transformer

The T5 transformer was introduced by 'Google AI' in 2019. Being a prominent architecture in the field of natural language processing, it is only natural to explore its applications, training methods, and specific impacts on the NLP projects made available publicly. The model can be found at [7].

2.6.1 Prominent characteristics

1. Encoder-Decoder Architecture: The T5 follows the standard encoder-decoder structure which is quite commonly used in sequence-to-sequence models. The encoder processes the input text, capturing its meaning and relationships between words. The decoder then utilizes the encoded information to generate the output text.

2. Transformer Blocks: The encoder and decoder consist of stacked Transformer blocks which is its standard feature. These blocks employ an attention mechanism, allowing the model to focus on relevant parts of the input sequence when generating the output.

3. Unified Text-to-Text Format: A core feature of T5 is supposed to be its ability to frame various NLP tasks as text-to-text problems. This is achieved by prepending a specific task prefix to the input text and instructing the model on the desired transformation like 'summarize' for summarization.

2.6.2 Available training methods

- Pre-training: The T5 models are trained on huge amounts of data of texts and code and the dataset is called C4 corpus. This pre-training plays a pivotal role in allowing the model to learn and understand the essence of a language in a broad sense and then it can be used by the model to apply it to various forthcoming downstream tasks.

- Fine-tuning: This is quite essential for reducing the training time when training is initialised so it is only a smart option to fine-tune the model rather than start training from scratch. Due to the pre-trained

knowledge, the T5 models are equipped with the ability to precisely fine-tuned on NLP tasks by adjusting the model weights on a task-specific dataset.

The broad range of applications that are possible due to the model are text-summarization, machine translation, question answering, text classification and text generation. The impacts on the field of NLP are quite prominent and models can be compared with others like BART, and BERT for performance and efficiency depending on the suitability for tasks [4]. The versatility, efficiency and performance improvements are quite evident because of which the ability to handle diverse NLP tasks through text-to-text formulation has streamlined workflows further reducing the need for specialized models for individual tasks. Pre-training on C4 corpus has helped in bettering the development cycles [49].

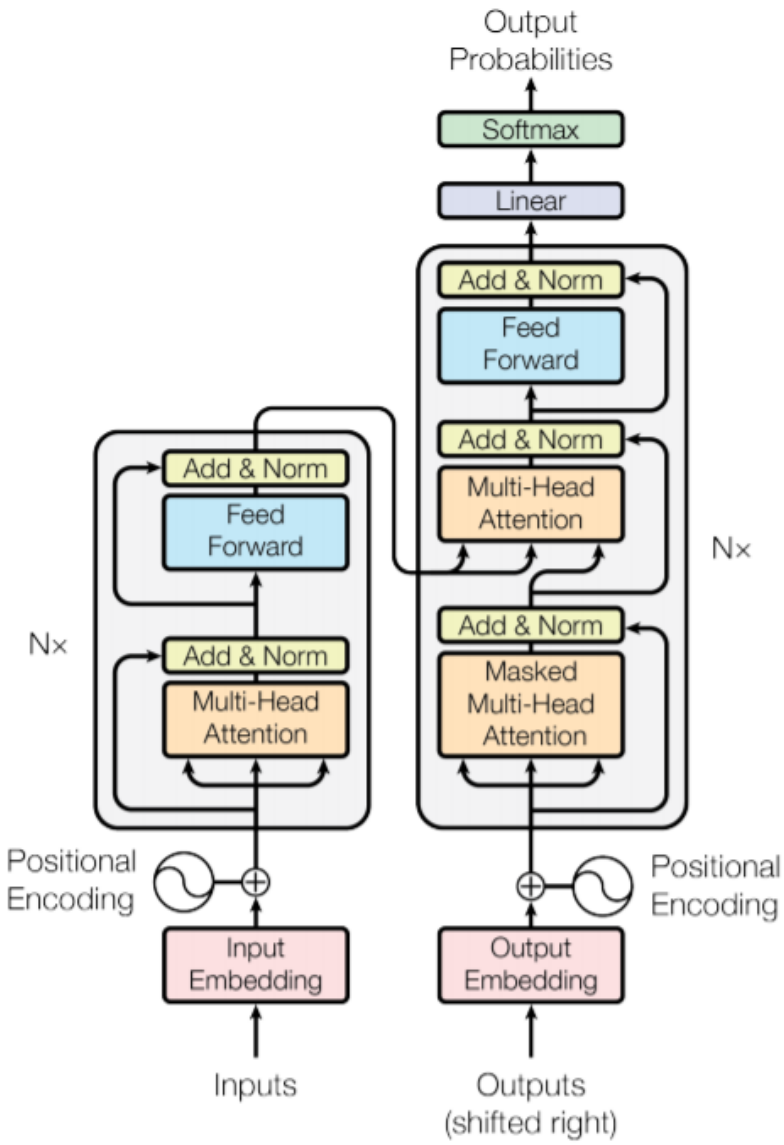

Figure 5: Transformer model [13]

There are more versions accurate for more complex data processing and for the requirements of more diverse category of tasks. The other models are T5-small, T5-large, T5-3B, T5-11B and UL2 or 'universal language model 2' [7].

1. T5 Small: This is the smallest version with approximately 60 million parameters. It's suitable where computational power is limited.

2. T5 Large: This mid-sized version with 770 million parameters. It offers a good balance between performance and efficiency and can handle more complex problems compared to the base model.

3. T5 3B: This version has 3 billion parameters. It can demonstrate improved performance on many NLP tasks as compared to the base and large models. However, it requires more computational resources for training and inference.

4. T5 11B: This is the largest version with a staggering 11 billion parameters. It achieves state-of-the-art performance on various benchmarks, but its immense size necessitates significant computational resources.

5. UL2 (Universal Language model 2): Though not strictly a T5 variant, UL2 shares a similar Transformer-based architecture. It's pre-trained on various denoising objectives, showcasing strong performance on tasks like filling in missing words and summarization.

2.7 Grammatical error correction pipelines

GEC pipelines have a crucial role to play in automating the process of identifying and correcting the grammatical errors in the input text to the system.

2.7.1 The core functionalities of the GEC pipelines

1. Error identification: The pipelines make the errors that are identified through the system in the input texts possible. These errors can include spelling mistakes, incorrect verb tenses, subject-verb agreement issues, prepositional phase errors, and sentence structure problems.

2. After recognizing the incorrections in the sentences, the corrections are made by either suggesting or generating the grammatically correct alternatives for the erroneous segments. Though this leverages the pre-trained models or specialised GEC models fine-tuned on error-annotated datasets.

3. The output is finally generated highlighting the problem with the input sentences and thus presenting the grammatically correct version.

2.7.2 Important components

1. Pre-processing Module: This module prepares the input text for the error identification model. It involves the necessary tasks like tokenization which is meant to break down the individual words into units or tokens, and normalization (handling punctuation and casing).

2. Error Identification Model: This core component of the pipeline employs NLP models, such as rule-based systems, statistical machine translation models, or deep learning models trained on GEC datasets, to analyse the input text and detect grammatical errors [52].

3. Error Correction Model: This module is usually based on pre-trained language models and/or specialized GEC models for the purpose of generating candidate corrections for the identified errors. It aims to produce grammatically correct and language-fluent solutions given the context surrounding the text [36].

4. Postprocessing Module: This final stage performs any necessary cleaning or formatting on the generated output text.

The benefits surrounding the component are improved communication quality, efficiency, scalability and adaptability. The automating process of the GEC reduces the time and effort compared to manual proofreading. There are though certain future directions to it as well, exploring is worthwhile [61]. Some of them are improved model performances, domain-specific adaptation, explainability and adaptability. More research is employed in the direction of making the GEC pipelines more transparent and certain domains of experiments like building the GEC pipelines specifically for writing styles can further enhance its performance in specialized contexts [31]. The 'Huggingface' provides the pipeline function called 'text-to-text generation' which has played pivotal role in recording predictions from the GEC and has been very helpful to the project [12].

2.8 Tokenizers

Tokenization plays a crucial role in GEC pipelines by preparing the input text for the error identification and correction models. The pre-processing steps include breaking the texts into smaller individual units so that they can be smoothly processed by the NLP models. It can be used for creation of POS or part-of-speech tags or morphological information. It can be used quite efficiently in strategies to locate the errors. Common approaches for tokenization is splitting the text into individual words based on whitespace boundaries. But there is small challenge as it cannot hyphenate words effectively. There is an approach which is computationally expensive which is breaking down texts into individual characters, which according to this approach captures finer-grained information about the morphology. Therefore, the better approach that balances the computational efficiency and morphology is splitting words into smaller units called subwords which means prefixes, suffixes or in some case whole words. Sentence-piece tokenization is a versatile approach that can combine word-level, character-level, and subword tokenization depending on the context [35]. This allows for flexibility in handling different language characteristics. Choosing the right kind of approach is imperative due to several reasons which can be the language, each of which has its own morphological complexity and can benefit from subword or character-level tokenization or more reasons like GEC model architecture and availability of computational resources [31]. Effective tokenization improves the error-detection ability of the model by revealing inconsistencies and highlighting potential error locations. Right strategy leads to efficient model performance [19].

2.9 Encoder-decoder architecture

The encoder is to process the variable-length input sequence and then compress it to the fixed-size representation called as 'context-vector'. For this specific purposes, recurrent neural networks or RNN like LSTMs (long short term memory) or gated recurrent units (GRU) are used. On the other hand, the decoder utilises encoded representation from encoder to generate new variable-length output sequence. The decoder typically predicts one element of the output sequence at a time, relying on the previously generated elements and the encoded context. This is how the input sentence is translated into an output sentence. The encoder-decoder architectures are powerful and versatile tools for various sequence-to-sequence learning tasks. There are many forthcoming advancements expected in the field of natural language processing due to experiments being carried out on attention mechanisms, novel architectures, and broader applications [66, 58].

More complex structure of the same architecture involves building hierarchical encoder-decoders to handle various complex input structures such as nested sentences or documents. Each level of the hierarchy focuses on capturing information at different granularities.

The challenges remain in training complexity due to being computationally expensive especially in the imperative cases of large datasets. Also, explaining the inner workings of the model remains a bit of a challenge, specifically the attention weights and therefore trust in their outputs can be increased once sufficient research measures are available to take in that direction [54, 50].

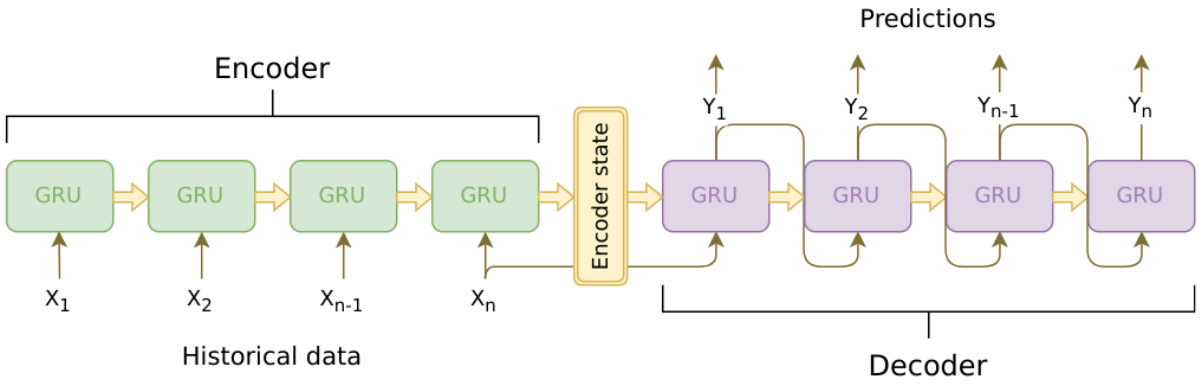

Figure 6: Encoder decoder architecture [2]

2.9.1 Applications

1. Machine Translation (MT): Encoder-decoder architectures are the foundation for most modern machine-translation systems. The encoder processes the source language sentence, and the decoder generates the target language translation.

2. Text Summarization: The encoder reads the input document, and the decoder condenses it into a shorter summary that captures the key points.

3. Text Generation: This includes tasks like question answering, dialogue systems, and image captioning. The encoder processes the input context (e.g., a question or an image), and the decoder generates the corresponding textual response or caption.

2.9.2 Attention mechanism

Significant advancement in the encoder-decoder architectures is the introduction of attention mechanism [15]. The reason why this has led to much more development in terms of the performance of models is how the 'Attention' allows the decoder to focus on specific parts of encoded representation that is relevant to generating the next element in the output sequence [23]. The architecture in "Attention is all you need" [53] relies completely on the attention mechanisms and not on recurrent or convolutional layers. Transformers have achieved state-of-the-art performance in many NLP tasks and offer advantages like parallelization for faster training.

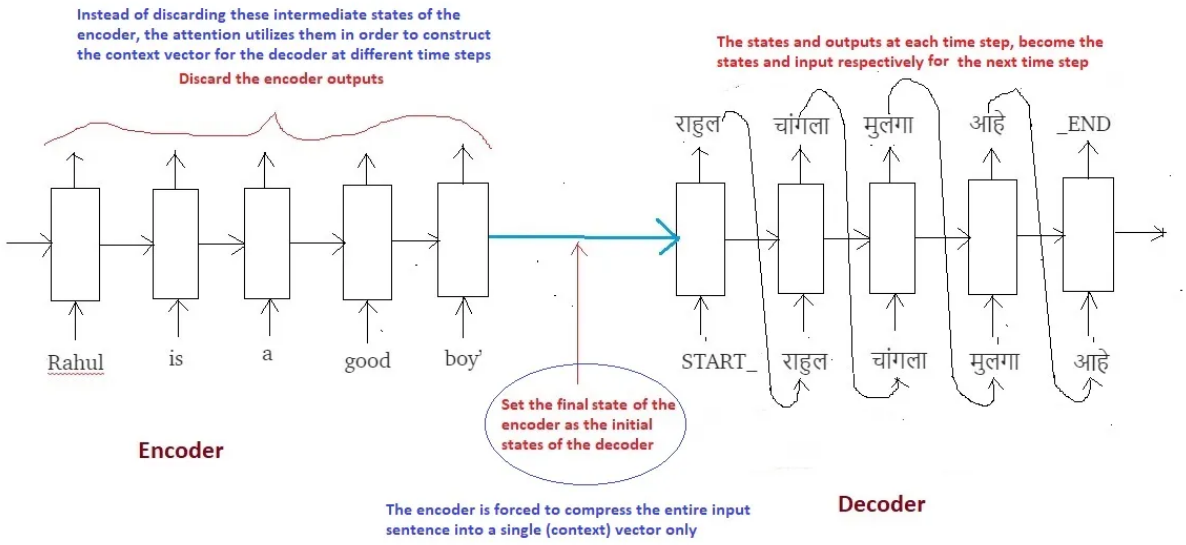

Figure 7: Example of Attention mechanism for English-Hindi language conversion [37]

2.10 More experiments with the lang-8 datasets

Lang-8 is the base dataset from which cLang-8 dataset was created. Therefore, certain experiments on the same dataset can provide a perspective highlighting the direction in which GEC model development has been heading. Some work has been done on a phrase-based statistical machine translation (SMT) approach for GEC. Using the lang-8 dataset, the model achieved significant improvement in error-correction over baseline methods [64]. An approach involves employing a neural-machine translation techniques using sequence-to-sequence models. After training on large portion of the lang-8 dataset, the best F0.5 score achieved was 39.90. The analysis was centred around benefits of neural methods over traditional SMT approaches [59]. An experiment on achieving a spectacular F0.5 score of 49.49 on a subset of lang-8 demonstrates the improved ability of the model to generalize well across different types of grammatical errors present in the dataset. This neural network-based

approach was focusing on using large scale datasets for training [22]. BERT model when trained on lang-8 dataset for a sufficient portion along with other datasets could result in F0.5 score of 45-50. There were some specific intricate configurations according to subsets of data utilized [32]. Another approach involves data augmentation by generating synthetic errors and then training the model on lang-8. The F0.5 score of 42.16 reveals the effective method of data augmentation on GEC models and how it can become capable of handling real-world errors [29].

2.11 High-performance computing

The computational demands for the high-performance computing or HPC is quite high for GEC tasks and many relevant NLP tasks. Those based on deep learning architectures like transformers constantly require processing massive amounts of data. This can involve millions of sentences with annotated errors, placing significant computational strain on standard CPUs. The trend in GEC is towards larger and more complex models for improved performance. However, training these models necessitates even greater computational power due to the vast number of parameters involved. The more crucial feature of HPCs comes into play when they are utilised for employing optimization techniques such as the gradient descent which proves to be extremely effective for training, accelerating the convergence process and thereafter improving the model performance.

2.11.1 Benefits of the HPC for GEC

1. Faster Training Times: HPC clusters with GPUs or TPUs can significantly reduce training times for GEC models compared to using standard CPUs. This allows for faster experimentation, model development cycles, and adaptation to new datasets.

2. Handling Large Datasets: HPC resources enable GEC researchers to leverage larger and more diverse datasets for training, potentially leading to models with better generalization capabilities and improved performance on unseen data.

3. Exploration of Complex Models: The computational power of HPC facilitates further exploration of intricate GEC models with advanced architectures, potentially leading to more impressive advancements in the field [25].

There are available alternatives, though for this project specifically the HPC is provided by the university called 'CREATE' [26] and is free for student usage but also proves to be computationally effective for the level of model built in the project. Though for more diverse purposes the considerable alternatives are pay-as-you-go cloud-based services provided by Amazon, Google and many other platforms. Another advantage includes leveraging distributed training frameworks across multiple computing nodes within an HPC cluster can further accelerate the training process for large GEC models [19, 34].

2.12 Errant as metric

Errant is a metric that penalizes primarily the number of errors in a machine translation output [18]. An error can be anything from a missing word to a completely nonsensical phrase. Particularly the Errant score is better as low score and is calculated by dividing the total number of errors by the total number of words. Though, F0.5 score is a notable metric and also the focus of this project. It is the harmonic mean of precision and recall. Here, precision is the portion of correctly translated words and recall is the portion of words in reference translation which are correctly translated in machine translation output. The F0.5 score specifically puts more emphasis on precision by weighting it by 0.5^2. A higher F0.5 score indicates a better translation. It is a variant of the F-beta measure which finds application in various machine learning tasks, particularly machine translation. It balances precision and recall with a stronger emphasis on precision (beta=0.5). This can be beneficial when mistranslations are more detrimental than missing a few words in the translation, such as in medical documents or legal contracts [9].

Its difference over the similar metrics like F1 score include accurate translations, thus reducing the penalty on missing information. This is extremely crucial in domains where factual accuracy is paramount. F0.5 is

much easier to interpret than more complex metrics like BLEU score, which considers factors like n-gram overlap and brevity penalty [60]. The clear focus on minimizing false positives makes it a straightforward measure for non-experts.

Though it is suitable for the project, for more diverse range of projects on similar topics its limitations can be a problem which is primarily overemphasis on precision. In certain scenarios it might overlook the importance of capturing all relevant information.

3 Objectives, design and specifications

3.1 Objective

3.1.1 Purpose of the project

The objective of the project is to experiment with a capable base-model with new datasets that have come into the field and then provide more clarity on the experimentations how accurate can be the utilisation of such new dataset to produce state-of-the-art grammatical error correction models. This is expected by equipping a pre-trained base model with additional sufficient fine-tuning and data-manipulation techniques to produce another model. Referring the new model as gt5, it is aimed at producing results with the new cLang-8 dataset and CoNLL-2014 with promising traits of helping in producing the model that can deliver results with less steps in fine-tuning and correcting the errors in a sentence. Therefore, the experiments are carried out to produce an array of results that can give an idea of the best approaches available to manoeuvre through the available pathways, if also required for more such projects in the future.

3.1.2 Scope of the project

The project's scope is to correct the errors of types punctuation errors, verb tense errors, subject-verb agreement errors, spelling mistakes and morphological errors. The datasets adopted for the project are utilized for the purpose of achieving this particular objective. Though the datasets don't explicitly categorize the types of errors rather all sentences in total are mixed with the kind of errors potentially present in them, some observations could be made about the nature of sentences present in them. There can be a higher concentration of errors reflecting challenges faced by learners, such as subject-verb agreement for complex tenses, misuse of articles ("a" vs. "the"), or confusion with phrasal verbs. These kinds of errors introduced due to manual intervention can make the dataset quite diverse. But it is intended that subtly the sentences contain the biases of the writers or the model used in the past for creating the datasets or amending them.

3.1.3 Expected outcomes

The outcomes of the project is dependent on the experiments planned by the model on the datasets. Since covering huge analysis in limited time is critical and also the hardware for carrying out the experimentation, initially the basic run for limited epochs (3 or 4) with basic hyper-parameters will be recorded for each dataset. Then to improve the performance, more techniques will be attempted like specifically data augmentation and hyper-parameter tuning and probably increasing the epochs simultaneously. Also experimenting with different proportion of train and test of dataset though not much is expected from changing the proportion. In all the cases, the proportion split will be constant (70%-30%) to make valid comparison in the F0.5 scores of the outputs which are from the Errant module. The expected F0.5 scores are in the range (0.3-0.5) which should be considered good benchmarks for this project. The span-based correction scores are the main ones; therefore, token-based correction scores will be compared later to provide a deeper analysis. If the scores are achieved in due time available, tweaking the intricate parameters can help in potentially achieving slightly higher scores. Therefore, after the experiments, best-case corrected sentences will be procured as the results to analyse the accuracy of grammar correction on the language by the model.

3.2 Design

3.2.1 Model architecture

The architecture aimed at experimenting and utilised finally is based on T5-model which text-to-text transfer transformer. It includes the encoder-decoder structure where the encoder is responsible for processing the input text and producing sequence of continuous representations. The decoder on the other hand, takes the continuous representations to generate output text. More key components are as follows: -

1. Tokenization is done by using the 'AutoTokenizer' associated with the t5-base model to convert the input text into tokens and then token IDs so that model can process them.

2. The key role of the attention mechanism is using scaled dot-product attention. In the encoder, attention allows each position to focus on all positions in the previous layer. In the decoder, masked self-attention ensures that the prediction for a particular token depends only on the known outputs at earlier positions.

3. Each layer in encoder and decoder contains the feed-forward network which is applied to each position independently.

The pretrained encoder-decoder T5 model is worthwhile choice for the project requirements for a variety of reasons. Text-correction is all about generating grammatically correct versions of the corrupted input sentences and at such sequence-to-sequence tasks, conditional generation models excel at. The model is trained on huge amounts of correct and incorrect sentences, the patterns and rules that exist between the texts and sentences are quickly adaptable by the model.

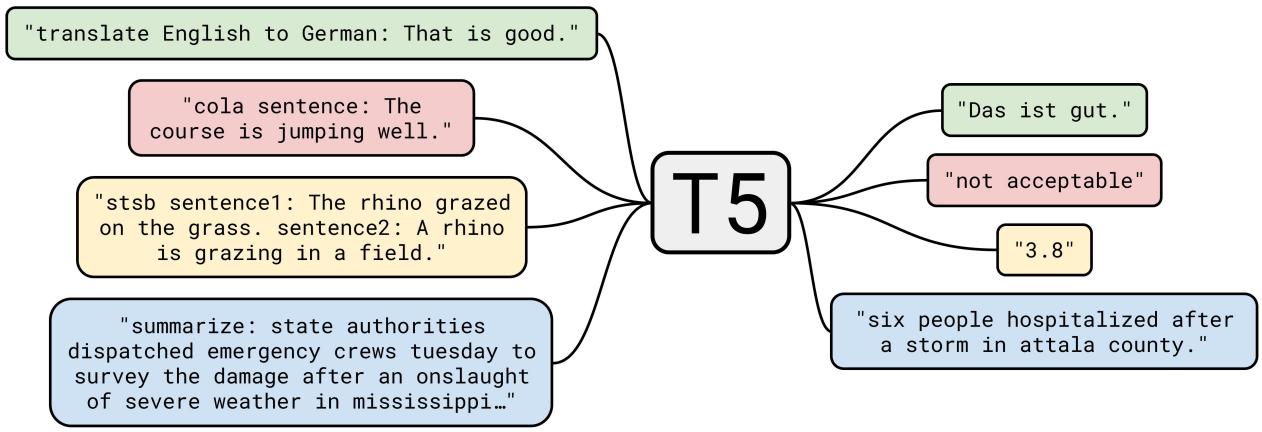

Figure 8: Functioning of T5 [11]

For the project, the chosen model is 't5-base' which has 220 million parameters and given the limitations of available memory space, is the optimal choice for the project compared to larger more complex models such as T5-3b and T5-11b with 3 billion and 11 billion parameters respectively.

3.2.2 Applicable hyper-parameters

The learning rate is an essential part of the architecture. The learning rate is a scalar value (η) that controls the magnitude of updates to the model's weights (w) during the optimization process. It determines how much the model adjusts its weights in response to the error (loss function) gradient.

Update rule: $w_{t+1} = w_t - \eta \nabla L(w_t)$

where,

- w_t represents the weight vector at the current iteration (t).
- $\nabla L(w_t)$ denotes the gradient of the loss function (L) with respect to the weights at the current iteration.
- w_{t+1} signifies the updated weight vector for the next iteration.

The higher learning rate η will lead to larger weight updates but smaller learning rate though being small, leads to more stable convergence process though being slower than former. But larger learning rate can help in faster convergence though it has high chances of increasing the risk of overshooting minimum loss function and oscillating around it. Therefore, for the required datasets in the project, it is chosen appropriately to be '5e-5' which is optimal but more feasible for the required purpose which means 5 times 10 raised to the power of -5 and relevant alternatives can be introduced by replacing '5e' with '2e'. The weigh decay (λ) is a hyper-parameter that introduces an L2 penalty term to the loss function. It acts as a regularizer to prevent overfitting by penalizing large weight values.

Loss function with weigh decay: $L'(w_t) = L(w_t) + \frac{\lambda}{2} ||w_t||^2$

where,

- $||w_t||^2$ represents the squared L2 norm of the weight vector, essentially the sum of squares of all weight values.

The weigh decay term penalizes model for having larger weights, compelling them to stay close to zero. Depending on the parameter being high or low, it results in high or low penalty on large weights. Higher weigh decay reduces overfitting but potentially increases the bias of the model. Smaller one on the other hand, ensures the model can adapt to complex patterns simultaneously increasing the chances of overfitting.

The gradient is calculated with respect to a subset of the training data (the batch), if applied instead of the entire dataset. Batch size (B) is the number of training examples used to compute the gradient of the loss function in a single update step. It affects the efficiency of the optimization process and the generalization performance of the model.

Gradient descent update (with batch size 'B'): $w_{t+1} = w_t - \eta \cdot \frac{1}{B} \nabla L(w_t, \{x_i, y_i\} \mid i \in B)$

Dropout rate is the regularization technique in neural networks to prevent overfitting. During training, a random subset of neurons in a layer is temporarily disabled (dropped out) with a probability 'p'. Therefore, the network is forced to learn features that are robust to the absence of any particular neuron and prevents neurons from co-adapting excessively to specific patterns in the training data.

3.2.3 Applied data augmentation

Data augmentation is the technique of introducing noise in the data to force the training model to better adapt to the patterns between the data. This supposedly increases the chances of performing better when faced with unseen datasets, thus increasing its generalizing ability. More advantages of data augmentation include the expansion of the effective training data size, allowing the model to learn from a broader distribution of examples. However, this technique is very subjective to the type and size of the dataset applied. Thus, there are equal chances of the model performing well or not given the particular kind of dataset. The focus of this project is introduce random errors to either insert, swap, replace or delete words randomly among the sentences. Before the pre-processing function, this technique is applied to both the training and validation set of the dataset so that the model can potentially improve its detection capabilities among the existing patterns in the language. This is carried out by the 'introduce_errors' function of the algorithm.

```
# ------- Applying data-augmentation
# Function introduces random errors.
def introduce_errors(sentence):
    words = sentence.split()
    if len(words) < 2:
        # Skips very short sentences
        return sentence
    error_type = random.choice(['swap', 'insert', 'delete', 'replace'])

    if error_type == 'swap' and len(words) > 1:
        # Introduces a swap error between random adjacent words
        index = random.randint(0, len(words) - 2)
        words[index], words[index + 1] = words[index + 1], words[index]
    elif error_type == 'insert':
        # Inserting a random word at a random position
        random_word = random.choice(words)
        index = random.randint(0, len(words))
        words.insert(index, random_word)
    elif error_type == 'delete' and len(words) > 1:
        # Deleting a random word
        index = random.randint(0, len(words) - 1)
        words.pop(index)
    elif error_type == 'replace':
        # Replacing a random word with another random word
        index = random.randint(0, len(words) - 1)
        random_word = random.choice(words)
        words[index] = random_word
    return ' '.join(words)
```

Listing 1: Applied data augmentation technique to the input dataset

3.2.4 Data pipeline

The data is loaded using 'datasets' module in the format of csv file for obtaining the performance by the model. The data is split into train and validation sets before pre-processing. In the pre-processing step, each set is passed into the function 'preprocess_function' which captures the data from the dataset in columns 1 and 2 which represent incorrect and correct sentences respectively. Further the input and target sentences after capturing them individually are tokenised and to ensure consistent input length for the model, padding and truncation are used. If these parameters are not utilised, the input would be of inconsistent shape and the model will crash every time before processing the inputs. Therefore, to avoid the inhomogeneous shape of the input, pre-processing is applied to ensure smooth data processing.

3.2.5 Training procedure

The training procedure is consisting of very important steps which includes the data loading, model initialisation, training setup and evaluation.

1. The 'T5ForConditionalGeneration' class is used for loading the T5 model, which is designed for various text generation tasks. This model converts any text-based problem into a text-to-text format.

2. The 'AutoTokenizer' class is utilised for loading the tokenizer associated with the t5-base model. The purpose of tokenizer is to pre-process the input and the target sentences by converting them into the token ids for the model to easily comprehend.

3. When the dataset is loaded in CSV format, performance is initially obtained on cLang-8 and CoNLL-2014 without data augmentation, but later, data augmentation is introduced to obtain various results. The data is augmented to introduce random errors. The split of the train-evaluation set is 70%- 30%.

4. The 'preprocess_function' is the function defined to tokenize the inputs and targets, ensuring they are padded and truncated to a maximum length of 512 tokens. This is imperative to not lose performance due to inconsistent processing of sentence-length.

5. The 'training_args' defines the needed parameters like learning rate, batch size, number of epochs, and checkpointing. The learning rate is set to '5e-5' suitable for the size of dataset used. The custom data collator, 'DataCollatorWithPadding' is used to handle padding during batch processing. The optimizer chosen for desirable performance is 'AdamW'.

6. The 'Trainer' class is initialised with 'trainer.train()'. All the inputs fed into it are model, training arguments, datasets, tokenizer, data collator, and optimizer.

7. After the training completes, the model is utilised for text generation using the pipeline function from the 'Hugging Face' library specifically using text2text-generation. Here, the predictions are saved for final analysis or calculating the scores for the evaluation dataset. The final score is done in a neighbouring file specifically for calculating F0.5 score.

8. The predictions when made on the HPC or 'CREATE' which is a high-power computing service provided by the institution, for the dataset were first imported from the HPC and then evaluated on the evaluation python script 'Gecmodel_errEval.py' for specifically processing the files through the Errant module. The file prepares the m2 format for hypothesis and reference text files before making a comparison and then produces the final scores.

3.2.6 Evaluation metrics

ERRANT also stands for Evaluation for Robustness of Reference-less Automatic Natural Language Translation. It is a toolkit designed for evaluating the quality of machine translation outputs, particularly when lacking reference translations for comparison. However, it is used to also apply to other tasks like Grammatical Error Correction (GEC) as an evaluation metric. Grammatical correctness would not necessarily be enough to analyse

how well the model as a whole works or how well is it capable of evolving. Thus to ensure more precise analysis is done such as testing the production of unnatural or awkward sentences and through that testing the fluency and naturalness of the language by analysing word choice and sentence structure, cohesion and coherence within the text along with readability and overall clarity. Errant is quite equipped to also correct the over correctness of the GEC system in case the output sentences totally lose their meaning. The F0.5 score is the primary metric for the project, including the span-based correction score, which is the correctness over the whole sentence level, and token-based, which is the correction on individual word-level or token-level. It falls under the concept of F-beta scores and combines the concepts of precision and recall. Precision measures the proportion of corrected sentences that are actually grammatically correct and Recall measures the proportion of grammatically incorrect sentences that the GEC corrects. Advantages of Errant scores include avoidance of false positives and it is quite suitable for imbalanced datasets as well. Unlike BLEU score which might reward overly similar outputs even if not grammatically correct, F0.5 score emphasizes identifying actual errors and correcting them precisely.

3.3 Specification

3.3.1 Dataset

Dataset is the centre of the GEC and careful planning is required to approach the processing of the data. The strategies are essential to ensure considerable results are obtained in time. cLang-8 is a cleaned version of the lang-8 corpora. Due to its origin from the English learners as English as a second language or ESL, it encompasses the errors from broader category of the language speakers than just the native ones. The mistakes are inclusive of grammar, syntax, vocabulary usage, and punctuation. Due to the advantageous nature of the dataset, the models trained on them can be better at learning the patterns of correcting the errors and produce promising performances. The dataset comes from real-world texts and has millions of sentences though for the project, it has been shortened to 1,000 sentences closer to size of another dataset CoNLL-2014 to be finally tested to evaluate the built GEC, the datasets are supposed to be almost equal in the error-distribution to compare the model performance. Therefore, a proper comparison can be made among the tested datasets. It also simplifies the training pipeline thus reducing the complexity in training the system. Another dataset that is to be considered for comparing the model built will be on CoNLL-2014 given its popularity among the benchmarks by past GEC models.

3.3.2 Hardware and software requirements

HPC or high-performance computing

The HPC utilised for the project is 'CREATE' by the University of King's College London as discussed before in Section [2]. Grammatical error correction is a complex challenge in the field of natural language processing. Therefore, to analyse sufficiently large amount of data with feasible time duration and accurate management of memory there is a need for a platform that is capable of extending such support. The reason why HPC clusters provide the ideal environment for such high-end computing is because they combine numerous high-performance computing nodes each of which are equipped with CPU, GPU or specialized accelerators. Through the action of distributing the workload across these nodes, HPC enables parallel processing, significantly accelerating training and evaluation times. For instance, training a complex GEC model on a single CPU could take days or even weeks, whereas an HPC cluster could complete the task in a fraction of the time.

Python IDE

The environment of 'PyCharm' IDE [30] provides a comfort zone to produce the necessary codes that are the ultimate goals of the project. Among many other possibilities is 'PyCharm' IDE, which provides an easy infrastructure to search and download required dependent modules and libraries. It is easy to locate the files on the machine and, for this project's purpose specifically, can be easily transferred to HPC for further detailed processing. The environment is equipped with terminals to update and run files and python version. Further, it becomes easier to directly interact with the HPC and make immediate changes when necessary. A more feasible option is the 'Jupyter' notebook, but if the task includes running complex systems as discussed, this should be

more dependent on the user's past experiences with the IDEs so that execution can be carried out in a desirable time and that simplifies the project-building.

3.3.3 Performance benchmarks

Specifically, CoNLL-2014 is a popular dataset to compare benchmarks, so quite a lot of performances are available to compare. Though cLang-8 is relatively new, there have been certain tests, and there are comparisons that can be included in the analysis. Even though it is used in portions and combinations with other datasets, the experiments give vivid analysis, and therefore, there is data of the transformer models, trained on cLang-8 or lang-8 (from which cLang-8 is derived) and CoNLL-2014 datasets, which will be in focus for the project's objective. The project's results can be compared with the existing analysis to obtain an idea of how close are the project's scores to the other scores.

Model	**Precision**	**Recall**	**F0.5**
GEC model [21]	0.6549	0.3221	0.4988
GEC model [28]	0.6677	0.3213	0.6128
GEC model [33]	0.724	0.441	0.65
GEC model [62]	0.739	0.461	0.6677

Table 1: The performances on CoNLL-2014 dataset

Model	**Precision**	**Recall**	**F0.5**
GEC model [39]	0.4120	0.1640	0.3170
GEC model [56]	0.4472	0.2216	0.3716
GEC model [63]	0.7412	0.3630	0.6134

Table 2: The performances on cLang-8 dataset

4 Methodology and implementation

The gt5 model is trained on clang8 dataset and before reaching the results delivered, it has been experimented through prominent sections of the dataset. The purpose of the gt5 model is to reduce the fine-tuning steps and attempt to create the new state-of-the-art GEC that can be easier to implement for future purposes. The base-model is deliberately chosen to be 't5' so that it can provide a good foundation as well as perform better due to its incredible text-to-text framework, scalability and capacity. The understanding of the model is justified by its impressive pre-training on huge amounts of data on C4 dataset or Colossal Clean Crawled Corpus. Later to obtain clearer analysis of model's performance among some past models trained on similar methodology, results are recorded with CoNLL-2014 dataset as well. All the runs are aimed at achieving substantial results for the model performance with the datasets mentioned.

4.1 Initial plan for the project

Essentially, the project schedule requires the following components to be effectively listed and followed. A schedule is formulated on the basis of the identified components and therefore the target is set by the end of July to attain full-fledged finished report with identified results. The remaining week before the deadline which is 6^{th} August, the focus will be on properly identifying the report and correcting any changes if any and verifying proper functioning of the model and packaging of source codes.

The components are as follows: -

1. Data acquisition: Particularly lang-8, for the preparation of the English dataset and the CoNLL-2014 dataset.
2. Model evaluation: -
   - Phase 1: Running and implementing base model or T5 with cLang-8.
   - Phase 2: Optimising the base-model and building gt5 model.
   - Phase 3: Obtaining more results with gt5 and cLang-8 dataset and CoNLL-2014.
   - Phase 4: Drawing final comparison of base-model among other models.
3. Theory development: Reporting the status of aims achieved and updating the recorded results.
4. Software development: Final analysis of the project code and finalising report. Uploading the source codes with appropriate citations in the appropriate repository.

Initially, the data was acquired, on which further processing and fine-tuning were planned. Following is the Gantt chart to have a recollection of project components as a whole.

4.2 Gantt chart

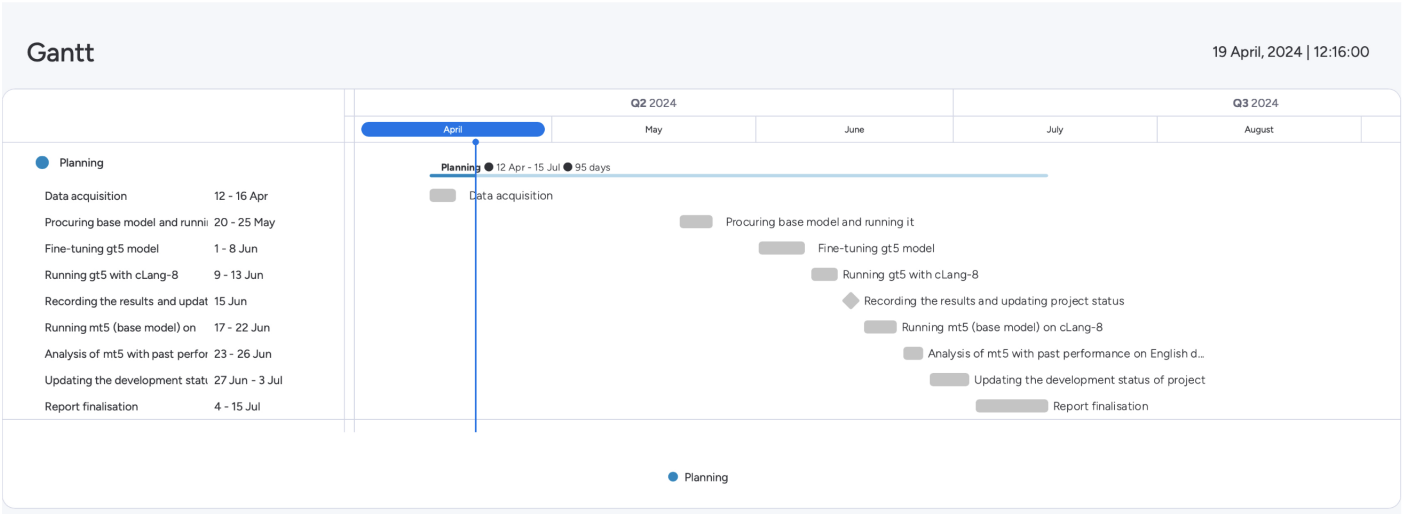

Figure 9: Gantt chart

4.3 Dataset preparation

The cLang-8 dataset is built on lang-8 dataset. It is the more cleaner version of the later. The LANG-8 dataset covers 80 languages and has been corrected by language learners by correcting each other's texts. Even though the distribution of languages is very skewed it includes Japanese and English as the prevalent languages with over a million ungrammatical-grammatical sentence pairs each, and only ten languages in total have more than 10,000 sentence pairs each. But the flip side to a huge amount of data collected is that, given the uncontrolled nature of the data collection, there are many of the examples which contain quite unnecessary paraphrasing as well as erroneous or incomplete corrections.

There are many GECs or grammatical error correction models that have utilised the dataset to produce state-of-the-art GECs. Though if there is the opportunity of the dataset becoming better version of themselves, there are higher chances of building GEC that can be tweaked a little depending on their architecture or functioning to train themselves on the same improved datasets if present.

The clang-8 was created by the authors by the models that were trained utilizing certain strategies to ensure the outputs can contain the updated and corrected sentences not only grammatically but also from the literature perspective [6]. A supervised model is used to output the sentences that are devoid of erroneous input sentences from the Lang-8 dataset. The advantageous aspects of such dataset production was that the word-error rate between the source and target words were reduced dramatically. This is why it was released so new GECs can be implemented using the respective dataset for easier production of the new state-of-the-art grammatical error correction models.

The CoNLL-2014 dataset comprises of the errors produced and ESL writings. Due to the variety of data collected, it has been experimented with for GEC and made a popular dataset to establish benchmarks against models and thus is openly available.

To produce the datasets, the lang-8 is collected and processed to create the CSV format of cLang-8 dataset and from the test set of CoNLL-2014, the dataset is procured in m2 format to be converted into CSV format for further processing.

4.4 Environmental setup

For running the scripts, environment with greater CPU or GPU and storage is essential and since training time for each sections of the experiments is huge, it cannot be run on local machine. For this project the HPC considered is 'CREATE'. The short scripts can be run on the local machine to see if the changes made in the

training parameters or later when implementing the data augmentation are working correctly. If there are errors and debugs, testing the code for 1 epoch and then correcting the mistakes on smaller datasets can be helpful to ensure the new strategy works without bugs and then it can be transferred to HPC for running the larger datasets. Therefore, it was very helpful to reduce the datasets and experiment on first few sentences to debug the code to avoid bugs in the HPC when it failed after long hours of running.

To setup HPC, first there was the need for asking the supervisor to provide the access to the institution's services. Later, creating an 'SSH' key for authentication is needed. Once the authentication is complete, it is possible to log-in to the HPC from the IDE terminal and directly transfer the files.

To run the files, first the python environment was setup which in this case was 'virtualenv' for python 3.8. Then 'jobs' are created to process the files in batch format. After storing the necessary files, setting up the virtual environment there is a need to install the necessary dependent libraries on the HPC terminal and then batches can be executed to run scripts in the virtual environment so that the processing can initiate. The 'jobs' are modified to notify the author when jobs are submitted for running the program and when the program stops running or fails before the completion time due to bugs.

4.5 Model implementation

4.5.1 Training parameters

The base model for the experiment is 't5-base' model from the pre-trained models by Google. There were quite some challenges faced before arriving at accurate checkpoints that were helpful in making the progression in the right direction. The tokenizer used for the system is associated with the base-model chosen for the system to ensure compatibility. The parameter-tuning for the model are as follows: -

1. Learning rate: '5e-5'
2. Number of training cycles processed per device (GPU or TPU): 4
3. Epochs: 3, 6 and 10
4. The dropout rate is set at 0.1 with the model to avoid overfitting when training the data.
5. Checkpoints: The recent 3 ones are to keep and the old ones are deleted. This is to ensure the disk space is not exploited.
6. Weigh_decay: 0.01. Thus penalty term introduced will help preventing overfitting.
7. At last it will load the best model for further phases of the algorithm.
8. Optimizer is set to AdamW to expect improvements in the results from model training.

4.5.2 Complete implementation of parameters in the model

The parameters utilised for the model are essential to find a balance between computational resources available and model training time and expected performance output. They are set for the purpose of the project given the available choices as follows: -

1. Padding of max-length to ensure all sequences are padded to same length and there is uniformity in the data.
2. Truncation is set to 'longest_first' ensuring the most relevant information is retained.
3. The maximum length of input and output sequences is set to 512. This is to make sure that no tokens are missed but requires more computation compared to shorter values.
4. Evaluation strategy is set to 'steps' to track the checkpoints produced. This is chosen as better alternative than 'epoch' to delete the previous checkpoint models and storing the latest 3, better performing ones. Though requiring more computation is space-friendly and avoid crashing of program due to storage issues.

5. Learning rate is set to '5e-5' suitable for the size of dataset being used. It determines the step size at each iteration while moving toward a minimum of the loss function.

6. Number of training samples per batch per device is kept at 4.

7. Number of epochs to be tested with variations between 3,6 and 10 to gather enough results for comparison. The larger the number of epoch, more time it consumes to execute.

8. 'weigh_decay' parameter set to 0.01 to prevent overfitting by adding penalty for larger weights.

9. Checkpoints for model are saved every 1000 steps.

10. Data collator, 'DataCollatorWithPadding' is used for this model to pad sequences to the same length in a batch, ensuring they can be processed simultaneously.

11. Optimizer is set to 'AdamW' for better performance and generalization combining the benefits of 'Adam' optimizer and weight decay.

```
# Defining training arguments
training_args = TrainingArguments(
    output_dir='./results_clang_10epc', # Store the results for epochs 3,6 and 10
    respectively.
    evaluation_strategy='steps',
    learning_rate=5e-5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=10,  # Can experiment with epochs, diverse results expected with 3,6 and
     10.
    weight_decay=0.01,
    save_steps=1000,  # Saving checkpoint every 1000 steps
    save_total_limit=3,  # Limit the total amount of checkpoints. Deletes the older
    checkpoints.
    logging_dir='./logs_clang8_tuned',  # Directory for storing logs, can be renamed for
    different epochs to store data.
    logging_steps=1000,  # Log every 1000 steps
    load_best_model_at_end=True,  # Load the best model when finished training
)
```

Listing 2: Hyperparameters applied after tuning

4.5.3 Inference pipeline

The GEC pipeline formalises the production of the predictions from the model after it finished training and is named 'gec-pipeline' in the project. This is for text-correction. From the inputs of the unseen validation dataset which are from the 'Column1' in the respective dataset, predicted outputs are generated which are the grammatically correct versions of the sentences. Following this, the references are extracted in a file to store them for further analysis which is the 'Column2' or the corrected sentences from the validation set. Following is the simple way of presenting the steps involved: -

1. Formatting: The pipeline prepares the text for the model by tokenizing it (breaking it down into smaller units).

2. Model Prediction: The formatted text is fed into the trained T5 model, which generates the most probable corrected text. The limit to generating output sequence is set at 1. The 'max_length' for tokens is 512 to ensure even the largest sentences are allowed as inputs and properly processed to produce accurate corrections.

3. Output Processing: The pipeline retrieves the generated text from the model and potentially performs further processing before returning it.

```
# ------- BUILDING THE GEC PIPE-LINE
gec_pipeline = pipeline(task='text2text-generation', model=model, tokenizer=tokenizer)
# Save predictions and references
predictions = gec_pipeline([example['Column1'] for example in eval_dataset], max_length=512,
    num_return_sequences=1)
pred_texts = [prediction['generated_text'] for prediction in predictions]
references = [example['Column2'] for example in eval_dataset]
# ------
```

Listing 3: GEC pipeline for the gt5 model

The pipeline setup includes initializing a text-to-text generation pipeline with the trained T5 model and its tokenizer. The 'pipeline' is a high-level API from 'Huggingface' for easy use of pre-trained models. Specifically the task is specified by 'text2text-generation'. It is appropriate for models to take in input and generate the corresponding output text. In order to extract predictions of generated texts or 'pred_texts', the incorrect sentences are passed through pipeline that has the limit of generated output tokens of 512. 1 output sequence is generated per input sentence. The references are extracted after these steps which records the correct sentences from the evaluation dataset.

4.5.4 Deployment

In summary, the overall system performs the following way as discussed in parts before: -

1. Loading the dataset in CSV format.
2. Pre-processing the data, an important step to tokenize the original and corrupted sentences both. Padding and truncation are applied to ensure all the sequences have same length to ensure smoother processing and minimise computational errors.
3. Loading the pre-trained T5 model for conditional generation from a checkpoint.
4. Initialise the Trainer object with model, tokenizer, arguments, dataset and data collator. Later the model is trained on training dataset.
5. For the evaluation purposes, the text-to-text generation pipeline is used by the model which generates the corrected sentences and then stores the records in files to be evaluated on later.
6. Saving the results and using Errant-toolkit to calculate F0.5 score on the saved text files.

Initially the experimentation was carried out on small section of dataset to look through flaws and advantages of the code and understand the possible challenges that can arrive. Such holes in the system can be covered using such approach and thus is more convenient on the local machine. But in order to process the larger dataset, the high-performance computing played a huge role. Therefore, the allowance provided by the university was sufficient to bridge the gap of need and thus processing on a larger scale was possible. The HPC or high-performance computing requires student access provided by the supervisor. On the HPC, in order to produce and store the results for further analysis, it was very important that the steps are carried out carefully planned.

1. 1. Therefore, the virtual environments were applied, primarily the 'virtualenv' initiates the environment 'run_prog' to produce the processing environment for the transferred scripts that were needed to run and also to store the necessary libraries and modules.
2. The jobs were created to process the scripts on nodes and output results.
3. The jobs were modified to notify the user when queueing for the process began and ended and whether the processing was a success or had bugs. This was helpful to immediately make changes to the code when the problem arose without any loss of unnecessary time.

The processing power made the execution much more feasible by reducing the time-span in hours and processing of 4 hours of script on local machine reduced to 2 hours on the HPC.

The model was supposed to provide the state-of-the-art performance given the architecture applied to it but it has resulted in a decent baseline model. Though this was the best experiment that turned out for successful production of the system, it provided possibilities for experimenting on cLang-8 dataset specifically which is quite new to the field of natural language processing. To provide a better comparison, later the same model is tested on CoNLL-2014 test dataset to provide F0.5 scores that can be used or relevant comparison.

4.6 Evaluation method

The evaluation for the project is not by traditional metrics like F1-score, accuracy but rather the outputs as predicted texts which are generated by the GEC and reference texts that are the correct versions of input texts, are used to calculate Errant F0.5 score. To process this, the needed text files including the incorrect text files containing all the incorrect input sentences from the dataset, predictions from the GEC and the 'references' file which contains the corrected sentences from the dataset are obtained by running the main file 'Gecmodel.py' which trains the model. The evaluation is done on the neighbouring script because it turned out to be an easier method to obtain the necessary files and then evaluate and store the results in an organized manner. Essentially, it is the same process just divided into 2 parts for convenience.

The next file to be used for evaluation is 'Gecmodel_errEval.py' [A.13] which runs Errant for final evaluation and saving results. This is done by first achieving the predictions, then using the files individually for incorrect and correct sentences to generate the needed m2 format of files references, and predictions. The last stage involves comparing the two m2 files to produce the final scores for analysis. The effective way to run the analysis is to use the 'subprocess' module in the script to run the terminal commands, and all the files that are processed are saved, including the m2 formats of references and predictions and the final score, which includes precision, recall and F0.5 that are saved in a separate text file. After all the scores are attained through diverse kinds of runs, analysis can be done on what fine-tuning steps or data augmentation techniques have proven to be useful to achieve more accurate translations of the input sentences into the grammatically correct ones.

```
# Running the ERRANT commands
subprocess.run(
    ["errant_parallel", "-orig", "Err_files_evals/clang8.incorrect", "-cor", "Err_files_evals/predictions.txt", "-out",
     "Err_files_evals/results.txt"],
    capture_output=True,  # Capture stdout and stderr
    text=True  # Return the output as a string
)
# subprocess.run(["errant_m2", "-silver", "Err_files_evals/references.txt", "-out", "Err_files_evals/ref_m2.txt"],  capture_output=True, text=True)
subprocess.run(
    ["errant_parallel", "-orig", "Err_files_evals/references.txt", "-cor", "Err_files_evals/predictions.txt", "-out",
     "Err_files_evals/ref_m2.txt"],
    capture_output=True,  # Capture stdout and stderr
    text=True  # Return the output as a string
)
output = subprocess.run(["errant_compare", "-hyp", "Err_files_evals/results.txt", "-ref", "Err_files_evals/ref_m2.txt"],
                        capture_output=True,  # Capture stdout and stderr
                        text=True  # Return the output as a string
                        )
```

Listing 4: Evaluating through 'subprocess' function the m2 files for final scores

4.7 Challenges encountered during fine-tuning and implementing the model training

The dataset finally produced had the incorrect and corrected sentences for the model implementation. Therefore the model trained on the based model t5 and clang-8 dataset had few shortcomings to be addressed before

arriving at the stage where the predictions from the pre-processed dataset can be extracted for finally evaluating it.

1. Choosing the evaluation metric for the model, though it already utilises evaluation loss for the model, deciding the metric was imperative. Therefore, getting to Errant-score was challenging initially.

2. More debugging involved solving the shape of the inputs problem. The inputs for processing were resulting in inconsistent shapes and usually, the input data was resulting in object data-type or arrays with inhomogeneous shape but those problems were solved by making changes to the hyper-parameters, pre-processing and later by properly extracting the outputs in the result files. For instance, printing intermediate results like shapes of tensors helped in identifying issues during training.

3. Running the errant score was challenging initially because of limited understanding how to compute the files. But the module itself was used to process the necessary files in m2 format for final phase of processing.

4. The HPC also proved to be a little struggle before enabling proper execution of the python scripts. The job scripts required proper specification of nodes, queueing notification to the author and specifying the address to store the important outputs of model training. The print statements are ineffective in the HPC therefore shorter debugging processes had to be carried out on local machine with smaller sections of dataset to manage the computing power.

5 Results, analysis and evaluation

The objective of the overall project was to attempt in establishing a state-of-the-art GEC that can provide convincing results in reduced fine-tuning steps. Therefore, the experiments that have been performed begin from the cLang-8 dataset and go on to CoNLL-2014 to give a vivid idea of the potential of the grammatical error correction model. The system's first version is very simple with newly implemented trainer classes with appropriate parameters, and the GEC pipeline to extract predictions. But in order to properly build a system, it requires the important components and therefore many versions with different combinations of epochs of the GEC model give a better idea of which version is most potent and how well it can adapt to either of the chosen datasets.

The Errant scores are recorded by comparing the 'm2' format of references and predictions, where references are the correct versions of the input sentences used to produce the sentences in the predictions file.

Precision: $\frac{TP}{TP+FP}$

Recall: $\frac{TP}{TP+FN}$

F0.5: $\frac{(1+0.5^2)\cdot\text{Precision}\cdot\text{Recall}}{0.5^2\cdot\text{Precision})+\text{Recall}}$

5.1 Results

The results are obtained from specific cases that have had a considerable impact. The epochs to be considered are 3, 6 and 10. When set for model evaluation, these epochs produced results that were helpful to the overall project analysis. In other cases, when tweaking the hyper-parameters, the results were distorted, and performance failure was more than predicted. This might be due to improper data augmenting techniques applied to the system or implementing higher learning rates over the dataset's medium size, which could not produce tangible results. However, these errors could be corrected by running the scripts with the necessary modifications.

Initially the model was tested on smaller versions of the dataset to debug and improve base performance before moving on to final datasets. Following is the most prominent kind of result and is the nearest to almost all the failed experiments. The results for CoNLL-2014 dataset were: -

- Precision: 0.1232
- Recall: 0.1165
- F0.5: 0.1218

This was specifically during the 10 epochs training because the lower epochs did not provide clear errors in the scores. The problem in the results seem to be increased detection of false negatives of 1092 and therefore the overall score has changed drastically. This has not happened yet to the performance of the model. However, the cLang-8 dataset performed considerably well: -

- Precision: 0.7157
- Recall: 0.3364
- F0.5: 0.584

But this alone cannot justify the model performance on large datasets. Moreover, there has to be consistency among the scores on each of the datasets of the actual intended size. Only this can ensure that the model is performing closer to the obtained scores in actuality and the results are consistent.

There are practically no changes in the hyper-parameters except changing the evaluation strategy to 'steps'. This is done to ensure there is enough space in the disk when running the algorithm and to avoid crashing the script in between training and losing the progress. Therefore, though the script now still records the model

checkpoints after 1000 steps, only the recent 3 and best-performing ones are stored. At the end, the system will load the best-performing model to run the predictions over the incorrect input sentences. The next step to improvement is data augmentation.

A drop-out rate of '0.1' is added. This is just to make sure there is no overfitting on the model. The data augmentation section of the code includes introducing random errors by randomly choosing to either insert, swap, delete or replace the words in the sentences. This experiment will provide a good reference point for further needed changes if any. The scores recorded are nearly good, but certain improvements are visible, and therefore, more improvements, if possible, on the datasets can show promising improvements. At last, after all the test cases, collective scores can help in better understanding the difference between the runs intuitively. Following are the test cases and their runs on the chosen datasets for the project. After the span-based correction scores, the token-based correction scores will be listed for deeper analysis.

5.1.1 Test-case 1

Initially, set to test the model for 3 epochs on the cLang-8 dataset, the model shows promising results and therefore the next steps on the algorithm were taken in the hopes of increasing the performance from this point of reference. The score was good and the parameters set for the model for all epochs were: -

1. Learning rate: 5e-5
2. Batch size: 4
3. Weight decay: 0.01

The scores for the run was: -

- Precision: 0.3212
- Recall: 0.2106
- F0.5: 0.2907

The score expected could be the result of the dataset's simplicity and its speciality to adapt with almost all kinds of base models to produce the results. Therefore, the next step would be to introduce the algorithm on the CoNLL-2014 dataset.

Therefore, for the CoNLL-2014 dataset the results recorded are as follows: -

- Precision: 0.4643
- Recall: 0.1159
- F0.5: 0.2899

The score is nearly good but certain improvements are visible and therefore more improvements can be attempted.

5.1.2 Test-case 2

For increasing the epochs to 6, the following results are obtained. The results for CoNLL-2014 dataset are: -

- Precision: 0.3779
- Recall: 0.0846
- F0.5: 0.2232

The results on cLang-8 for the same epochs are: -

- Precision: 0.2656

- Recall: 0.2269
- F0.5: 0.2568

Though the scores have dropped than the initial cases, there is a chance of consistency among the scores if more epochs are tested. This can ensure that the system is at least sustaining itself in terms of performance across the datasets.

5.1.3 Test-case 3

In the last case of 10 epochs, the maximum epochs possible are attempted to verify the results. Given the limited time and resources, concluding this experiment can still provide analysis of the model performance on the datasets and how the changes have affected the model performance across the range of epochs.

The results on the CoNLL-2014 dataset: -

- Precision: 0.3003
- Recall: 0.1255
- F0.5: 0.2349

The scores for cLang-8 dataset are: -

- Precision: 0.1719
- Recall: 0.2087
- F0.5: 0.1782

5.2 Analysis

5.2.1 Metric interpretation

The training loss for all the above-discussed cases has been very close to 0.2-0.4 in all epochs. The slow and steady decrease indicates promising performance and steady convergence. The improvement in predictions is evident in [A.3] for the cLang-8 dataset with a training loss of 0.43. The training loss is 0.5 [A.4] in the CoNLL-2014 dataset which is the best case. The evaluation loss for the datasets cLang-8 and CoNLL-2014 of best-case model are 0.05 [A.3] and 0.01 [A.4], inferring the decent performance of the model on the unseen data.

The training time considering the sizes of datasets has been low for low epochs for obvious reasons. But the batch size is unchanged and choosing to keep only the recent 3 best-performing models did not abnormally increase the training time. Therefore for the maximum epochs, i.e. 10 the training time was 6.5 hours on HPC or 'CREATE'. Therefore, this shows that the model was not stuck in the training duration and the convergence was successful.

The difference in the epochs set for the datasets and algorithm gives clear idea that the model's performance was maximum and the parameters were well-implemented to get the desirable scores. Though there seems to be room for improvement, the algorithm appears to have sufficient components for the intended results. This puts the model in the base-model hierarchy as a sufficient and reliable base model if not one of the best-performing ones on the same datasets [3].

Model	CoNLL-2014	cLang-8
Test-case 1	0.2899	0.2907
Test-case 2	0.2232	0.2568
Test-case 3	0.2349	0.1782

Table 3: Span-based F0.5 scores of gt5 model

Therefore, the F0.5 scores to be considered given the set of obtained scores is from the minimal epochs (3 epochs) in both datasets. There is minimal difference in later epochs on CoNLL dataset and cLang-8 being

more straightforward in terms of accurate data of English sentences reduces the performance with an increase in epochs which can infer towards the model losing its accuracy over the larger training epochs given the size and complexity of the same dataset. To avoid misinformation on the model's performance on the datasets, rather than taking the average, the best epochs are considered as the best performance by the model and how well it performs on unseen data specifying its generalizing ability. Given the data complexity, model architecture and hyperparameter settings, the best scores are 0.2907 and 0.2899 for cLang-8 [A.1] and CoNLL-2014 [A.2] datasets, respectively. The less epochs training suggests that for the given model, the dataset was enough to be trained and evaluated on less epochs given the parameters are tuned right and data augmentation is implemented to build the sophistication of the overall system.

Even though the past comparison tables provided [1] and [2] likely encompasses a combination of span and token-level evaluations, the token-based scores [A.9] [A.10] suggest that the model performs robustly at the token level and is competitive with these benchmarks [A.5] [A.6]. The full comparison of all test cases is given in the Table in [4].

Model	CoNLL-2014	cLang-8
Test-case 1	0.5524	0.6886
Test-case 2	0.3696	0.4565
Test-case 3	0.5633	0.373

Table 4: Token-based F0.5 scores of gt5 model

5.2.2 Model components impact

The hyper-parameters were beneficial to the system and any changes did not yield better results and the ones implemented were reliable. The learning rate '5e-5' served as the perfect component as the higher ones are better for excessively large amounts of datasets, though not so efficient for this particular project. The batch size was perfect not to consume a large amount of duration for training but rather was sufficient even on the HPC that was 'CREATE'. Except for the small bugs which were easily solved after some trial runs, the main code could run smoothly to evaluate the given dataset.

The data augmentation, however, had variable results. The data augmentation when introduced to the dataset CoNLL-2014 yielded better performance than before because of improved generalising ability and expansion of data. The false positives were largely reduced and the score increased quite a lot from base model that consisted of just the basic parameters. This is a common technique and is still very potent for the given purpose of the project. Therefore, when applied to cLang-8 dataset it showed promising results. With a greater strategy, more data from the cLang-8 dataset can be processed efficiently without a decrease in the model's performance. However, the technique was applied while ensuring that it is sufficient to improve the model's performance and that it provides sufficient results in the limited duration available for the testing of the system.

5.3 Evaluation

5.3.1 Comparative analysis

Errant records the errors corrected by the GEC model, and most of the focus of the particular model is on correcting the types of errors among the words specifically. The best-case model for both datasets returns a relatively good and decent F0.5 score for token-based correction. Therefore, the system shows promising signs of improvement for future works for complex data of sentences but is rather good at performing corrections on the word level. The scores for cLang-8 and CoNLL-2014 dataset are 0.5524 [A.5] and 0.6886 [A.6]. However, the main concerning F0.5 scores (span-based correction) that show the overall performance of the system through sentence-wise correction by the model on the respective datasets was a little less than the benchmark models, which are 0.2907 and 0.2899 [3]. But overall, it seems fair to judge the generalizing ability of the model given the dataset size and computational resources to be quite decent.

From the given examples in the Tables in [5] and [6], the system shows improvement towards serving as a base model. All the experiments were conducted given the time duration available and resources available and

therefore the GEC could be produced. Even though it is not close to extremely well-performing base models, it can be considered a decent base model that can be developed further on the given datasets though its existing version is capable of providing a decent score. The data augmentation section of the code called 'introduce_errors' function randomly chooses to either insert, add, replace or delete certain words in the input sentences in order to compel the model to adapt better at recognizing patterns and discerning accurate predictions to the inputs. But given the predictions made, it is capable of making grammatical corrections and also producing more meaningful sentences. The errors in sentences are corrected not only grammatically, punctuation-wise and sentence style-wise but also by retaining the meaning behind the sentence. The corrected sentences are left as it is. The reason for using 'Errant' as metric is also to retain the meaning behind the produced sentences and simultaneously correcting sentences when outputting scores for the model-built. The system seems to produce such corrects to a decent degree.

Corrected sentences	**Input sentences**
In addition , a woman testing positive for BRCA1 would be expected to allow her brother or sister to disclose such information to a niece or nephew rather than herself , although this may be further complicated by an individual 's response to risk information as they do not want to know .	In addition , a woman testing positive for BRCA1 would be expected to allow her brother or sister to disclose such information to a niece or nephew rather than do it herself , although this may be further complicated by an individual 's response to risk information as they not want to know .
The PV of my best friend 's band is now up at last !	The PV of the song of my best friend 's band is up at last !
I think what he meant was that " love " is the feeling that makes the moon look brighter than the moon you see by yourself .	I think what he meant was that " love " is the feeling that makes the moon looks brighter than the moon you see by yourself .

Table 5: Corrected sentences from cLang-8 dataset

Corrected sentences	**Input sentences**
People that are living in the modern world really can not live without the social media sites like Twitter and Facebook .	People that living in the modern world really can not live without the social media sites like Twitter and Facebook .
However , I thought that the genetic information is one person 's personal privacy which should be protected by the law .	However , I thought that the genetic information is one person 's personal privacy which should be protected by the law .
To utilize the technology well , we should do our best to not only balance the pros and cons , but also lengthen the pros and shorten the cons .	To utilize the technology well , we should do our effort to not only balance the pros and cons but also lengthen the pros and shorten the cons .

Table 6: Corrected sentences from CoNLL-2014 dataset

5.3.2 Future improvements

Further improvements in the system can be expected in terms of the size of the dataset. The dataset processed is enough for the system to be processed on the provided HPC. Though, with more resources and time if provided, further experiments can be conducted to understand whether the given system has potential to increase its performance and if so then at exactly which kind of errors. Though the experiments conducted were performed given the limited time and resources, it has given a proper idea for the kind of approach that must be taken for the objective considered for the project. It was sufficient to spot the kind of errors that the system was capable of examining and correcting. Though more amount of techniques can be potentially applied

to ensure that more kinds of errors are corrected and this can be better contribution to the overall project. If this approach is combined with training the system on larger sections of the dataset with more resources available then the performance can be improved given more experiments are possible to analyse the steady progress of such a project.

Some research directions must be given to the project, which can include techniques not applied in the project to potentially increase its performance while sustaining its progress.

- For improved error-correction outcomes, the technique to train the model on n-gram language models (considering sequences of n words) that capture the surrounding context can be considered because this helps identify grammatically correct options based on how words typically co-occur.
- Analyse the grammatical relationships between words in the surrounding text. This allows the model to understand how different parts of speech interact and choose corrections that maintain these relationships.
- Identifying and categorizing the named entities such as people, places, organizations, etc in the text can be a huge help to the overall system. This can help ensure subject-verb agreement and pronoun resolution based on the entity type which can help in discerning more contextual information surrounding the language being processed.

Some of the discussed techniques add to the complexity of the models, further increasing the time duration for training, but they show promising improvements and are among the most researched techniques that share the project's objective.

6 Conclusion

This study aimed to evaluate the performance of a GEC model called gt5 on the English datasets cLang-8 and CoNLL-2014. Through rigorous experimentation and analysis, a demonstration was possible on how well the model can perform on training datasets and correcting the input errors such as grammar, punctuation, and style and produce accurate grammatically corrected outputs which are compared to reference or correct sentences from the relevant datasets. Though the aim was to produce the base model capable of scoring F0.5 of 0.3-0.5, the model falls short to just near 0.3. However, the model has potential as it is quite adept at correcting errors on word level (token-based correction). Span-based or sentence-level corrections require understanding the broader context and making coherent changes across multiple tokens, which is more challenging than token-level corrections.

These limitations in results for complex sentences could be due to the limited dataset for implementation and time and computational resources. The training time was made efficient by using cloud platform and experiments for the initial aims of achieving some results on the basic datasets were made possible. The model aimed at producing the algorithm that can provide some clarity on the requirements for producing GEC on which further work can be done. Therefore, the application of hyper-parameter tuning and data augmentation was imperative for the chosen datasets. More improvements in the model can be expected on larger datasets and larger available resources for upgrading the GEC system.

The findings indicate that the model is not the best but significant at correcting entire spans of text with F0.5 scores of 0.2907 and 0.2899 for datasets cLang-8 and CoNLL-2014 respectively. This indicates its overall correction performance among whole sentences. However, the model was able to perform comparatively well on token-based correction or specifically performing the corrections at word-level with F0.5 scores of 0.552 [A.5] and 0.688 [A.6] for datasets cLang-8 and CoNLL-2014. This indicates strong performance in detecting and correcting individual erroneous tokens with good precision. These scores can put the model is the competitive hierarchy to past performances and surpass some state-of-the-art GECs. This kind of achievement on the cLang-8 dataset has been good starting point, for it being one of the newest datasets to be experimented on in the research field. These results contribute to the ongoing research in GEC by implementing the data augmentation techniques and specifically the one applied in the project is a 'noisy channel simulation' for the data augmentation which essentially implements real-world errors in the sentences to create synthetically erroneous data that can help the GEC model learn to identify and correct different error patterns and testing the generalising ability of the overall model. The amount of epochs expected to breed the best results is directly proportional to the complexity and size of the dataset and not necessarily will large epochs produce best results. The hyper-parameters including the applied learning rate in complementary with the chosen drop-out rate produce the most effective model which converges better than others. Therefore, the best cases were recorded to finalise the best-performing models and the subsequent checkpoints for the model.

While the model exhibited promising results, several challenges were encountered, including reducing the training time when training the model, improving the generalising ability of the model through the techniques applied like data augmentation, implementing the GEC pipeline to extract predictions from the model trained and obtaining F0.5 score on final results through Errant score. Addressing these limitations potentially enhanced the model's overall performance to a decent degree. Future research could explore testing on much larger datasets and computational resources, implementing more advanced techniques to improve model generalisation on larger amounts of unseen data and trying out newer base models to have more boost on the overall project development. While the span-based correction is relatively lower, it still demonstrates the system's ability to identify and correct a significant portion of grammatical errors. The future work, as discussed in Section [5.3.2], can be focused on improving the span-based performance of the respective model.

Overall, this study provides valuable insights into the capabilities and limitations of GEC models. By building upon these findings, the field can develop more accurate and effective GEC systems for various applications.

7 Legal, social, ethical and professional issues

The code of conduct during the development phases of this project has been per the BCS (British Computing Society) code of conduct. This highlights the ethics that must be followed in the practices associated with IT or computing science. The rule is most relevant and sincerely followed: '1. Public Interest: You shall:...b. have due regard for the legitimate rights of Third Parties;' [17].
The rule is for safeguarding and protecting intellectual property and has been carefully followed in the phases of development. The rule ensures that all the supports are carefully acknowledged and ethics have been followed in all the phases of the project's progress. All the tools and libraries used for the development of the project are open source and specifically for the purpose. All the references are sincerely referenced and the ideas are procured to build the project's strong foundation on the openly available methodologies. The use of third-party code, ideas and information is accepted within the field of computer science in the interests of progress and to build on the work already compiled by previous individuals. These are also duly specified in the research section and all the respectable authors are credited for the research materials they made available. It is imperative to give due credit to the respective third parties and ensure there is no infringement on their intellectual property rights. In this regard, a significant amount of care has been taken to source and reference each idea, code, and information prepared by third parties so that I do not claim ownership of the work performed by others. I declare that no part of this submission has been generated by AI software. These are my own words.

8 References

[1] "BERT. *huggingface.co*. URL: https://huggingface.co/docs/transformers/en/model_doc/bert.

[2] "Deploying. *a Seq2Seq Model with TorchScript — PyTorch Tutorials 2.3.0+cu121 documentation, " pytorch.org*. URL: https://pytorch.org/tutorials/beginner/deploy_seq2seq_hybrid_frontend_tutorial.html.

[3] "Documentation. *www.overleaf.com*. URL: https://www.overleaf.com/learn.

[4] "Exploring. *Transfer Learning with T5: the Text-To-Text Transfer Transformer, " research.google*. URL: https://research.google/blog/exploring-transfer-learning-with-t5-the-text-to-text-transfer-transformer/.

[5] "google-research-datasets/clang8. *GitHub, Jun. 07, 2024*. URL: https://github.com/google-research-datasets/clang8.

[6] "google-research-datasets/clang8. *GitHub, Jun. 07, 2024*. URL: https://github.com/google-research-datasets/clang8#.

[7] "google-t5/t5-base. · *Hugging Face, " huggingface.co*. Mar. 5, 2024. URL: https://huggingface.co/google-t5/t5-base.

[8] "Improving. *Grammatical Error Correction via Contextual Data Augmentation, " arxiv.org*. URL: https://arxiv.org/html/2406.17456v1.

[9] "Machine. *Learning Mastery.*" Machine Learning Mastery. 2016. URL: https://machinelearningmastery.com/.

[10] "NAIST. *Lang-8 Learner Corpora, " sites.google.com*. URL: https://sites.google.com/site/naistlang8corpora

[11] "Papers and Code. *T5 Explained, " paperswithcode.com*. URL: https://paperswithcode.com/method/t5.

[12] "Shared. *Task: Grammatical Error Correction, " Nus.edu.sg*. 2014. URL: https://www.comp.nus.edu.sg/~nlp/conll14st.html.

[13] "T5. *A Lazy Data Science Guide, " mohitmayank.com*. URL: http://mohitmayank.com/a_lazy_data_science_guide/natural_language_processing/T5/.

[14] "The. *Basics of Language Modeling with Transformers: BERT — Emerging Technologies, " etc.cuit.columbia.edu*. URL: https://etc.cuit.columbia.edu/news/basics-language-modeling-transformers-bert.

[15] K. Aitken. *Understanding How Encoder-Decoder Architectures Attend*. Accessed: Jul. 02. 2024. URL: https://arxiv.org/pdf/2110.15253.

[16] A. Anand et al. "GEC-DCL: Grammatical Error Correction Model with Dynamic Context Learning for Paragraphs and Scholarly Papers". In: *Lecture notes in computer science* (Jan. 2023), pp. 95–110. DOI: 10.1007/978-3-031-49601-1_7.

[17] BCS. "BCS, THE CHARTERED INSTITUTE FOR IT CODE OF CONDUCT FOR BCS MEMBERS". June 2022. URL: https://www.bcs.org/media/2211/bcs-code-of-conduct.pdf.

[18] C. Bryant. *chrisjbryant/errant*. GitHub, Mar. 02. 2024. URL: https://github.com/chrisjbryant/errant.

[19] C. Bryant et al. "Grammatical Error Correction: A Survey of the State of the Art". In: *Computational Linguistics* (June 2023), pp. 1–59. DOI: 10.1162/coli_a_00478.

[20] W. Chan et al. *KERMIT: Generative Insertion-Based Modeling for Sequences*. arXiv.org, Jun. 04. 2019. URL: https://arxiv.org/abs/1906.01604.

[21] S. Chollampatt and H. T. Ng. *Neural Quality Estimation of Grammatical Error Correction*. ACLWeb, Oct. 01. 2018. URL: https://aclanthology.org/D18-1274/.

[22] S. Chollampatt, K. Taghipour, and H. T. Ng. *Neural Network Translation Models for Grammatical Error Correction*. arXiv.org, Jun. 01. 2016. URL: https://arxiv.org/abs/1606.00189.

[23] M. R. Costa-jussà, Á. Nuez, and C. Segura. "Experimental Research on Encoder-Decoder Architectures with Attention for Chatbots". In: *Computación y Sistemas* 22.4 (Dec. 2018). DOI: 10.13053/cys-22-4-3060.

[24] J. Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. arXiv.org, Oct. 11. 2018. URL: https://arxiv.org/abs/1810.04805.

[25] X. Ding et al. "HPC-GPT: Integrating Large Language Model for High-Performance Computing". In: *arXiv (Cornell University)* (Nov. 2023). DOI: 10.1145/3624062.3624172.

[26] e-Research. *Acknowledging - King's College London e-Research*. docs.er.kcl.ac.uk. URL: https://docs.er.kcl.ac.uk/CREATE/acknowledging/.

[27] M. Felice and T. Briscoe. *Towards a standard evaluation method for grammatical error detection and correction*. Association for Computational Linguistics, 2015. Accessed: Jul. 16. 2024. URL: https://aclanthology.org/N15-1060.pdf.

[28] R. Grundkiewicz and M. Junczys-Dowmunt. *Near Human-Level Performance in Grammatical Error Correction with Hybrid Machine Translation*. arXiv.org, Apr. 16. 2018. URL: https://arxiv.org/abs/1804.05945.

[29] R. Grundkiewicz, Marcin Junczys-Dowmunt, and K. Heafield. "Neural Grammatical Error Correction Systems with Unsupervised Pre-training on Synthetic Data". In: *Edinburgh Research Explorer (University of Edinburgh)* (Jan. 2019), pp. 252–263. DOI: 10.18653/v1/w19-4427.

[30] JetBrains. *PyCharm*. JetBrains. 2019. URL: https://www.jetbrains.com/pycharm/.

[31] R. Kaljahi et al. *Syntax and Semantics in Quality Estimation of Machine Translation*. Accessed: Jul. 02. 2014. URL: https://aclanthology.org/W14-4008.pdf.

[32] M. Kaneko et al. *Encoder-Decoder Models Can Benefit from Pre-trained Masked Language Models in Grammatical Error Correction*. arXiv.org, May 31. 2020. URL: https://arxiv.org/abs/2005.00987.

[33] Satoshi Kiyono et al. "An Empirical Study of Incorporating Pseudo Data into Grammatical Error Correction". In: *arXiv (Cornell University), Sep* (2019). DOI: 10.48550/arxiv.1909.00502.

[34] M. Kobayashi, M. Mita, and M. Komachi. *Large Language Models Are State-of-the-Art Evaluator for Grammatical Error Correction*. arXiv.org, May 26. 2024. URL: https://arxiv.org/abs/2403.17540.

[35] T. Kudo and J. Richardson. "SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing". [cs]. Aug. 2018. arXiv: 1808.06226. URL: https://arxiv.org/abs/1808.06226.

[36] S. Lai et al. *Type-Driven Multi-Turn Corrections for Grammatical Error Correction*. arXiv.org, Mar. 17. 2022. URL: https://arxiv.org/abs/2203.09136.

[37] H. Lamba. *Intuitive Understanding of Attention Mechanism in Deep Learning*. Medium, May 09. 2019. URL: https://towardsdatascience.com/intuitive-understanding-of-attention-mechanism-in-deep-learning-6c9482aecf4f.

[38] G. Lample and A. Conneau. "Cross-lingual Language Model Pretraining". [cs]. Jan. 2019. arXiv: 1901.07291. URL: https://arxiv.org/abs/1901.07291.

[39] J. Lichtarge, C. Alberti, and S. Kumar. "Data Weighted Training Strategies for Grammatical Error Correction". In: *Transactions of the Association for Computational Linguistics* 8 (Oct. 2020), pp. 634–646. DOI: 10.1162/tacl_a_00336.

[40] Y. Liu. "Fine-tune BERT for Extractive Summarization". [cs], Sep. 2019. arXiv: 1903.10318. URL: https://arxiv.org/abs/1903.10318.

[41] Y. Liu et al. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv.org, Jul. 26. 2019. URL: https://arxiv.org/abs/1907.11692.

[42] M. AL-Ma'amari. *NLP — Sequence to Sequence Networks— Part 2—Seq2seq Model (EncoderDecoder Model)*. Medium, Apr. 22. 2021. URL: https://towardsdatascience.com/nlp-sequence-to-sequence-networks-part-2-seq2seq-model-encoderdecoder-model-6c22e29fd7e1.

[43] N. Madnani, J. Tetreault, and M. Chodorow. *Exploring Grammatical Error Correction with Not-So-Crummy Machine Translation*. ACLWeb, Jun. 01. 2012. URL: https://aclanthology.org/W12-2005/.

[44] M. McCloskey and N. J. Cohen. *Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem*. ScienceDirect, Jan. 01. 1989. URL: https://www.sciencedirect.com/science/article/abs/pii/S0079742108605368.

[45] H. Ng et al. "The CoNLL-2014 Shared Task on Grammatical Error Correction". In: *Association for Computational Linguistics* (2014). URL: https://aclanthology.org/W14-1701.pdf.

[46] K. Omelianchuk et al. "GECToR – Grammatical Error Correction: Tag, Not Rewrite". In: *arXiv (Cornell University)* (May 2020). DOI: 10.48550/arxiv.2005.12592.

[47] C. Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *Journal of Machine Learning Research* 21 (2020), pp. 1–67. URL: https://www.jmlr.org/papers/volume21/20-074/20-074.pdf.

[48] S. Rothe, S. Narayan, and A. Severyn. "Leveraging Pre-trained Checkpoints for Sequence Generation Tasks". In: *Transactions of the Association for Computational Linguistics* 8 (July 2020), pp. 264–280. DOI: 10.1162/tacl_a_00313.

[49] K. Schrempf. *Horner Systems: How to efficiently evaluate non-commutative polynomials (by matrices)*. arXiv.org, Oct. 03. 2019. URL: https://arxiv.org/abs/1910.01401.

[50] Z. Shen et al. *Efficient Attention: Attention With Linear Complexities*. openaccess.thecvf.com. 2021. URL: http://openaccess.thecvf.com/content/WACV2021/html/Shen_Efficient_Attention_Attention_With_Linear_Complexities_WACV_2021_paper.html.

[51] K. Song et al. *MASS: Masked Sequence to Sequence Pre-training for Language Generation*. proceedings.mlr.press, May 24. 2019. URL: http://proceedings.mlr.press/v97/song19d.html.

[52] F. Stahlberg and B. Byrne. *The CUED's Grammatical Error Correction Systems for BEA-2019*. arXiv.org, Jun. 29. 2019. URL: https://arxiv.org/abs/1907.00168.

[53] A. Vaswani et al. *Attention Is All You Need*. arXiv.org, Jun. 12. 2017. URL: https://arxiv.org/abs/1706.03762.

[54] N. Wang et al. *Training Deep Neural Networks with 8-bit Floating Point Numbers*. Neural Information Processing Systems. 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/335d3d1cd7ef05ec77714a215Abstract.html.

[55] R. Weng et al. "Acquiring Knowledge from Pre-Trained Model to Neural Machine Translation". In: *Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, no. 05*. Apr. 2020, pp. 9266–9273. DOI: 10.1609/aaai.v34i05.6465.

[56] K. Wey et al. *Grammatical Error Correction for Code-Switched Sentences by Learners of English*. Accessed: Aug. 01. 2024. URL: https://arxiv.org/pdf/2404.12489.

[57] L. Xue et al. "mT5: A massively multilingual pre-trained text-to-text transformer". [cs]. Mar. 2021. arXiv: 2010.11934. URL: https://arxiv.org/abs/2010.11934.

[58] R. Yasrab, N. Gu, and X. Zhang. "An Encoder-Decoder Based Convolution Neural Network (CNN) for Future Advanced Driver Assistance System (ADAS)". In: *Applied Sciences* 7.4 (Mar. 2017), p. 312. DOI: 10.3390/app7040312.

[59] Z. Yuan and T. Briscoe. *Grammatical error correction using neural machine translation*. Association for Computational Linguistics, 2016. Accessed: Jul. 02. 2024. URL: https://aclanthology.org/N16-1042.pdf.

[60] R. Zhan et al. *Difficulty-Aware Machine Translation Evaluation*. arXiv.org, Jul. 29. 2021. URL: https://arxiv.org/abs/2107.14402.

[61] S. Zhang et al. *Few-Shot Domain Adaptation for Grammatical Error Correction via Meta-Learning*. arXiv.org, Jan. 29. 2021. URL: https://arxiv.org/abs/2101.12409.

[62] W. Zhao et al. *Improving Grammatical Error Correction via Pre-Training a Copy-Augmented Architecture with Unlabeled Data*. arXiv.org, Jun. 11. 2019. URL: https://arxiv.org/abs/1903.00138.

[63] W. Zhao et al. *Improving Grammatical Error Correction via Pre-Training a Copy-Augmented Architecture with Unlabeled Data*. Accessed: Aug. 01. 2024. URL: https://arxiv.org/pdf/1903.00138.

[64] W. Zhou et al. *Improving Grammatical Error Correction with Machine Translation Pairs*. Accessed: Jul. 02. 2024. URL: https://arxiv.org/pdf/1911.02825.

[65] J. Zhu et al. *Incorporating BERT into Neural Machine Translation*. openreview.net, Sep. 25. 2019. URL: https://openreview.net/forum?id=Hyl7ygStwB.

[66] Y. Zhu et al. "Encoder-Decoder Architecture for Supervised Dynamic Graph Learning: A Survey". [cs]. Mar. 2022. arXiv: 2203.10480. URL: https://arxiv.org/abs/2203.10480.

A Appendix

A.1 Appendix A

The score file for best performing test-case of cLang-8 dataset.

```
Standard Output:

=========== Span-Based Correction ============
TP      FP      FN      Prec    Rec     F0.5
203     429     761     0.3212  0.2106  0.2907
==============================================


Standard Error:
```

A.2 Appendix B

The score file for best performing test-case of CoNLL-2014 dataset.

```
Standard Output:

=========== Span-Based Correction ============
TP      FP      FN      Prec    Rec     F0.5
130     150     992     0.4643  0.1159  0.2899
==============================================


Standard Error:
```

A.3 Appendix C

The training result of model when training on cLang-8 dataset.

```
{'loss': 4.1611, 'grad_norm': 0.6678088307380676, 'learning_rate': 4.523809523809524e-05, 'epoch': 0.29}
{'eval_loss': 0.07037542015314102, 'eval_runtime': 350.896, 'eval_samples_per_second': 1.71, 'eval_steps_per_second': 0.427, 'epoch': 0.29}
{'loss': 0.0692, 'grad_norm': 0.27168968319892883, 'learning_rate': 4.047619047619048e-05, 'epoch': 0.57}
{'eval_loss': 0.044095221906900406, 'eval_runtime': 349.9618, 'eval_samples_per_second': 1.714, 'eval_steps_per_second': 0.429, 'epoch': 0.57}
{'loss': 0.0533, 'grad_norm': 0.10687293112277985, 'learning_rate': 3.571428571428572e-05, 'epoch': 0.86}
{'eval_loss': 0.02696247771382332, 'eval_runtime': 358.0029, 'eval_samples_per_second': 1.676, 'eval_steps_per_second': 0.419, 'epoch': 0.86}
{'loss': 0.041, 'grad_norm': 0.1948838084936142, 'learning_rate': 3.095238095238095e-05, 'epoch': 1.14}
{'eval_loss': 0.019159551709890366, 'eval_runtime': 360.3258, 'eval_samples_per_second': 1.665, 'eval_steps_per_second': 0.416, 'epoch': 1.14}
{'loss': 0.0338, 'grad_norm': 0.21707555651664734, 'learning_rate': 2.6190476190476192e-05, 'epoch': 1.43}
{'eval_loss': 0.016715496778488816, 'eval_runtime': 359.2971, 'eval_samples_per_second': 1.67, 'eval_steps_per_second': 0.417, 'epoch': 1.43}
{'loss': 0.0215, 'grad_norm': 0.13568343222141266, 'learning_rate': 2.1428571428571428e-05, 'epoch': 1.71}
{'eval_loss': 0.015457618981599808, 'eval_runtime': 333.5707, 'eval_samples_per_second': 1.799, 'eval_steps_per_second': 0.45, 'epoch': 1.71}
{'loss': 0.0194, 'grad_norm': 0.07677408307790756, 'learning_rate': 1.6666666666666667e-05, 'epoch': 2.0}
{'eval_loss': 0.014559399336576462, 'eval_runtime': 326.6373, 'eval_samples_per_second': 1.837, 'eval_steps_per_second': 0.459, 'epoch': 2.0}
{'loss': 0.0174, 'grad_norm': 0.10475517064332962, 'learning_rate': 1.1904761904761905e-05, 'epoch': 2.29}
{'eval_loss': 0.014138543047010899, 'eval_runtime': 325.3136, 'eval_samples_per_second': 1.844, 'eval_steps_per_second': 0.461, 'epoch': 2.29}
{'loss': 0.0166, 'grad_norm': 0.27483874559402466, 'learning_rate': 7.142857142857143e-06, 'epoch': 2.57}
{'eval_loss': 0.013798808678984642, 'eval_runtime': 319.1617, 'eval_samples_per_second': 1.88, 'eval_steps_per_second': 0.47, 'epoch': 2.57}
{'loss': 0.0153, 'grad_norm': 0.08374791592359543, 'learning_rate': 2.3809523809523808e-06, 'epoch': 2.86}
{'eval_loss': 0.013669227249920368, 'eval_runtime': 347.159, 'eval_samples_per_second': 1.728, 'eval_steps_per_second': 0.432, 'epoch': 2.86}
{'train_runtime': 12978.6969, 'train_samples_per_second': 0.323, 'train_steps_per_second': 0.081, 'train_loss': 0.4244306659130823, 'epoch': 3.0}
Directory Err_files_evals created.
```

A.4 Appendix D

The training result of model when training on CoNLL-2014 dataset.

```
{'loss': 2.1838, 'grad_norm': 0.2233293056488037, 'learning_rate': 3.826291079812207e-05, 'epoch': 0.7}
{'eval_loss': 0.09442944824695587, 'eval_runtime': 198.5006, 'eval_samples_per_second': 1.909, 'eval_steps_per_second': 0.479, 'epoch': 0.7}
{'loss': 0.0696, 'grad_norm': 0.27740368247032166, 'learning_rate': 2.6525821596244134e-05, 'epoch': 1.41}
{'eval_loss': 0.06482503563165665, 'eval_runtime': 199.5282, 'eval_samples_per_second': 1.899, 'eval_steps_per_second': 0.476, 'epoch': 1.41}
{'loss': 0.0536, 'grad_norm': 0.20726701617240906, 'learning_rate': 1.4788732394366198e-05, 'epoch': 2.11}
{'eval_loss': 0.05593892186880112, 'eval_runtime': 198.7497, 'eval_samples_per_second': 1.907, 'eval_steps_per_second': 0.478, 'epoch': 2.11}
{'loss': 0.045, 'grad_norm': 0.18434035778045654, 'learning_rate': 3.051643192488263e-06, 'epoch': 2.82}
{'eval_loss': 0.05404600128531456, 'eval_runtime': 203.0354, 'eval_samples_per_second': 1.867, 'eval_steps_per_second': 0.468, 'epoch': 2.82}
{'train_runtime': 4307.5251, 'train_samples_per_second': 0.396, 'train_steps_per_second': 0.099, 'train_loss': 0.5548876081833817, 'epoch': 3.0}
Directory Err_files_evals created.
```

A.5 Appendix E

The token-based correction file for best performing test-case of cLang-8 dataset.

```
==================== Token-Based Detection ====================
Category        TP      FP      FN      P       R       F0.5
M:ADJ           0       0       4       1.0     0.0     0.0
M:ADV           2       0       1       1.0     0.6667  0.9091
M:CONJ          0       1       2       0.0     0.0     0.0
M:CONTR         0       0       1       1.0     0.0     0.0
M:DET           11      38      24      0.2245  0.3143  0.2381
M:NOUN          9       3       13      0.75    0.4091  0.6429
M:OTHER         14      16      12      0.4667  0.5385  0.4795
M:PART          0       0       1       1.0     0.0     0.0
M:PREP          3       10      9       0.2308  0.25    0.2344
M:PRON          2       7       5       0.2222  0.2857  0.2326
M:PUNCT         7       8       12      0.4667  0.3684  0.443
M:VERB          4       3       7       0.5714  0.3636  0.5128
M:VERB:FORM     0       2       1       0.0     0.0     0.0
M:VERB:TENSE    1       1       3       0.5     0.25    0.4167
R:ADJ           4       0       3       1.0     0.5714  0.8696
R:ADV           0       1       5       0.0     0.0     0.0
R:CONTR         6       0       1       1.0     0.8571  0.9677
R:DET           4       0       11      1.0     0.2667  0.6452
R:MORPH         7       3       10      0.7     0.4118  0.614
R:NOUN          12      11      28      0.5217  0.3     0.4545
R:NOUN:NUM      6       19      21      0.24    0.2222  0.2362
R:NOUN:POSS     0       0       3       1.0     0.0     0.0
R:ORTH          24      16      61      0.6     0.2824  0.4898
R:OTHER         109     24      128     0.8195  0.4599  0.7087
R:PART          0       0       1       1.0     0.0     0.0
R:PREP          3       2       23      0.6     0.1154  0.3261
R:PRON          5       4       11      0.5556  0.3125  0.4808
R:PUNCT         10      21      22      0.3226  0.3125  0.3205
R:SPELL         7       5       7       0.5833  0.5     0.5645
R:VERB          11      4       24      0.7333  0.3143  0.5789
R:VERB:FORM     4       5       6       0.4444  0.4     0.4348
R:VERB:INFL     0       2       0       0.0     1.0     0.0
R:VERB:SVA      2       6       7       0.25    0.2222  0.2439
R:VERB:TENSE    7       10      33      0.4118  0.175   0.3241
R:WO            9       30      69      0.2308  0.1154  0.1923
U:ADJ           3       7       8       0.3     0.2727  0.2941
U:ADV           4       8       8       0.3333  0.3333  0.3333
U:CONJ          3       2       5       0.6     0.375   0.5357
U:CONTR         1       1       1       0.5     0.5     0.5
U:DET           4       6       36      0.4     0.1     0.25
U:NOUN          6       13      22      0.3158  0.2143  0.2885
U:NOUN:POSS     0       1       2       0.0     0.0     0.0
U:OTHER         276     97      76      0.7399  0.7841  0.7484
U:PART          1       0       2       1.0     0.3333  0.7143
U:PREP          7       6       12      0.5385  0.3684  0.493
U:PRON          1       6       8       0.1429  0.1111  0.1351
U:PUNCT         21      10      30      0.6774  0.4118  0.6
U:VERB          8       12      17      0.4     0.32    0.381
U:VERB:FORM     0       2       3       0.0     0.0     0.0
U:VERB:TENSE    2       4       5       0.3333  0.2857  0.3226

=========== Token-Based Detection ============
TP      FP      FN      Prec    Rec     F0.5
620     427     804     0.5922  0.4354  0.5524
==============================================
```

A.6 Appendix F

The token-based correction file for best performing test-case of CoNLL-2014 dataset.

```
==================== Token-Based Detection ====================
Category        TP      FP      FN      P       R       F0.5
M:ADJ           0       1       5       0.0     0.0     0.0
M:ADV           1       2       8       0.3333  0.1111  0.2381
M:CONJ          0       0       5       1.0     0.0     0.0
M:DET           4       1       52      0.8     0.0714  0.2632
M:NOUN          5       1       20      0.8333  0.2     0.5102
M:NOUN:POSS     1       0       1       1.0     0.5     0.8333
M:OTHER         5       0       37      1.0     0.119   0.4032
M:PART          0       0       2       1.0     0.0     0.0
M:PREP          1       1       32      0.5     0.0303  0.122
M:PRON          0       0       4       1.0     0.0     0.0
M:PUNCT         7       0       7       1.0     0.5     0.8333
M:VERB          2       1       13      0.6667  0.1333  0.3704
M:VERB:FORM     0       0       5       1.0     0.0     0.0
M:VERB:TENSE    0       0       9       1.0     0.0     0.0
R:ADJ           1       0       11      1.0     0.0833  0.3125
R:ADJ:FORM      0       1       1       0.0     0.0     0.0
R:ADV           6       1       4       0.8571  0.6     0.7895
R:CONJ          0       0       1       1.0     0.0     0.0
R:CONTR         0       1       4       0.0     0.0     0.0
R:DET           2       4       22      0.3333  0.0833  0.2083
R:MORPH         4       3       30      0.5714  0.1176  0.3226
R:NOUN          13      3       40      0.8125  0.2453  0.5556
R:NOUN:INFL     0       1       1       0.0     0.0     0.0
R:NOUN:NUM      2       5       44      0.2857  0.0435  0.1351
R:ORTH          10      4       12      0.7143  0.4545  0.641
R:OTHER         272     9       259     0.968   0.5122  0.8218
R:PART          0       1       5       0.0     0.0     0.0
R:PREP          2       3       36      0.4     0.0526  0.1724
R:PRON          3       1       10      0.75    0.2308  0.5172
R:PUNCT         0       0       18      1.0     0.0     0.0
R:SPELL         1       5       12      0.1667  0.0769  0.1351
R:VERB          12      1       63      0.9231  0.16    0.4724
R:VERB:FORM     2       4       13      0.3333  0.1333  0.2564
R:VERB:SVA      2       5       26      0.2857  0.0714  0.1786
R:VERB:TENSE    7       3       43      0.7     0.14    0.3889
R:WO            3       0       7       1.0     0.3     0.6818
U:ADJ           2       4       4       0.3333  0.3333  0.3333
U:ADV           4       2       5       0.6667  0.4444  0.6061
U:CONJ          1       1       2       0.5     0.3333  0.4545
U:DET           13      10      38      0.5652  0.2549  0.4545
U:NOUN          5       2       15      0.7143  0.25    0.5208
U:NOUN:POSS     1       0       1       1.0     0.5     0.8333
U:OTHER         248     9       78      0.965   0.7607  0.9158
U:PART          0       0       1       1.0     0.0     0.0
U:PREP          0       2       22      0.0     0.0     0.0
U:PRON          4       0       3       1.0     0.5714  0.8696
U:PUNCT         1       2       17      0.3333  0.0556  0.1667
U:VERB          2       1       21      0.6667  0.087   0.2857
U:VERB:FORM     0       1       3       0.0     0.0     0.0
U:VERB:TENSE    1       2       6       0.3333  0.1429  0.2632

=========== Token-Based Detection ============
TP      FP      FN      Prec    Rec     F0.5
650     98      1078    0.869   0.3762  0.6886
==============================================
```

A.7 Appendix G

The Test-case 2 (6 epochs) for both the datasets.

For cLang-8

```
Standard Output:

=========== Span-Based Correction ============
TP      FP      FN      Prec    Rec     F0.5
179     495     610     0.2656  0.2269  0.2568
==============================================


Standard Error:
```

For CoNLL-2014

```
Standard Output:

=========== Span-Based Correction ============
TP      FP      FN      Prec    Rec     F0.5
91      212     634     0.3003  0.1255  0.2349
==============================================


Standard Error:
```

A.8 Appendix H

The Test-case 3 (10 epochs) for both the datasets.

For cLang-8

```
Standard Output:

=========== Span-Based Correction ============
TP      FP      FN      Prec    Rec     F0.5
154     742     584     0.1719  0.2087  0.1782
==============================================


Standard Error:
```

For CoNLL-2014

```
Standard Output:

=========== Span-Based Correction ============
TP      FP      FN      Prec    Rec     F0.5
91      212     634     0.3003  0.1255  0.2349
==============================================


Standard Error:
```

A.9 Appendix I

Token-based correction F0.5 scores for Test-case 2 (6 epochs)

For CoNLL-2014

```
==================== Token-Based Detection ====================
Category       TP       FP       FN       P        R        F0.5
M:ADJ          0        0        4        1.0      0.0      0.0
M:ADV          0        0        8        1.0      0.0      0.0
M:CONJ         0        0        1        1.0      0.0      0.0
M:DET          2        5        50       0.2857   0.0385   0.125
M:NOUN         2        0        8        1.0      0.2      0.5556
M:NOUN:POSS    0        0        1        1.0      0.0      0.0
M:OTHER        4        2        19       0.6667   0.1739   0.4255
M:PART         0        0        1        1.0      0.0      0.0
M:PREP         1        1        19       0.5      0.05     0.1786
M:PRON         0        0        3        1.0      0.0      0.0
M:PUNCT        2        0        4        1.0      0.3333   0.7143
M:VERB         1        1        16       0.5      0.0588   0.2
M:VERB:FORM    0        0        1        1.0      0.0      0.0
M:VERB:TENSE   1        1        6        0.5      0.1429   0.3333
R:ADJ          2        2        4        0.5      0.3333   0.4545
R:ADJ:FORM     0        1        0        0.0      1.0      0.0
R:ADV          4        0        5        1.0      0.4444   0.8
R:CONJ         0        0        1        1.0      0.0      0.0
R:CONTR        0        0        1        1.0      0.0      0.0
R:DET          1        1        17       0.5      0.0556   0.1923
R:MORPH        5        2        23       0.7143   0.1786   0.4464
R:NOUN         3        1        31       0.75     0.0882   0.3
R:NOUN:INFL    0        0        1        1.0      0.0      0.0
R:NOUN:NUM     5        8        35       0.3846   0.125    0.2717
R:NOUN:POSS    0        0        1        1.0      0.0      0.0
R:ORTH         0        3        8        0.0      0.0      0.0
R:OTHER        42       15       203      0.7368   0.1714   0.444
R:PART         0        0        4        1.0      0.0      0.0
R:PREP         5        1        29       0.8333   0.1471   0.431
R:PRON         1        1        14       0.5      0.0667   0.2174
R:PUNCT        0        2        18       0.0      0.0      0.0
R:SPELL        0        1        5        0.0      0.0      0.0
R:VERB         6        2        57       0.75     0.0952   0.3158
R:VERB:FORM    1        6        17       0.1429   0.0556   0.1087
R:VERB:SVA     1        10       18       0.0909   0.0526   0.0794
R:VERB:TENSE   4        0        27       1.0      0.129    0.4255
R:WO           10       0        5        1.0      0.6667   0.9091
U:ADJ          1        2        2        0.3333   0.3333   0.3333
U:ADV          9        1        5        0.9      0.6429   0.8333
U:DET          7        12       22       0.3684   0.2414   0.3333
U:NOUN         5        1        9        0.8333   0.3571   0.6579
U:NOUN:POSS    0        0        2        1.0      0.0      0.0
U:OTHER        10       5        42       0.6667   0.1923   0.4464
U:PART         0        0        2        1.0      0.0      0.0
U:PREP         1        1        16       0.5      0.0588   0.2
U:PRON         1        1        3        0.5      0.25     0.4167
U:PUNCT        0        3        13       0.0      0.0      0.0
U:VERB         0        1        5        0.0      0.0      0.0
U:VERB:TENSE   1        3        7        0.25     0.125    0.2083

=========== Token-Based Detection ============
TP      FP      FN      Prec    Rec     F0.5
138     96      793     0.5897  0.1482  0.3696
==============================================
```

For cLang-8

```
==================== Token-Based Detection ====================
Category        TP      FP      FN      P       R       F0.5
M:ADJ           0       1       2       0.0     0.0     0.0
M:ADV           0       1       3       0.0     0.0     0.0
M:CONJ          1       1       2       0.5     0.3333  0.4545
M:CONTR         0       0       1       1.0     0.0     0.0
M:DET           16      37      18      0.3019  0.4706  0.3252
M:NOUN          1       1       4       0.5     0.2     0.3846
M:NOUN:POSS     0       0       2       1.0     0.0     0.0
M:OTHER         12      10      12      0.5455  0.5     0.5357
M:PART          0       1       0       0.0     1.0     0.0
M:PREP          1       14      5       0.0667  0.1667  0.0758
M:PRON          4       6       1       0.4     0.8     0.4444
M:PUNCT         13      32      15      0.2889  0.4643  0.3125
M:VERB          4       6       7       0.4     0.3636  0.3922
M:VERB:FORM     0       4       2       0.0     0.0     0.0
M:VERB:TENSE    2       2       4       0.5     0.3333  0.4545
R:ADJ           7       3       9       0.7     0.4375  0.625
R:ADV           1       0       2       1.0     0.3333  0.7143
R:CONTR         1       0       0       1.0     1.0     1.0
R:DET           10      3       15      0.7692  0.4     0.6494
R:MORPH         2       3       10      0.4     0.1667  0.3125
R:NOUN          18      7       20      0.72    0.4737  0.6522
R:NOUN:NUM      5       16      24      0.2381  0.1724  0.2212
R:NOUN:POSS     0       0       2       1.0     0.0     0.0
R:ORTH          38      34      52      0.5278  0.4222  0.5026
R:OTHER         52      34      76      0.6047  0.4062  0.5508
R:PART          0       1       1       0.0     0.0     0.0
R:PREP          7       9       19      0.4375  0.2692  0.3889
R:PRON          2       2       9       0.5     0.1818  0.3704
R:PUNCT         12      12      22      0.5     0.3529  0.4615
R:SPELL         2       9       3       0.1818  0.4     0.2041
R:VERB          6       4       12      0.6     0.3333  0.5172
R:VERB:FORM     6       8       13      0.4286  0.3158  0.4
R:VERB:SVA      0       6       2       0.0     0.0     0.0
R:VERB:TENSE    13      13      30      0.5     0.3023  0.4422
R:WO            9       33      53      0.2143  0.1452  0.1957
U:ADJ           5       3       10      0.625   0.3333  0.5319
U:ADV           7       5       10      0.5833  0.4118  0.5385
U:CONJ          2       3       5       0.4     0.2857  0.3704
U:CONTR         1       0       1       1.0     0.5     0.8333
U:DET           4       13      26      0.2353  0.1333  0.2041
U:NOUN          14      12      21      0.5385  0.4     0.5036
U:NOUN:POSS     0       2       0       0.0     1.0     0.0
U:OTHER         78      14      35      0.8478  0.6903  0.8108
U:PART          0       2       1       0.0     0.0     0.0
U:PREP          2       12      6       0.1429  0.25    0.1562
U:PRON          3       6       9       0.3333  0.25    0.3125
U:PUNCT         16      17      27      0.4848  0.3721  0.4571
U:VERB          5       14      9       0.2632  0.3571  0.2778
U:VERB:FORM     2       1       0       0.6667  1.0     0.7143
U:VERB:TENSE    2       3       6       0.4     0.25    0.3571

=========== Token-Based Detection ============
TP      FP      FN      Prec    Rec     F0.5
386     420     618     0.4789  0.3845  0.4565
==============================================
```

A.10 Appendix J

Token-based correction F0.5 scores for Test-case 3 (10 epochs)

For CoNLL-2014

```
==================== Token-Based Detection ====================
Category       TP       FP       FN       P        R        F0.5
M:ADJ          0        0        7        1.0      0.0      0.0
M:ADV          1        0        8        1.0      0.1111   0.3846
M:CONJ         1        1        4        0.5      0.2      0.3846
M:DET          3        7        31       0.3      0.0882   0.2027
M:NOUN         4        2        12       0.6667   0.25     0.5
M:NOUN:POSS    0        0        1        1.0      0.0      0.0
M:OTHER        2        1        25       0.6667   0.0741   0.2564
M:PART         0        0        2        1.0      0.0      0.0
M:PREP         2        2        17       0.5      0.1053   0.2857
M:PRON         0        1        2        0.0      0.0      0.0
M:PUNCT        2        1        6        0.6667   0.25     0.5
M:VERB         1        3        12       0.25     0.0769   0.1724
M:VERB:FORM    0        1        2        0.0      0.0      0.0
M:VERB:TENSE   0        0        7        1.0      0.0      0.0
R:ADJ          1        1        11       0.5      0.0833   0.25
R:ADV          2        1        5        0.6667   0.2857   0.5263
R:CONJ         0        0        1        1.0      0.0      0.0
R:CONTR        0        1        0        0.0      1.0      0.0
R:DET          1        8        21       0.1111   0.0455   0.0862
R:MORPH        5        8        15       0.3846   0.25     0.3472
R:NOUN         6        3        21       0.6667   0.2222   0.4762
R:NOUN:NUM     10       22       29       0.3125   0.2564   0.2994
R:ORTH         5        3        7        0.625    0.4167   0.5682
R:OTHER        40       3        176      0.9302   0.1852   0.5155
R:PART         0        0        1        1.0      0.0      0.0
R:PREP         8        7        23       0.5333   0.2581   0.4396
R:PRON         1        2        9        0.3333   0.1      0.2273
R:PUNCT        3        5        11       0.375    0.2143   0.3261
R:SPELL        3        7        14       0.3      0.1765   0.2632
R:VERB         8        2        27       0.8      0.2286   0.5333
R:VERB:FORM    0        5        12       0.0      0.0      0.0
R:VERB:INFL    0        1        0        0.0      1.0      0.0
R:VERB:SVA     5        17       18       0.2273   0.2174   0.2252
R:VERB:TENSE   7        9        27       0.4375   0.2059   0.3571
R:WO           10       1        10       0.9091   0.5      0.7812
U:ADJ          3        2        3        0.6      0.5      0.5769
U:ADV          0        3        3        0.0      0.0      0.0
U:CONJ         0        1        0        0.0      1.0      0.0
U:DET          7        16       21       0.3043   0.25     0.2917
U:NOUN         4        2        12       0.6667   0.25     0.5
U:NOUN:POSS    0        0        2        1.0      0.0      0.0
U:OTHER        202      13       35       0.9395   0.8523   0.9207
U:PART         0        1        2        0.0      0.0      0.0
U:PREP         4        4        9        0.5      0.3077   0.4444
U:PRON         1        0        2        1.0      0.3333   0.7143
U:PUNCT        0        0        17       1.0      0.0      0.0
U:VERB         2        1        10       0.6667   0.1667   0.4167
U:VERB:FORM    0        1        1        0.0      0.0      0.0
U:VERB:TENSE   3        3        5        0.5      0.375    0.4688

=========== Token-Based Detection ============
TP      FP      FN      Prec    Rec     F0.5
357     172     696     0.6749  0.339   0.5633
==============================================
```

For cLang-8

```
==================== Token-Based Detection ====================
Category       TP       FP       FN       P        R        F0.5
M:ADJ          1        1        1        0.5      0.5      0.5
M:ADV          2        2        4        0.5      0.3333   0.4545
M:CONJ         0        1        3        0.0      0.0      0.0
M:CONTR        0        0        1        1.0      0.0      0.0
M:DET          9        53       7        0.1452   0.5625   0.1705
M:NOUN         1        7        5        0.125    0.1667   0.1316
M:NOUN:POSS    1        0        0        1.0      1.0      1.0
M:OTHER        12       19       13       0.3871   0.48     0.4027
M:PART         1        0        1        1.0      0.5      0.8333
M:PREP         2        10       7        0.1667   0.2222   0.1754
M:PRON         1        10       2        0.0909   0.3333   0.1064
M:PUNCT        11       35       18       0.2391   0.3793   0.2582
M:VERB         5        12       7        0.2941   0.4167   0.3125
M:VERB:FORM    0        2        0        0.0      1.0      0.0
M:VERB:TENSE   1        5        3        0.1667   0.25     0.1786
R:ADJ          3        1        5        0.75     0.375    0.625
R:ADV          3        0        2        1.0      0.6      0.8824
R:CONTR        2        0        4        1.0      0.3333   0.7143
R:DET          12       7        16       0.6316   0.4286   0.5769
R:MORPH        5        10       7        0.3333   0.4167   0.3472
R:NOUN         14       12       22       0.5385   0.3889   0.5
R:NOUN:NUM     5        21       9        0.1923   0.3571   0.2119
R:ORTH         44       63       35       0.4112   0.557    0.4339
R:OTHER        84       77       73       0.5217   0.535    0.5243
R:PART         0        2        1        0.0      0.0      0.0
R:PREP         8        10       16       0.4444   0.3333   0.4167
R:PRON         1        5        7        0.1667   0.125    0.1562
R:PUNCT        14       38       22       0.2692   0.3889   0.2869
R:SPELL        1        16       2        0.0588   0.3333   0.0704
R:VERB         14       10       21       0.5833   0.4      0.5344
R:VERB:FORM    4        7        8        0.3636   0.3333   0.3571
R:VERB:INFL    1        3        0        0.25     1.0      0.2941
R:VERB:SVA     3        9        3        0.25     0.5      0.2778
R:VERB:TENSE   13       34       21       0.2766   0.3824   0.2928
R:WO           5        61       42       0.0758   0.1064   0.0804
U:ADJ          4        8        4        0.3333   0.5      0.3571
U:ADV          6        12       7        0.3333   0.4615   0.3529
U:CONJ         2        1        2        0.6667   0.5      0.625
U:DET          0        23       20       0.0      0.0      0.0
U:NOUN         8        16       20       0.3333   0.2857   0.3226
U:NOUN:POSS    0        3        3        0.0      0.0      0.0
U:OTHER        87       97       55       0.4728   0.6127   0.4954
U:PART         0        1        2        0.0      0.0      0.0
U:PREP         5        14       7        0.2632   0.4167   0.2841
U:PRON         2        4        8        0.3333   0.2      0.2941
U:PUNCT        14       18       29       0.4375   0.3256   0.4094
U:VERB         13       11       6        0.5417   0.6842   0.5652
U:VERB:FORM    0        2        0        0.0      1.0      0.0
U:VERB:TENSE   1        1        5        0.5      0.1667   0.3571

========== Token-Based Detection ============
TP      FP      FN      Prec    Rec     F0.5
425     754     556     0.3605  0.4332  0.373
==============================================
```

A.11 Appendix K

Source code and main folder: Contains the main folder which has all the files required to run the model.

To run the source code, following are the components of the **'gt5_files'** folder that contains all the code, datasets and bestcase results.

- py_files: Contains all the python scripts to run the model.
    1. main.py: Runs the GEC model from the file 'Gecmodel.py'.
    2. Gecmodel.py: Contains the complete source code of GEC model.
    3. Gecmodel_errEval.py: Contains the script for obtaining F0.5 scores on the results obtained by 'Gecmodel.py'.
    4. ReadMe.md: Instructions to guide running and evaluating the python model.
- datasets: Contains the datasets of the project, CoNLL-2014 and cLang-8 respectively.
- bestCase_results: Contains the complete folders of bestcase results of both the datasets.
    1. CoNLL-2014
        (a) result.out: Contains the recorded epochs and evaluation loss and training loss.
        (b) Err_files_evals
            – CoNLL.incorrect: Incorrect sentences input to the model.
            – Err_score.txt: Final F0.5 scores, precision and recall.
            – predictions.csv: Predictions made by the model in csv format.
            – preditions.txt: Predicitions converted to text-format required for 'm2' format conversion.
            – ref_m2.txt: m2 format of references files.
            – references.txt: Correct form of sentences that were input to the model.
            – results.txt: m2 format of recorded results by the model for comparing with ref_m2 file to obtain scores.
    2. cLang_8
        (a) result.out: Contains the recorded epochs and evaluation loss and training loss.
        (b) Err_files_evals
            – clang8.incorrect: Incorrect sentences input to the model.
            – Err_score.txt: Final F0.5 scores, precision and recall.
            – predictions.csv: Predictions made by the model in csv format.
            – preditions.txt: Predicitions converted to text-format required for 'm2' format conversion.
            – ref_m2.txt: m2 format of references files.
            – references.txt: Correct form of sentences that were input to the model.
            – results.txt: m2 format of recorded results by the model for comparing with ref_m2 file to obtain scores.

Main file to run the GEC model ('main.py')

```
# Shivang Chaudhary
# Course: MSc Artificial Intelligence
# Year: 2023-24
# This is the main file to run the main GEC model.
# Once results are recorded after running the file, 'Gecmodel_errEval.py' must be run to evaluate the recorded predictions and obtain final F0.5 scores.

import subprocess
import Gecmodel


def run_file(script):
    # Running the file using 'subprocess'.
    try:
        result = subprocess.run(['python', script], check=True, capture_output=True, text=True)
        print(f"Output of {script}:\n{result.stdout}")
    except subprocess.CalledProcessError as e:
        print(f"Error while running {script}:\n{e.stderr}")


# Press the green button in the gutter to run the script.
if __name__ == '__main__':
    # Imported the GEC - script and putting in as parameter to run it.
    script = Gecmodel
    run_file(script)
```

A.12 Appendix L

Main file or the GEC model ('Gecmodel.py')

```
# -----------
# Author: Shivang Chaudhary
# Course: MSc Artificial Intelligence
# Year: 2023-24
# Following is the GEC 'gt5' that is experimented on limited versions of cLang-8 and CoNLL-2014 dataset
# -----------

import os
import random

import pandas as pd
from datasets import load_dataset, Dataset
from transformers import AutoTokenizer, T5ForConditionalGeneration, Trainer, TrainingArguments, \
    TextClassificationPipeline, AutoModelForSequenceClassification, DataCollatorWithPadding, AutoModelForSeq2SeqLM, \
    pipeline, AdamW, T5Config
import evaluate

# LOADING THE DATASET FOR THE MODEL
# Load CLang-8 dataset : The dataset labelling -> Column1= all incorrect sentences, Column2= all correct sentences
df = pd.read_csv('Data/clang8.csv')

# Create Hugging Face Dataset
# dataset = Dataset.from_pandas(df)

# LOAD TOKENIZER
tokenizer = AutoTokenizer.from_pretrained('t5-base')

# ----Introducing dropout:-
config = T5Config.from_pretrained('t5-base', dropout_rate=0.1)
# ----

# Load pretrained model :-
model = T5ForConditionalGeneration.from_pretrained('t5-base',
                                                   config=config
                                                   # num_labels=len(unique_labels),
                                                   # label2id=label2id,
                                                   # id2label=id2label,
                                                   )


# ------- Applying data-augmentation
# Function introduces random errors.
def introduce_errors(sentence):
    words = sentence.split()
    if len(words) < 2:
        # Skips very short sentences
        return sentence

    error_type = random.choice(['swap', 'insert', 'delete', 'replace'])

    if error_type == 'swap' and len(words) > 1:
        # Introduces a swap error between random adjacent words
        index = random.randint(0, len(words) - 2)
        words[index], words[index + 1] = words[index + 1], words[index]
    elif error_type == 'insert':
        # Inserting a random word at a random position
        random_word = random.choice(words)
```

```
        index = random.randint(0, len(words))
        words.insert(index, random_word)
    elif error_type == 'delete' and len(words) > 1:
        # Deleting a random word
        index = random.randint(0, len(words) - 1)
        words.pop(index)
    elif error_type == 'replace':
        # Replacing a random word with another random word
        index = random.randint(0, len(words) - 1)
        random_word = random.choice(words)
        words[index] = random_word

    return ' '.join(words)


# Creating a new column with augmented data
df['augmented'] = df['Column1'].apply(introduce_errors)

# Combine the original and augmented data
augmented_df = pd.DataFrame({
    'Column1': pd.concat([df['Column1'], df['augmented']]),
    'Column2': pd.concat([df['Column2'], df['Column2']])
})

# Updated dataset for further experimentation:-
dataset = Dataset.from_pandas(augmented_df)

# -----------------------------------


# Splitting the dataset
train_test_split = dataset.train_test_split(test_size=0.3)
train_dataset = train_test_split['train']
eval_dataset = train_test_split['test']


# ------


# PRE-PROCESSING FUNCTION:-
def preprocess_function(examples):
    inputs = examples['Column1']
    targets = examples['Column2']

    model_inputs = tokenizer(inputs, padding='max_length', truncation='longest_first',
    return_tensors='pt'
                             , max_length=512)

    # with tokenizer.as_target_tokenizer():
    labels = tokenizer(targets, padding='max_length', truncation='longest_first',
    return_tensors='pt'
                       , max_length=512)

    model_inputs['labels'] = labels['input_ids']
    # Used for testing purposes ---
    # print(model_inputs)
    # print(labels)
    # ---
    return model_inputs


# -------Retaining the dataset for backup (if required in later processing phase.)
original_eval_dataset = eval_dataset
# --------
```

```
120
121 # -- MAKING THE TRAIN AND VALIDATION SET
122 train_dataset = train_dataset.map(preprocess_function, batched=True)
123 eval_dataset = eval_dataset.map(preprocess_function, batched=True)
124 # print(eval_dataset) -> For testing phase.
125 # ----
126
127
128 # Setting up evaluation - THIS IS BACKUP-METRIC, NOT THE FOCUS OF THE PROJECT.
129 metric = evaluate.load('accuracy')
130
131 # Defining training arguments
132 training_args = TrainingArguments(
133     output_dir='./results_clang_10epc', # Store the results for epochs 3,6 and 10
    respectively.
134     evaluation_strategy='steps',
135     learning_rate=5e-5,
136     per_device_train_batch_size=4,
137     per_device_eval_batch_size=4,
138     num_train_epochs=10,  # Can experiment with epochs, diverse results expected with 3,6 and
     10.
139     weight_decay=0.01,
140     save_steps=1000,  # Saving checkpoint every 1000 steps
141     save_total_limit=3,  # Limit the total amount of checkpoints. Deletes the older
    checkpoints.
142     logging_dir='./logs_clang8_tuned',  # Directory for storing logs, can be renamed for
    different epochs to store data.
143     logging_steps=1000,  # Log every 1000 steps
144     load_best_model_at_end=True,  # Load the best model when finished training
145 )
146 # --> metric_for_best_model="accuracy" -> NOT THE PROJECT'S FOCUS.
147
148
149 # Define custom data collator
150 data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
151 # -----
152
153 # ------- Adding AdamW optimizer
154 optimizer = AdamW(model.parameters(), lr=training_args.learning_rate)
155 # ------------------------------
156
157 # Initialize Trainer
158 trainer = Trainer(
159     model=model,
160     args=training_args,
161     train_dataset=train_dataset,
162     eval_dataset=eval_dataset,
163     tokenizer=tokenizer,
164     data_collator=data_collator,
165     optimizers=(optimizer, None),
166     # compute_metrics=compute_metrics ( IF REQUIRED )
167 )
168
169 # Training the model
170 trainer.train()
171
172 # ------- BUILDING THE GEC PIPE-LINE
173 gec_pipeline = pipeline(task='text2text-generation', model=model, tokenizer=tokenizer)
174 # Save predictions and references
175 predictions = gec_pipeline([example['Column1'] for example in eval_dataset], max_length=512,
    num_return_sequences=1)
176 pred_texts = [prediction['generated_text'] for prediction in predictions]
177 references = [example['Column2'] for example in eval_dataset]
178 # ------
```

```
179
180
181 # ------------------------------------------------------------------------------
182
183 # Err_files_evals for storing the corrected sentences and scores respectively.
184 # Saving to verify:-
185 dir_path = 'Err_files_evals'
186 # Verify if the directory is present or not:-
187 if not os.path.exists(dir_path):
188     os.makedirs(dir_path)
189     print("Directory", dir_path, "created.")
190 else:
191     print("Directory", dir_path, "already exists.")
192
193 # ---
194 df = pd.DataFrame(predictions)
195 df.to_csv("Err_files_evals/predictions.csv", index=False)  # Save to CSV
196 # ---
197
198
199 # Save predictions and references-----
200 with open('Err_files_evals/predictions.txt', 'w') as pred_file, \
201         open('Err_files_evals/references.txt', 'w') as ref_file:
202     for pred, ref in zip(pred_texts, eval_dataset['Column2']):
203         pred_file.write(pred.strip() + '\n')
204         ref_file.write(ref.strip() + '\n')
205 # -----
206
207 # ---- Evaluation phase: Obtaining the needed files:-
208 # Getting the original inputs:-
209 # Prepare input files for ERRANT
210 with open('Err_files_evals/clang8.incorrect', 'w') as orig_file:
211     for incorrect in eval_dataset['Column1']:
212         orig_file.write(incorrect.strip() + '\n')
213
214 # ------
215
216 # Ensure the lengths of dataset['Column1'] and pred_texts are the same:-
217 assert len(eval_dataset['Column1']) == len(pred_texts), "Mismatch in number of original texts
      and predictions"
218 # ------
219
220 # -- FINAL TESTING THROUGH ERRANT ON NEIGHBOURING SCRIPT, 'Gecmodel_errEval.py'.
```

A.13 Appendix M

Evaluation file: To run after predictions have been captured from main model to calculate final scores ('Gecmodel_errEval.py')

```
# Author: Shivang Chaudhary
# Course: MSc Artificial Intelligence
# Year: 2023-24
# The following is the file to produce scores of the recorded sentences files.
# IT REQUIRES THE M2 FORMAT OF THE PREDICTED AND REFERENCE TEXT FILES TO PRODUCE THE FINAL SCORES.

import subprocess


def create_blank_file(filepath):
    """Creates a blank file at the given path if it doesn't exist.
  Args:
      filepath: The path to the file.
  """
    with open(filepath, "w") as file:
        pass


# Verifying the file paths ---
filepath1 = "Err_files_evals/results.txt"
create_blank_file(filepath1)

filepath2 = "Err_files_evals/ref_m2.txt"
create_blank_file(filepath2)
# ------


# Running the ERRANT commands
subprocess.run(
    ["errant_parallel", "-orig", "Err_files_evals/clang8.incorrect", "-cor", "Err_files_evals/predictions.txt", "-out",
     "Err_files_evals/results.txt"],
    capture_output=True,  # Capture stdout and stderr
    text=True  # Return the output as a string
)
# subprocess.run(["errant_m2", "-silver", "Err_files_evals/references.txt", "-out", "Err_files_evals/ref_m2.txt"],  capture_output=True, text=True)
subprocess.run(
    ["errant_parallel", "-orig", "Err_files_evals/references.txt", "-cor", "Err_files_evals/predictions.txt", "-out",
     "Err_files_evals/ref_m2.txt"],
    capture_output=True,  # Capture stdout and stderr
    text=True  # Return the output as a string
)
output = subprocess.run(["errant_compare", "-hyp", "Err_files_evals/results.txt", "-ref", "Err_files_evals/ref_m2.txt"],
                        capture_output=True,  # Capture stdout and stderr
                        text=True  # Return the output as a string
                        )

# -------- Saving the final score
output_file_path = 'Err_files_evals/Err_score.txt'
# Writing the output to file,
with open(output_file_path, "w") as file:
    file.write("Standard Output:\n")
    file.write(output.stdout)
    file.write("\n\nStandard Error:\n")
    file.write(output.stderr)
# ---------
```

B List of Figures and Tables

Figures

Tables