**TRIBHUVAN UNIVERSITY**

**INSTITUTE OF ENGINEERING**

**THAPATHALI CAMPUS**

**Minor Project Report**

**On**

# Comparative Study for Grammar Error Correction using Encoder-Decoder and T5 Transformer model

**Submitted by:**

| | |
|---|---|
| Anuj Rayamajhi | (THA076BCT007) |
| Anup Gelal | (THA076BCT008) |
| Grishma Raj Khanal | (THA076BCT016) |
| Kaustuv Karki | (THA076BCT017) |

**Submitted to:**

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

March, 2023

**TRIBHUVAN UNIVERSITY**

**INSTITUTE OF ENGINEERING**

**THAPATHALI CAMPUS**

**Minor Project Report**

**On**

# Comparative Study for Grammar Error Correction using Encoder-Decoder and T5 Transformer model

**Submitted by:**

| | |
|---|---|
| Anuj Rayamajhi | (THA076BCT007) |
| Anup Gelal | (THA076BCT008) |
| Grishma Raj Khanal | (THA076BCT016) |
| Kaustuv Karki | (THA076BCT017) |

**Submitted to:**

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

In partial fulfillment for the award of the bachelor's degree in computer engineering.

**Under the supervision of**

Er. Kiran Chandra Dahal

March, 2023

## DECLARATION

We hereby declare that the report of the project entitled as "**Grammar Error Correction"** which is being submitted to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus,** in the partial fulfilment of the requirement for the award of the Degree of Bachelor of Engineering in Computer Engineering, is a report of the work carried out by us. The materials contained in this report have not been submitted to any University or institution for the award of any degree and we are the only author of this complete work and no sources other than the listed here have been used in this work.

| | | |
|---|---|---|
| Anuj Rayamajhi | (Class Roll No: THA076BCT007) | ________________ |
| Anup Gelal | (Class Roll No: THA076BCT008) | ________________ |
| Grishma Raj Khanal | (Class Roll No: THA076BCT016) | ________________ |
| Kaustuv Karki | (Class Roll No: THA076BCT017) | ________________ |

**Date:** March, 2023

## CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the **Department of Electronics and Computer Engineering, IOE, Thapathali Campus,** a minor project work entitled "**Grammar Error Correction"** submitted by **Anuj Rayamajhi, Anup Gelal, Grishma Raj Khanal and Kaustuv Karki** in partial fulfilment for the award of Bachelor's Degree in Computer Engineering. The project was carried out under the special supervision and within the time frame prescribed by the syllabus.

We found the students to be hardworking, skilled, and ready to undertake any related work to their field of study and hence we recommend the award of partial fulfilment of bachelor's degree of Computer Engineering.

____________________________

Project Supervisor

**Mr. Kiran Chandra Dahal**

Department of Electronics and Computer Engineering, Thapathali Campus

____________________________

External Examiner

____________________________

Project Co-ordinator

**Mr. Umesh Kanta Ghimire**

Department of Electronics and Computer Engineering, Thapathali Campus

____________________________

**Mr. Kiran Chandra Dahal**

Head of the Department

Department of Electronics and Computer Engineering, Thapathali Campus

**March, 2023**

## COPYRIGHT

## ACKNOWLEDGEMENT

We would like to express our gratitude towards the Institute of Engineering, Tribhuvan University for presenting us the opportunity to do a minor project in the sixth semester of Computer Engineering. We are also thankful towards our Department of Electronics and Computer Engineering for the guidance during the project "**Comparative Study for Grammar Error Correction using encoder-decoder and T5 Transformer mode**l".

We would like to acknowledge the authors of various research papers that we have referenced for developing our project. We are thankful to all the people who are directly involved during the study and development process of this project.

We would also like to thank our supervisor, Mr. Kiran Chandra Dahal for his valuable guidance, support, and ideas to initiate and improvise the project.

| | |
|---|---|
| Anuj Rayamajhi | (THA076BCT007) |
| Anup Gelal | (THA076BCT008) |
| Grishma Raj Khanal | (THA076BCT016) |
| Kaustuv Karki | (THA076BCT017) |

## ABSTRACT

Grammar correction by implementing concepts of neural network is a relevant field of interest. Before the introduction of deep learning, grammar was corrected using Rule Based Approach, Classification Based Approach, Machine Translation Approach, and many other methods that were computationally intensive and the content needed to be parsed every time a check was performed. But by using deep learning concepts like RNN, GRU, LSTM, and Transformers, a model could be constructed that would be trained to detect and correct grammatical errors, which in turn, once trained would be able to correct it without any need to cross check any datasets. In this project, we are aiming to implement LSTM encoder-decoder neural network model for performing Grammar Error Correction (GEC). and train this model with lang-8 dataset procured from NAIST Lang-8 Learner Corpora And after achieving proper accuracy, implement this model in a web app.

*Keywords: Corpus, Encoder-Decoder, Grammar Error Correction, Gated Recurrent Unit (GRU), Lang-8, LSTM, RNN, Transformer, T5*

**Table of contents**

## List of Figures

## List of Tables

## List of Abbreviations

| | |
|---|---|
| AM | Attention Mechanism |
| BLEU | Bi-Lingual Evaluation Understudy |
| CPU | Central Processing Unit |
| CUDA | Compute Unified Device Architecture |
| DFD | Data Flow Diagram |
| DNN | Deep Neural Network |
| GEC | Grammar Error Correction |
| GLUE | General Language Understanding Evaluation |
| GPU | Graphics Processing Unit |
| LSTM | Long Short-Term Memory |
| MTS | Multivariate Time Series |
| MVT | Model View Template |
| NLP | Natural Language Processing |
| PDF | Probability Density Function |
| RAM | Random Access Memory |
| RNN | Recurrent Neural Network |
| UI | User Interface |
| VRAM | Video Random Access Memory |

# 1. INTRODUCTION

English is one of the most used languages in the world and might be most popular secondary language. In the process of learning English, writing is most common method, and writing ability is an important indicator of English proficiency. It would not be an understatement to consider English language as one of the most written languages in the entire world. But the problem is many people who use English, are not native speakers or it is not their primary language. This creates one major problem, which is grammar errors. Even natives make grammatical errors when writing something in English. Such grammatical errors are very common in the whole world.

However, affected by native language transfer, English texts created by many people have grammatical errors. Grammar is what defines the structure of sentence. If someone wants to be good at writing essays, poems, novels, then the person must be able to configure the correct words to form a meaningful sentence which pleased the reader. But the simple errors can cause misunderstandings as it could change the perception of the sentence. There are many nuances or cases for English grammar rules which you need to study many years to be error free which not all people will have time neither will to do. So, it is necessary to learn about grammatical construct of English language or at least get it right. Even though, it could be learned, mistakes are inevitable. So, many people go to English professors or editors before publishing articles or books which can cost a lot.

Most of the non-native English speakers find it difficult while writing as there are chances of grammatical errors. So, it can be safely assumed that people would love to have some smart system in which could point out the errors in their writing. Since there are more or less specific rules that needs to be considered while writing something. But the format of writing depends on the intention of the writers. So, the same words can perceive different meanings. We as a human can easily understand the context of the writing. So, we can easily draw the intended purpose of the writing. But we cannot simply write the sets of rules using some programming language so that the compiler can detect the grammatical errors and correct the mistakes. There are many words which are treated in the same way while checking for a grammar. So, it is difficult to figure out the grammatical errors while checking for the similar entities.

The method of correcting a grammar falls under the Natural Language Processing (NLP) task. Nowadays, Deep Learning is being used instead of traditional methods to perform Grammar Error Correction by treating text as a sequential data and training the model to detect errors and improve it as well.

## 1.1 Motivation

Despite many of us learning English as a secondary or even primary language since childhood, the need for improved grammar in written communication in professional or personal life is growing as the importance of digital communication in today's world and the need for error-free written messages. This project aims to mitigate that problem by being able to detect and correct grammar, punctuation, and syntax errors. This will help people communicate effectively with little chance of misunderstanding and more clarity of the work.

## 1.2 Objectives

The major objectives of our project are mentioned below:

- To create an English grammar correction model for error detection and correction.
- To compare the performance of different grammar correction models.

## 1.3 Problem Definition

In this project, we aim to accurately identify and correct grammar errors in text-format. Finding and fixing grammar, punctuation, and syntax issues within a given piece of text would be the problem specification for a grammar correction project. This can include, recognizing and fixing errors in sentence structure, subject-verb agreement, verb tense, and word choice. The project's objective is to make the material clearer and more accessible for intended audience. The research tries to increase the precision of the software.

Table 1-1 : Grammar Sample

| Incorrect | Correct |
|---|---|
| We runs half-marathon every November. | We run half-marathon every November. |
| Do you know Hari | Do you know Hari? |

## 1.4 Scope and Application

The scope of this project is to develop and compare the performance of grammar error correction models using encoder-decoder, encoder-decoder with Attention Mechanism and T5 Transformer models and to develop a site for correcting and identifying errors in English grammar. English language learners, writers, and editors can use the project to improve their grammar and writing abilities. This can also be applied to other areas where written communication is important, such as education, communication, and publishing. To create grammar correction models for those languages, the project can be expanded to include additional languages.

## 1.5 Project Organization

The report is organized into the following sections:

The INTRODUCTION section provides an overview, the motivation, objectives, and scope of the project.

The LITERATURE REVIEW section discusses the related research and studies on grammar error correction using encoder-decoder and T5 Transformer models.

The REQUIREMENT ANALYSIS section discusses the project requirements and feasibility studies for developing the grammar error correction model.

The SYSTEM ARCHITECTURE AND METHODOLOGY section explains the system block diagram, DFD, and the methodology used for developing this project.

The IMPLEMENTATION DETAILS section explains the details of the implementation of the grammar error correction models, the challenges faced, and the solutions adopted.

The RESULT AND ANALYSIS section presents the results of the project and the performance analysis of different grammar error correction models.

The FUTURE ENHANCEMENT section discusses the possible technical improvements that can be implemented in the grammar error correction models.

The CONCLUSION section summarizes the project and discusses the performance of the developed grammar error correction models.

# 2. LITERATURE REVIEW

## 2.1 Grammatical Error Checking Systems: A Review of Approaches and Emerging Directions

Grammatical error checking involves identifying and occasionally correcting problematic words in written material. There are different ways to locate and correct text in various languages. Techniques based on rules, syntax, statistics, categorization, and neural networks have all been used. This article analyzes issues with current grammatical error detection and repair systems, and reviews recent work in the field. It has made some recommendations for the future. [1]

## 2.2 Grammar Error Correction using Neural Network

The performance of the six distinct models was assessed in this study using the GLUE score. With a GLUE score of 0.53, the attention-based word-level model was determined to be the most effective modality out of these six. To operate within computational limitations, this model combined Luong attention with dot scoring function. This study used a simple LSTM rather than a bidirectional LSTM and a little amount of data. [2]

## 2.3 English Grammar Error Detection using Recurrent Neural Network

This study proposed a cyclic neural network-based approach for detecting grammatical errors in English verbs. Grammar mistakes become more frequent because of verb tense irregularities and inconsistencies. Because the LSTM can retrain the valid information during training, it was chosen for this strategy. With the encoding process, each word in the dictionary is mapped to a specific numerical value. As a result, generated vectors are simple to use but they risk losing dimensionality and sequential information order. To preserve the positional arrangement of the text, the data is successively transferred to a lower-dimensional space vector using a text encoder called an embedding model. When this method was put to the test against "Jouku" and "Bingguo," the proposed algorithm came out on top. [3]

## 2.4 An Automatic Grammar Error Correction Model Based on Encoder-Decoder Structure for English Texts

This method recommends utilizing a novel encoder model in place of the conventional encoders to automatically repair grammar errors. In this approach, it suggests reducing grammatical errors for learners, considering that English is one of the most significant information carrier languages. The dual-encoder structure used here is made to capture the details of the source phrase and the context sentence separately. The decoder can integrate the output information when it receives it from the encoder since it is built with a gated structure. Using a self-attention technique, long-distance information is extracted. The goal of this technique was to increase word prediction accuracy using a dynamic search algorithm. Moreover, this technique reduces the likelihood of word repetition. The outcome demonstrates the efficiency and performance of the dual encoder. [4]

## 2.5 Sequence to Sequence Learning with Neural Networks

Deep Neural Networks (DNNs), powerful models, have excelled in challenging learning problems. Sequences can be mapped using DNNs to other sequences, but they cannot be mapped to labeled training sets. This article describes a general end-to-end learning method for sequences that places the fewest constraints on the sequence structure. This method maps the input sequence to a vector of a given dimension using a multilayered Long Short-Term Memory (LSTM), then utilizes a deep LSTM to decode the target sequence from the vector. The team's main discovery is that an English to French translation job from the WMT'14 dataset produced translations by the LSTM that earn a BLEU score of 34.8 on the entire test set, with the LSTM's BLEU score being deducted for terms that are not in its lexicon. The LSTM also had no issues with lengthy sentences. A phrase based SMT system receives a 33.3 BLEU score on the same dataset. When the LSTM is used to re-rank them, the BLEU score of the 1000 hypotheses produced by a forementioned SMT system rises to 36.5, which is close to the prior best performance on this assignment. In addition, the LSTM picked up representations of logical phrases and sentences that are responsive to word order and essentially independent of the active and passive voice. The team also found that changing the word order in all source sentences (but not target sentences) improved the performance of the LSTM by introducing several short-term dependencies between the source and target sentence that simplified the optimization problem. [5]

## 2.6 Effective Approaches to Attention-based Neural Machine Translation

This paper shows some recent works about neural machine translation (NMT) that has been improved by selectively focusing on specific parts of the source text during translation utilizing an attentional mechanism. But there has not been much research into practical architectures for attention based NMT. The global method, which always pays to all source words, and the local approach, which only looks at a subset of source words at a time, are the two classes of [6]attentional mechanisms that are examined in this study. There is a use of WMT translation tasks between English and German in both directions to show the efficiency of both approaches. Using local attention, it significantly outperforms non-attentional systems that already employ well-known strategies like dropout by 5.0 BLEU points. In the WMT'15 English to German translation problem, our ensemble model using several attention architectures produced a new state-of-the-art result with 25.9 BLEU points, an improvement of 1.0 BLEU points over the previous best system supported by NMT and an n-gram reranked. [6]

## 2.7 Attention is All You Need.

Extensive convolutional or recurrent neural networks with an encoder and a decoder are the foundation of the most popular sequence transduction models. The best models also connect the encoder and decoder via an attention mechanism. The researchers proposed Transformer, a new, simple network architecture that does away with both recurrence and convolutions and depends solely on attention processes. Experiments on two machine translation tasks show that these models perform better in terms of quality, parallelizability, and training time. This model outperformed the previous best results, including ensembles, by more than 2 BLEU on the WMT 2014 English to German translation task, achieving 28.4 BLEU. After training for 3.5 days on eight GPUs, our model produced a new single-model state-of-the-art BLEU score of 41.8 on the WMT 2014 English to French translation issue, which is a small fraction of the training costs of the best models from the literature. It is shown how well the Transformer generalizes to other tasks by successfully applying it to English constituency parsing with both large and little quantities of training data. [7]

So, after reading numerous research articles, it was discovered that although applying deep learning techniques to grammar error correction is still being developed, the problem itself is not new. Grammar correction up until this point has been done using a rule-based method, where sentences are regularly reviewed against hard-coded rules. But, by employing a neural network, it can then fine-tune the model and implement it on a variety of devices after training it on a dataset that contains wrong and their matching correct utterances. To compare the results of the three neural network models—Encoder-Decoder LSTM network, Encoder-Decoder LSTM with attention, and finally Transformers plan to employ three different neural network models.

## 2.8 Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer

This paper proposes an NLP model call T5 (Text-To-Text-Transfer-Transformer) model which is based on the transformer model proposed on the paper "Attention is All You Need" by Vaiswani et al. [7] using the text-to-text approach where both the input and the output are text sequences. This model was trained on the C4 (Colossal Clean Crawled Corpus) dataset achieving state-of-the-art results. The C4 dataset was cleaned by only retaining the sentences that end with punctuation and discarding the sentences with smaller length and containing obscene words. The T5- transformer achieves state of the art results on various Natural Language Processing tasks such as machine translation, summarization, text-classification, and other tasks. The authors demonstrate the effectiveness of the T5-model using various benchmarks. The model can be fine-tuned on a small amount of task specific data and still achieve high performance.

The base version of T5 or T5-base for both the encoder and decoder part of the model there are 12 blocks where each block contains a self-attention, encoder-decoder attention, and feed-forward-neural networks. The T5-based model from this paper containing about 220 million parameters has achieved Bilingual Language Evaluation Understudy (BLEU) score of 30.9 and General Language Understanding Evaluation (GLUE) of 82.7 achieving state of the art results. [8]

In general, the literature indicates that models based on Transformers and attention are successful for correcting grammar. On numerous benchmarks, the T5 Transformer type in particular has demonstrated exceptional performance. It is important to keep in mind that these models need substantial computational resources, which may restrict their usefulness in particular situations.

In contrast, however simpler and requiring less processing power, encoder-decoder-based models could not be as effective as attention-based or Transformer-based models. So, the exact task at hand and the available resources would be deciding factor to choose the model to use. The availability and caliber of training data was taken into account when selecting a model for grammar error repair. Although while deep learning models have excelled in several NLP tasks, they need a lot of high-quality training data to work at their best. Yet, getting such information was difficult, particularly for jobs like grammar check that need for annotated text.

Although, the T5 transformer model would perform well, it also has potential drawbacks due to its complexity. Because it consists of a large number of layers and parameters. Thus, making it challenging to interpret and diagnose any type of errors or issues that could come up during training. Moreover, fine-tuning process of the T5 model is a time-consuming and resource-intensive task, limiting its practical use in certain scenarios. Hence, the trade-offs between model complexity and performance, as well as the available resources, when choosing a model for grammar error correction should be considered.

In conclusion, the comparative analysis of attention-based and T5 Transformer-based models for grammatical error correction demonstrates the efficacy of both types of models for this task. The final model selection will, however, be determined by the particular requirements and the resources that are available. Future studies might investigate different strategies, like hybrid models or transfer learning methods, to enhance the performance of grammatical mistake correction models with the least amount of training data or computer resources required.

# 3. REQUIREMENT ANALYSIS

## 3.1 Functional Requirement

The system will take the input sentence from the user using User Interface. When the input sentence is passed into pre-trained model and if there are grammar or punctuation errors in the sentence, the system will be able to correct it. There are three different models in this system. So, the system will be used in an accuracy comparison.

## 3.2 Non-functional Requirements

**Speed requirement:** The system is fast in correcting grammatical mistakes. The speed can be enhanced.

**Efficiency:** This system is efficient in comparing the accuracy.

**Portability and compatibility:** The system can run on every operating system, and it does not conflict with other resources.

## 3.3 System Requirement

### 3.3.1 Software Requirement

The software requirement of the projects are as follows:

**Python:** Python is a high-level programming language that is widely used for developing a variety of software applications. Python supports multiple programming paradigms, including procedural, functional, and object-oriented programming.

**Tensorflow:** Tensorflow is an open-source software library developed by Google that is used for building and training machine learning and deep learning models. Tensorflow provides a platform for developing wide range of machine learning applications. Tensorflow in built to work with various hardware like CPUs, GPUs and TPUs.

**Pandas:** Pandas is an open-source Python library that is used for data manipulation, visualization, and data analysis. Pandas provides various tools for working with structured data.

**Matplotlib:** Matplotlib is a python library that is used for data visualization. Matplotlib provides range of tools for creating various static, interactive, and animated visualizations in Python.

**PyTorch:** PyTorch is an open-source python library that is used for building and training deep learning models. It is designed to be user friendly and flexible for building and training the models.

**Django:** Django is based on Model-View-Template (MVT) architecture, which separates an application into three main components: the data model (Model), controller (Views) and the user-interface display (Template).

### 3.3.2 Hardware Requirements:

Being a software project there are not much hardware requirements. But the training of the model required some resources. We used Google Colab and Kaggle environment to train the models. Their hardware details are listed below:

**Google Colab**

Table 3-1 : Google Colab's Hardware Specification

| Hardware Components | Details |
|---|---|
| CPU | Intel Xeon 2-core 2.2 GHz |
| GPU Model | NVIDIA Tesla K80 |
| Number of GPUs | 1 |
| CUDA Cores per GPU | 2496 |
| Clock speed (GPU) | 560 MHz (base) / 875 MHz (boost) |
| VRAM (GPU) | 12 GB GDDR5 |
| Memory Bandwidth (GPU) | 240 GB/s |
| RAM | 12.72 GB |
| Storage | 107 GB SSD |

**Kaggle**

Table 3-2 : Kaggle's Hardware Specification

| Hardware Components | Details |
|---|---|
| CPU | Intel Xeon 2-core 2.2 GHz |
| GPU model | NVIDIA Tesla P100, Tesla T4 |
| Number of GPUs (P100, T4) | 1, 1 |
| CUDA Cores per GPU (P100, T4) | 3584, 320 |
| Tensor Core per GPU (P100, T4) | 56, 320 |
| Clock Speed | 1.30 GHz (base) / 1.48 GHz (boost),<br>1.60 GHz (base) / 1.77 GHz (boost) |
| VRAM (P100, T4) | 16 GB HBM2, 16 GB GDDR6 |
| Memory Bandwidth (P100, T4) | 720 GB/s, 320 GB/s |
| RAM | 16 GB |

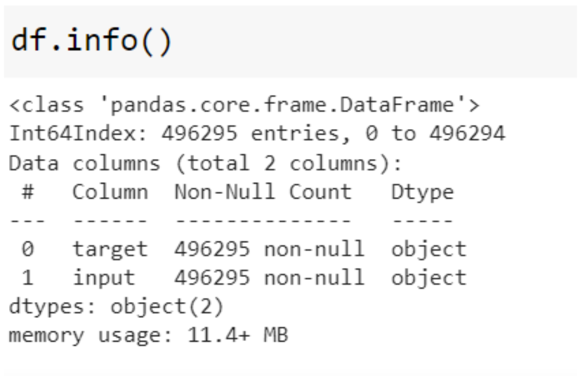
# 4. DATASET ANALYSIS

Before feeding the data to any kind of model, for any purpose, we must analyze it so that we can know which model is the most suitable for the properties of the given dataset. Dataset analysis involves using various statistical and computational techniques to extract meaningful information and insights from large volumes of unstructured textual data. Dataset analysis is the process of examining the dataset to gain a better understanding of its characters, including its size, distribution, and structure. It also helps to identify the strengths and limitations of the datasets. There are two datasets that are used in this project.

## 4.1 NAIST Lang-8 Corpus

One of the datasets is NAIST (Nara Institute of Science and Technology) Lang-8 Corpus of Learner English or Lang-8 for short [9]. It is clean dataset where the sentences by English language learners are corrected by native speakers. As it is not corrected by the users with different English proficiency, which weakens the quality of the datasets. Besides the dataset is not annotated with errors and corrections. The initial lang-8 corpus consists of about 1.1 million sentences that were filtered then the following was found.

Then after removing the duplicates present in the dataset, we are remaining with the following data:



```
df.info()
```

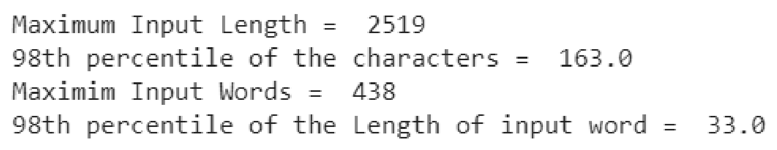
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 496295 entries, 0 to 496294
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   target  496295 non-null  object
 1   input   496295 non-null  object
dtypes: object(2)
memory usage: 11.4+ MB
```

Figure 4-1 : Filtered Dataset Information

For the given data there are two major data columns they are:

### 4.1.1 Incorrect sentences

The total number of sentences in the incorrect column is equivalent to the total number of sentences pair in the dataset and the sentences were further analyzed. After analyzing the incorrect sentences in the dataset, we get the following results.



```
Maximum Input Length =  2519
98th percentile of the characters =  163.0
Maximim Input Words =  438
98th percentile of the Length of input word =  33.0
```

Figure 4-2 : Incorrect Data Length Analysis

From the above information we can determine that 98 percent of the input data falls under 163 character kength limit and 33 word limit.

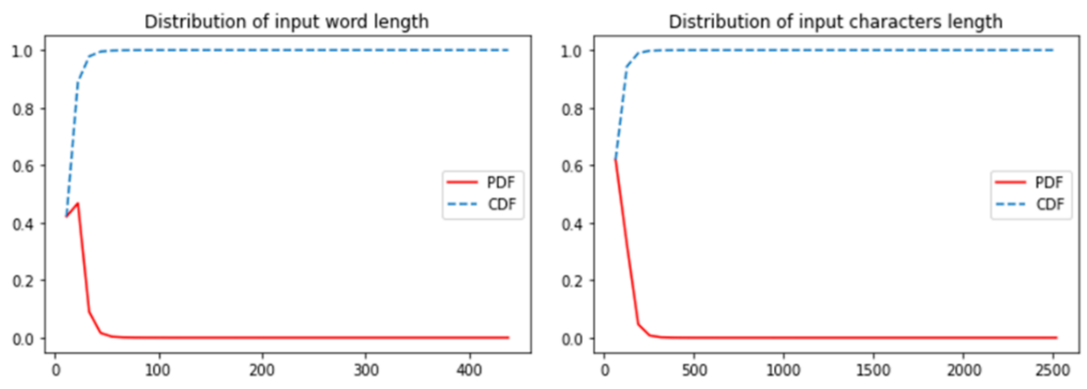


Figure 4-3 : PDF/CDF for incorrect data

From the above probability distribution plots, we can know about the skewness of the data and the probability of which length of words and length of characters have higher probability of occurring. It shows the overall characteristics of the input data.

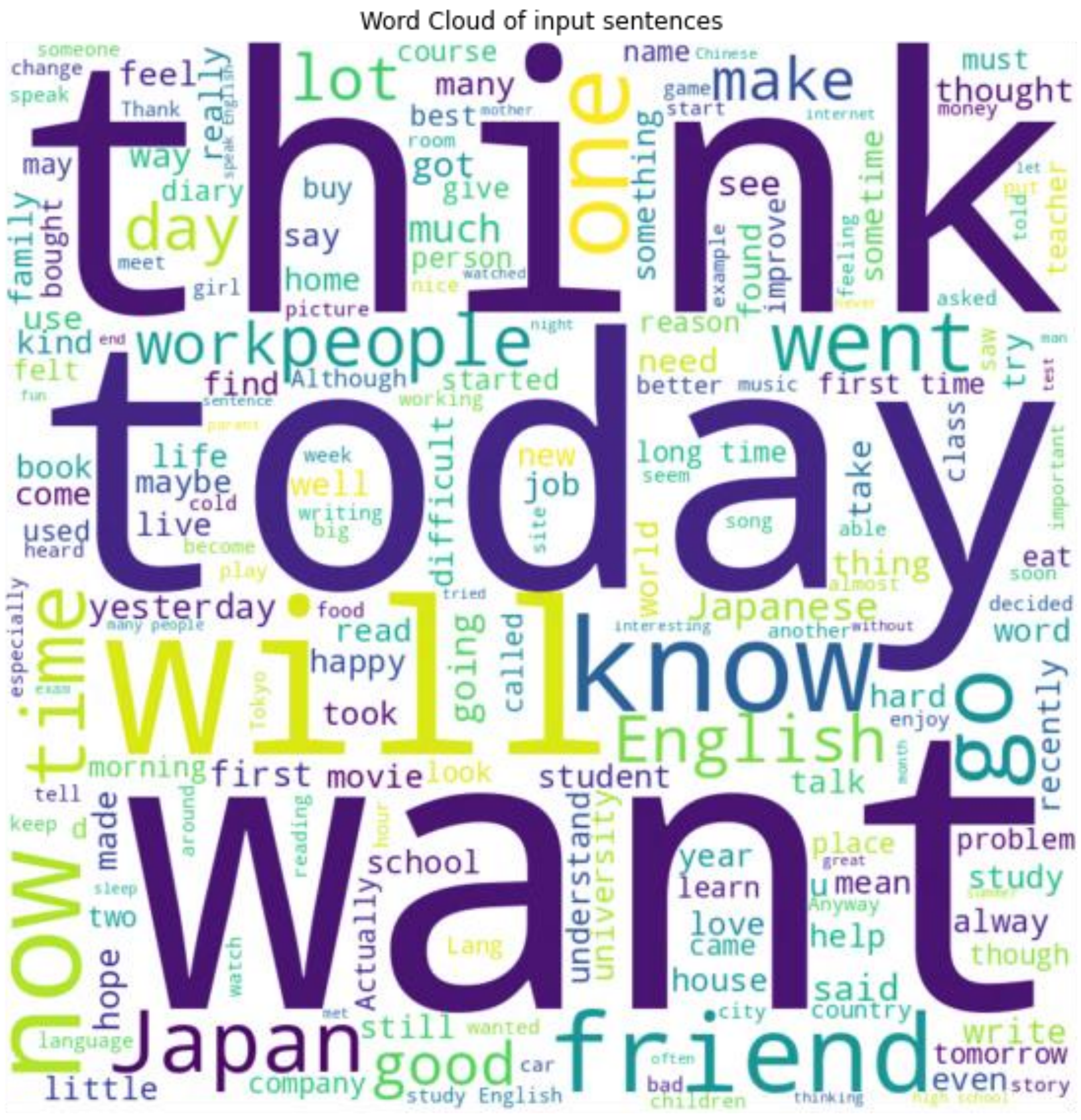
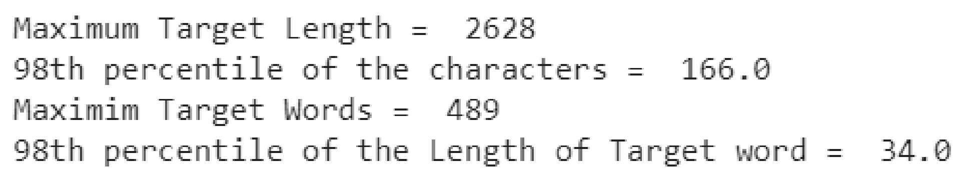


Figure 4-4 : Word Cloud for incorrect data

The above word cloud represents the words in the incorrect column. The higher the frequency of the word present in the dataset the larger the word appears in the word cloud. It can give us a quick overview on the topics present in the data corpus.

### 4.1.2 Correct Sentences

The number of incorrect and correct sentences in the dataset needs to be the same.



```
Maximum Target Length =  2628
98th percentile of the characters =  166.0
Maximim Target Words =  489
98th percentile of the Length of Target word =  34.0
```

Figure 4-5 : Correct Data Length Analysis

From above we can see that the 99 percent of the correct characters are under 189 characters and the 99 percent of the word count are under the 38.

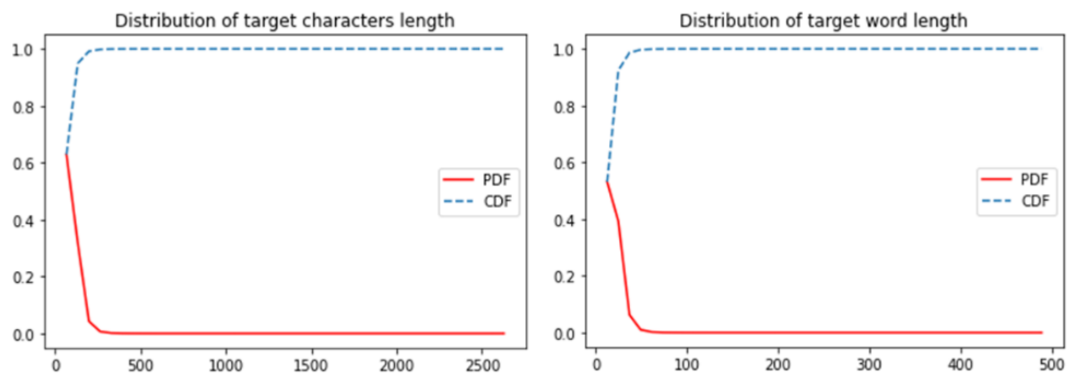


Figure 4-6 : PDF/CDF for Correct Data

From the above probability distribution plots, we can know about the skewness of the data and the probability of which length of words and length of characters have higher probability of occurring. It shows the overall characteristics of the target data.

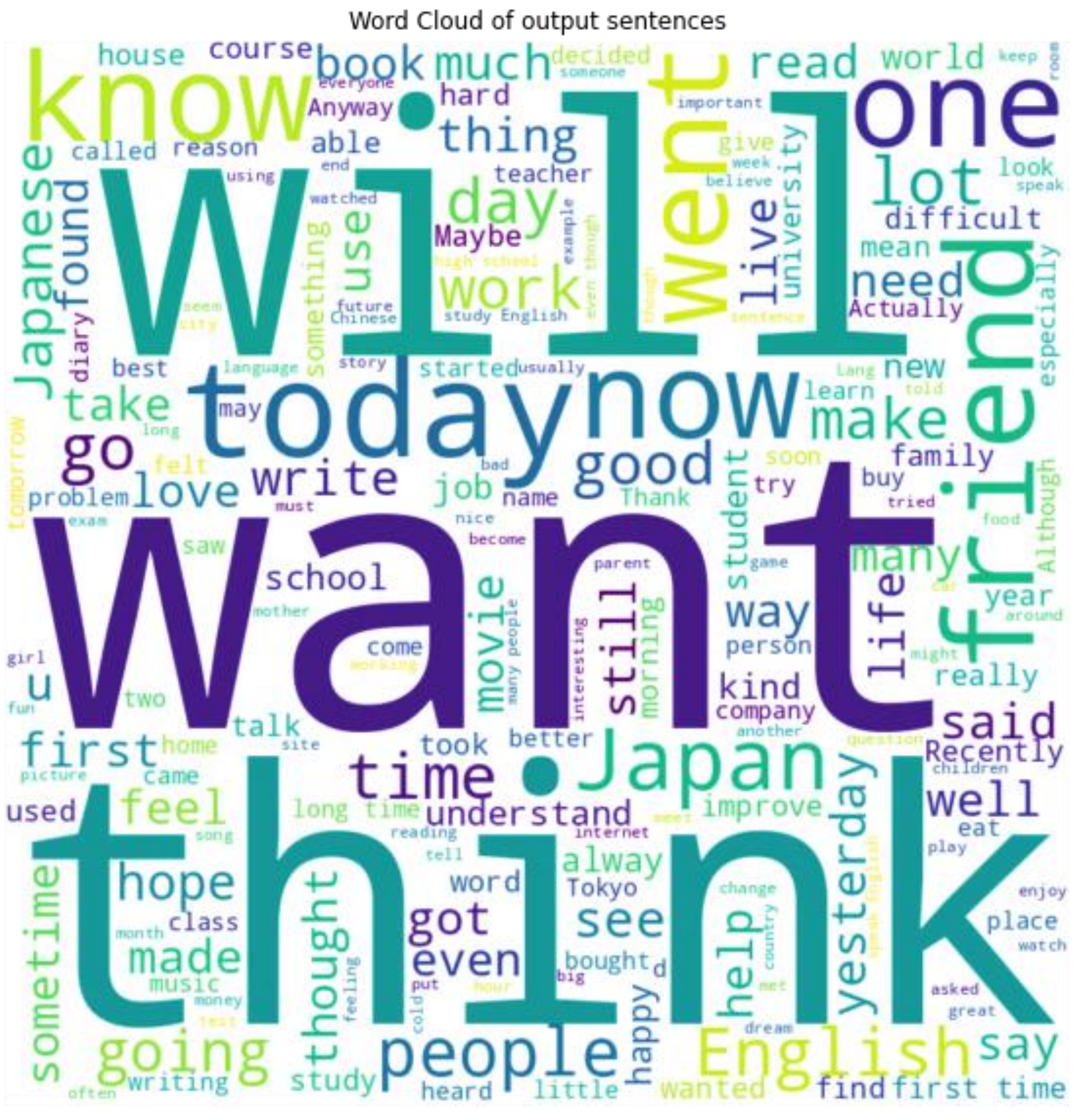
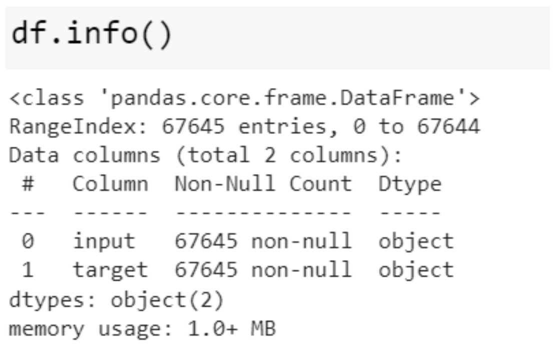


Figure 4-7 : Word Cloud for Correct Data

The above word cloud represents the words in the correct column. The higher the frequency of the word present in the dataset the larger the word appears in the word cloud. It can give us a quick overview of the topics present in the data corpus.

## 4.2 Modified Dataset

This dataset is formed by the combination of J HU Fluency-Extended GUG Corpus (J FLEG), CoNLL-2014 and Colossal Clean Crawled Corpus. The sentences from these predefined datasets were taken and converted into the dataset.



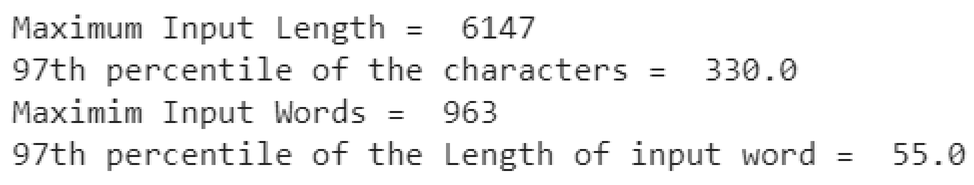
```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 67645 entries, 0 to 67644
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   input   67645 non-null  object
 1   target  67645 non-null  object
dtypes: object(2)
memory usage: 1.0+ MB
```

Figure 4-8 : Filtered dataset analysis for Modified Dataset

The dataset is in the form of two columns one is the input sentence which is also incorrect sentences and the other is target sentence which is the correct form of the input sentences.

### 4.2.1 Input Sentences

The total number of sentences in the input is equivalent to the total number of sentences pair in the dataset and the sentences were further analyzed. After analyzing the input sentences in the dataset, we get the following results.



```
Maximum Input Length =  6147
97th percentile of the characters =  330.0
Maximim Input Words =  963
97th percentile of the Length of input word =  55.0
```

Figure 4-9 : Input Data Length Analysis

From the above information we can determine that 97 percentile of the input sentences fall under the 330 character and 55 words count.

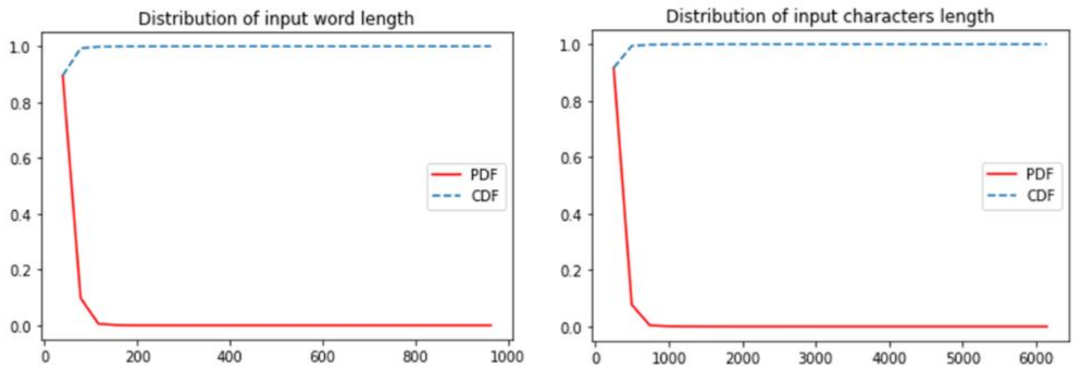


Figure 4-10 :  PDF/CDF for input data

From the above probability distribution plots, we can know about the skewness of the data and the probability of which length of words and length of characters have higher probability of occurring. It shows the overall characteristics of the input data in the modified dataset.

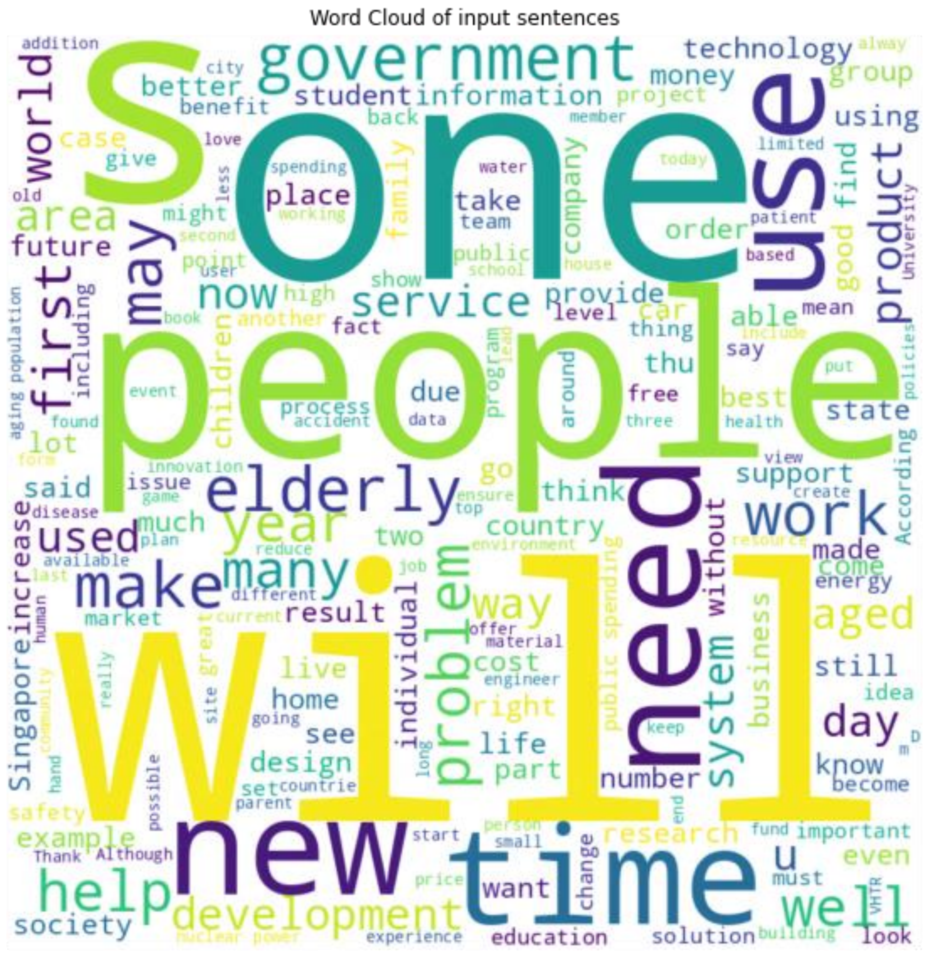


Figure 4-11 : Word cloud for Input dataset

The above word cloud represents the words in the input column. The higher the frequency of the word present in the dataset the larger the word appears in the word cloud. It can give us a quick overview of the topics present in the data corpus.

### 4.2.2 Target Sentences

The total number of sentences in the target is equivalent to the total number of sentences pair in the dataset and the sentences were further analyzed. After analyzing the target sentences in the modified dataset, we get the following results.

```
Maximum Target Length =  984
97th percentile of the characters =  331.0
Maximim Target Words =  149
97th percentile of the Length of Target word =  55.0
```

Figure 4-12 : Target Data Length Analysis

From the above information we can determine that 97 percentile of the input sentences fall under the 331 character and 55 words count.

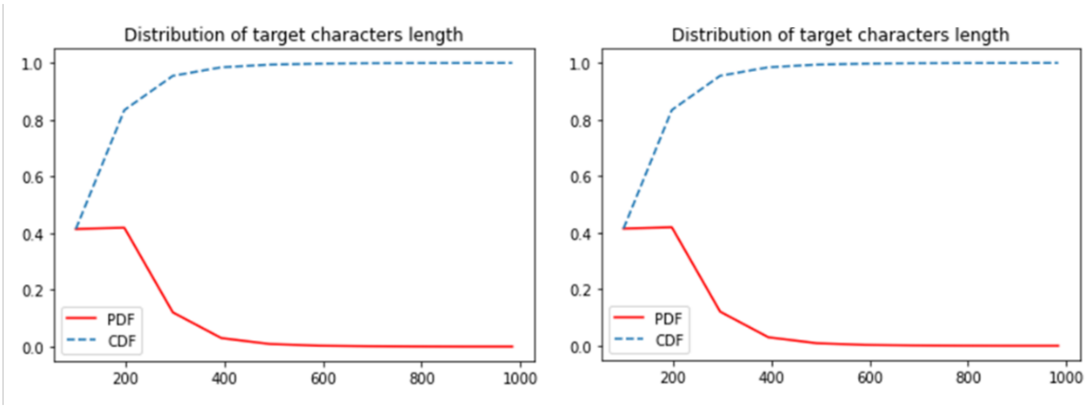


Figure 4-13 : PDF/CDF for Target data

From the above probability distribution plots, we can know about the skewness of the data and the probability of which length of words and length of characters have higher probability of occurring. It shows the overall characteristics of the target data in the modified dataset.

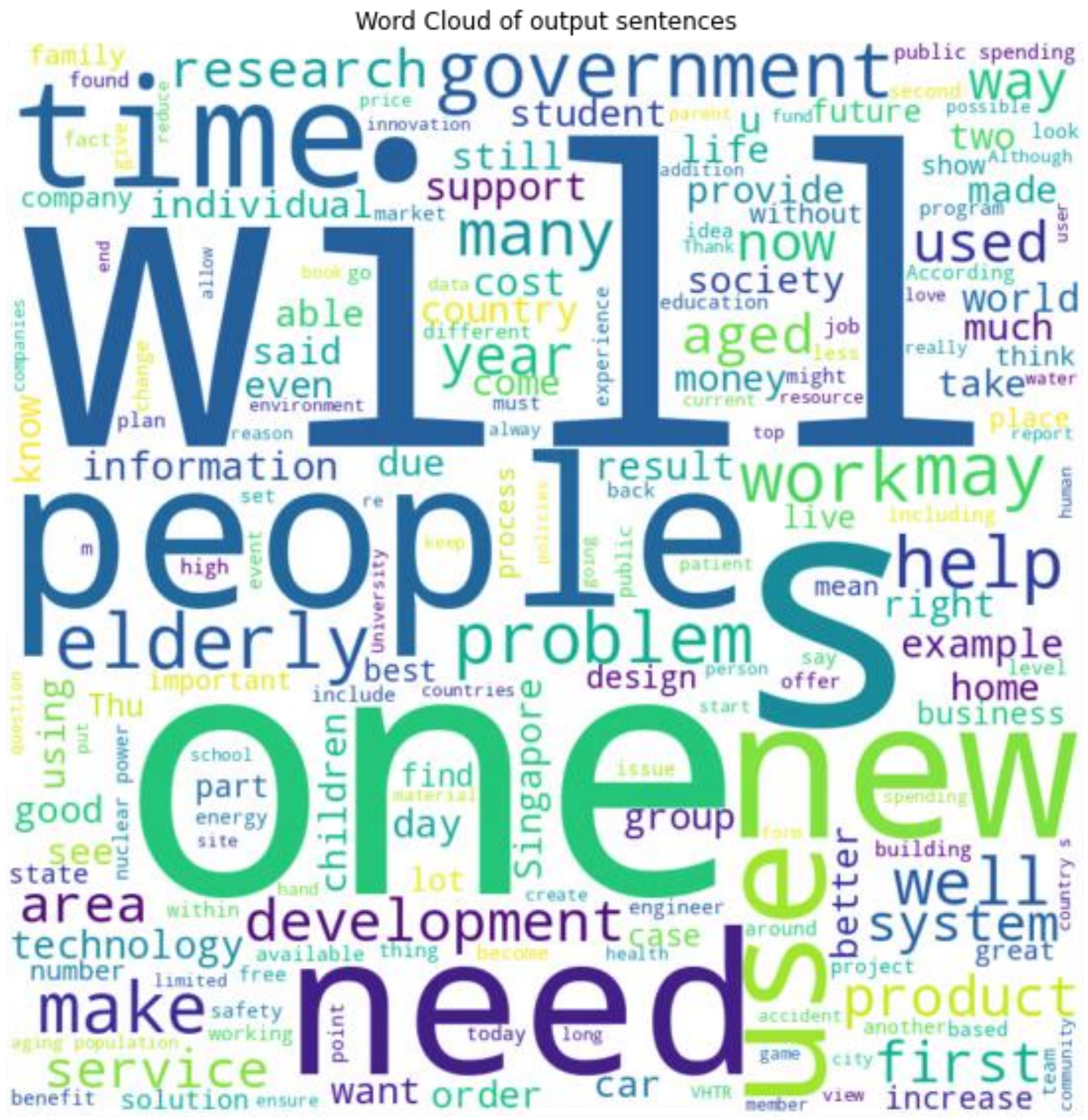


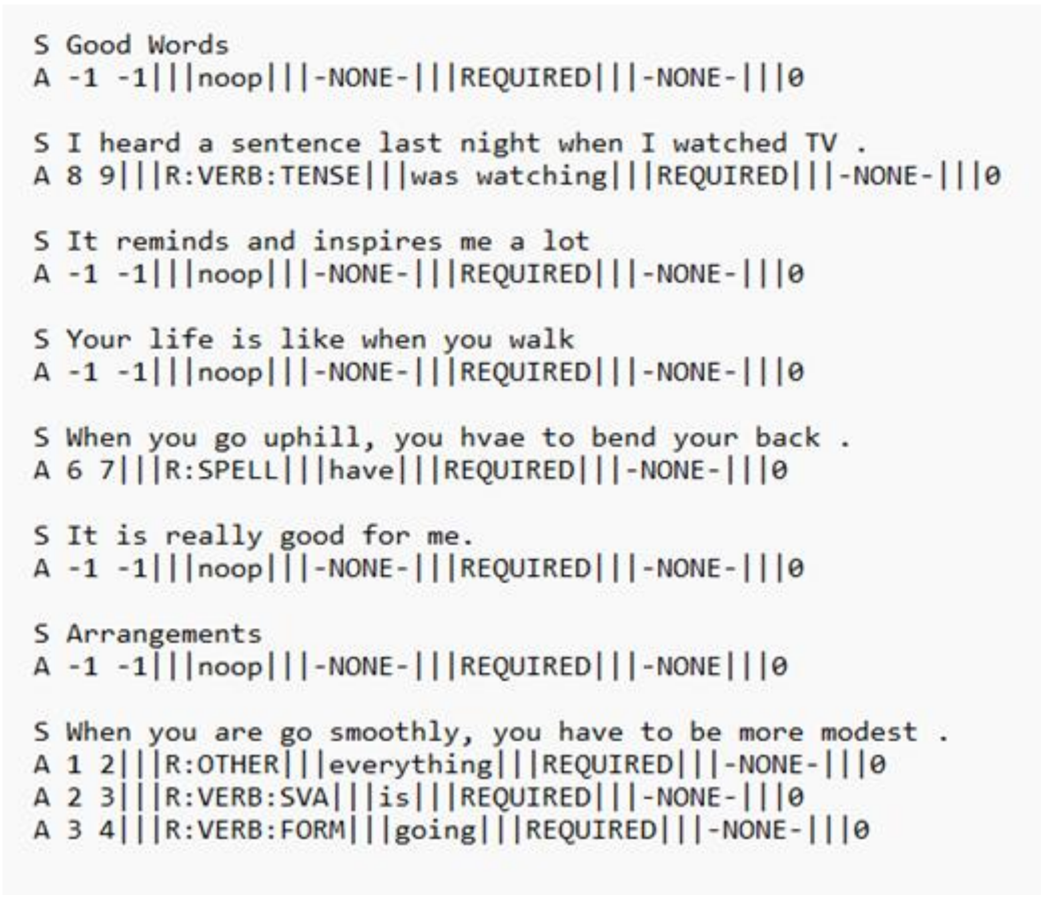
Figure 4-14 : Word Cloud for Target Data

The above word cloud represents the words in the target column. The higher the frequency of the word present in the dataset the larger the word appears in the word cloud. It can give us a quick overview of the topics present in the data corpus.

## 5. METHODOLOGY

This section contains information about the models and techniques we would use to make and present the GEC system.

### 5.1 Training Dataset



```
S Good Words
A -1 -1|||noop|||-NONE-|||REQUIRED|||-NONE-|||0

S I heard a sentence last night when I watched TV .
A 8 9|||R:VERB:TENSE|||was watching|||REQUIRED|||-NONE-|||0

S It reminds and inspires me a lot
A -1 -1|||noop|||-NONE-|||REQUIRED|||-NONE-|||0

S Your life is like when you walk
A -1 -1|||noop|||-NONE-|||REQUIRED|||-NONE-|||0

S When you go uphill, you hvae to bend your back .
A 6 7|||R:SPELL|||have|||REQUIRED|||-NONE-|||0

S It is really good for me.
A -1 -1|||noop|||-NONE-|||REQUIRED|||-NONE-|||0

S Arrangements
A -1 -1|||noop|||-NONE-|||REQUIRED|||-NONE|||0

S When you are go smoothly, you have to be more modest .
A 1 2|||R:OTHER|||everything|||REQUIRED|||-NONE-|||0
A 2 3|||R:VERB:SVA|||is|||REQUIRED|||-NONE-|||0
A 3 4|||R:VERB:FORM|||going|||REQUIRED|||-NONE-|||0
```

Figure 5-1 : Training Data Sample

The above figure shows a sample of the data that would be used to train our system. This data is provided by “NAIST Lang-8 Corpus of Learner English for the 14th BEA Shared Task “ [9]. This dataset contains over a million datapoints, collected by more than 29,000 participants. This dataset needs to be preprocessed as well to make it sensible and usable for our model.

This data is available in ‘m2’ file format, containing lines followed by ‘S’ which denotes an original sentence, containing lines followed by ‘A’, indicating an edit annotation, and there is more than one annotation in case of incorrect sentences.

The dataset provider has also provided the code that will turn the above format of data into csv file which contains two columns for incorrect and correct sentences as shown in below figure.

Table 5-1 : Transformed Dataset

| Input | Target |
|---|---|
| And he took in my favorite subject like soccer. | And he took in my favorite subjects like soccer. |
| His Kanji’s ability is much better than me. | His Kanji ability is much better than mine. |
| Do not know why. | I do not know why. |
| I liked the winter Finland | I liked Finland in the Winter. |
| The making souvenir is a hard and interesting work | Making souvenirs is hard but interesting work. |
| The third memory is the house we lives. | The third memory is the house where we lived. |
| Still she has not recovered from them? | She still has not recovered from them? |
| He is de main character , although appearing others like Cornelius Silla , Metelus , and the Caesar's grandfather . | He is the main character , although others like Cornelius Silla , Metelus , and Caesar's grandfather also appear . |
| So take a transfer class on this Friday | So, I will take a transfer class this Friday |
| The PC using Ubuntu is running well as if it is a new PC ! | The PC running Ubuntu is running as well as any new PC . |

The original datapoints contains a lot of unnecessary words and character making it a very noisy dataset to use. But it was handled by creating a custom function, and whole data was passed through that function to generate a clean dataset. Above image is small section of the whole dataset after data cleaning is done.

For this project, we are going to tokenize the dataset on the word level rather than character level. When implementing a grammar error correction model for the English language, it is better to implement it on a word basis rather than an alphabet basis.

Here are a few reasons why:

- **Word-level correction is more meaningful:** Grammar errors often occur at the level of words or phrases, not individual letters. Correcting errors at the word level will result in more meaningful corrections that are more likely to improve the overall grammaticality of the sentence.
- **Contextual information is more readily available at the word level:** Correcting errors at the word level allows the model to take into account the context in which the word appears. This can be important for determining the correct correction, as the same error might have different correct corrections in different contexts.
- **Word-level correction is more efficient:** Correcting errors at the word level can be more efficient than correcting errors at the letter level. The model will not need to consider all combinations of letters to correct the error but can instead focus on the relevant word or words.

### 5.2 General Neural Network Architecture

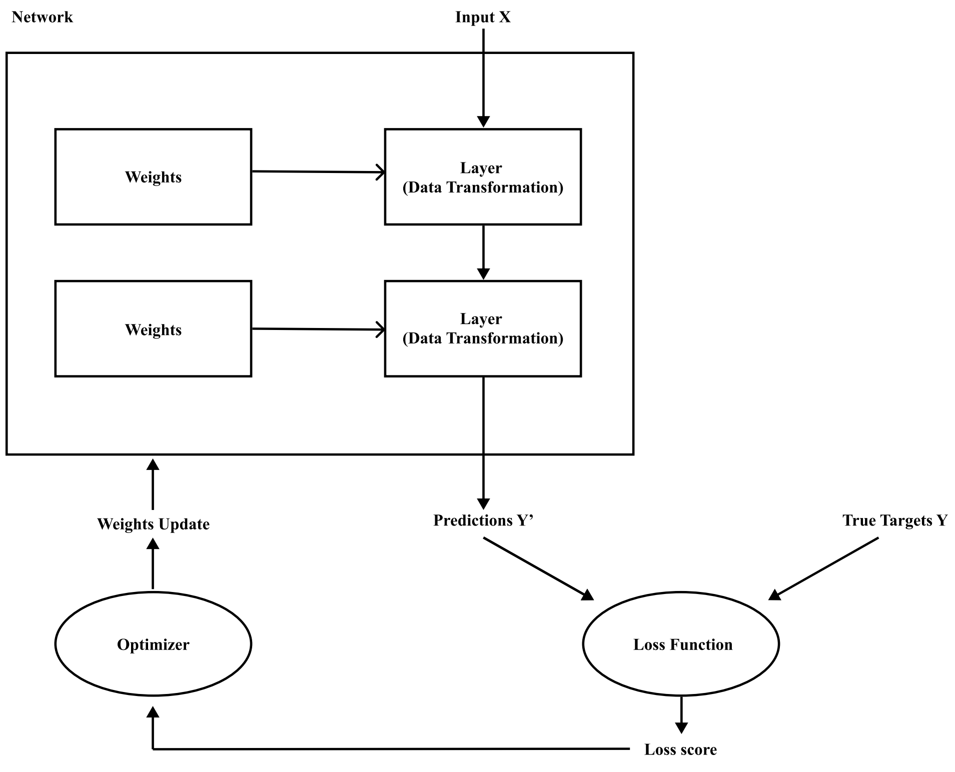


Figure 5-2 : Block diagram of General Neural Network Architecture

Neural Network in general contains different layers of neurons interconnected with each other. Based on the type of neurons used, Artificial neural network can be classified into various categories like, Multilayer Perceptron Network, Convolutional Neural Network, Recurrent Neural Network and so on.

Within these layers there are multiple weights associated with each neuron and these weights are changed while training the network with certain data X and target data Y. The function used to calculate the distance of predicted value from the target value is called Loss function. This calculates loss score and bases on this score another function called optimizer updates the weights within the network.

Thus, selection of Loss function and Optimizer also plays the vital role in the performance of the model.

## 5.3 Long Short-Term Memory (LSTM)

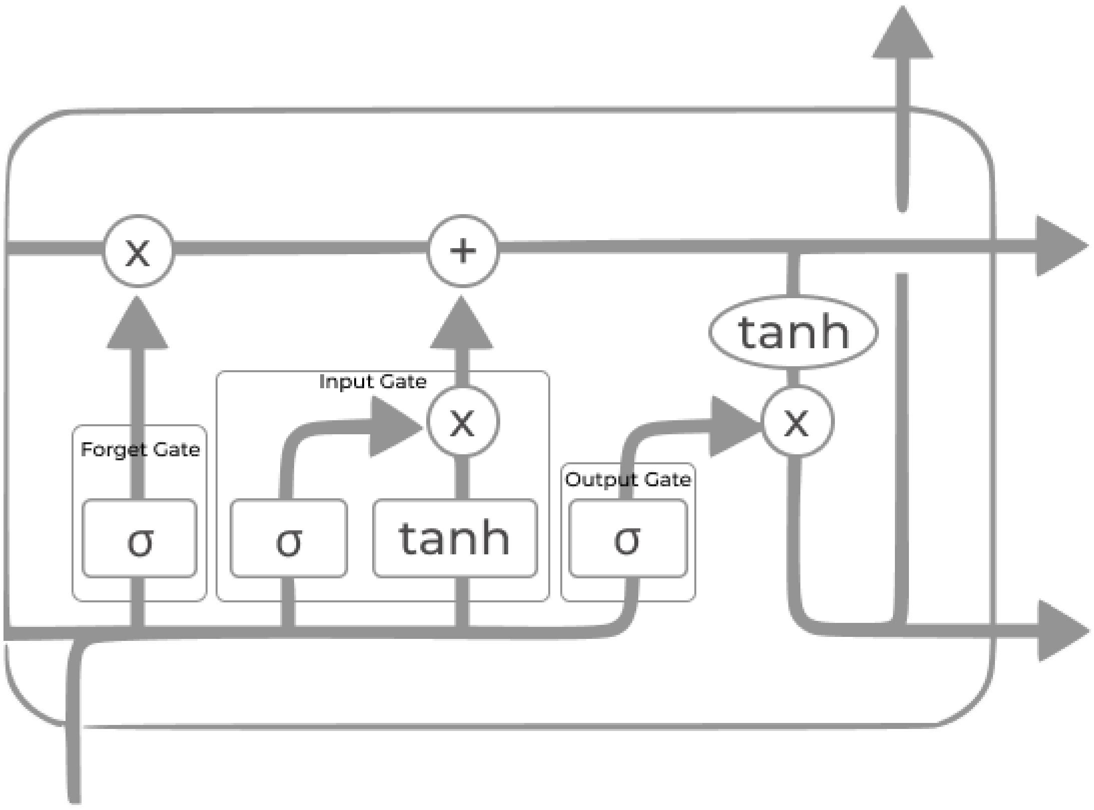


Figure 5-3 : Long Short-Term Memory cell

Long Short-term Memory networks (LSTM) is a variant of RNNs, capable of adapting to long-term dependencies, like in long sequence predictions. It has feedback connections. They are implemented in tasks like speech recognition, machine translations, etc.

One of the distinct features of LSTM is cell state, '$C_t$.' It can be seen in the diagram above, the top horizontal line, is the line through which the cell state information flows, which could be changed or unchanged, regulated by the status of various gates: Forget Gate, Input gate, and Output gate. These gates optionally let the information flow in or out.

The node '*' in the diagram denotes pointwise multiplication and '+' node denotes addition. '$X_t$' represents the input of the present context time, '$C_{t-1}$' represents the cell state of previous time stamp, '$h_{t-1}$' denotes the hidden layer state of previous time stamp. Each of these parameters has a different weight associated with it at different nodes within the cell and each node has their own bias value.

Forget gate implements sigmoid activation function, by summing up the product of incoming parameters with its associated weight and its bias value. The output of the forget gate is then multiplied, pointwise, to the previous cell state. This determines the importance of the previous cell information, if not, cell state would be rendered to zero value. In similar manner, based on the diagram, input and output gates are handled.

### 5.3.1 Sigmoid Activation Function

A Sigmoid function is a mathematical function which has a characteristic S-shaped curve. Sigmoid function is normally used to refer specifically to the logistic function, also called the logistic sigmoid function.

$$S(x) = \frac{1}{1 + e^{-x}} \qquad [5\text{-}1]$$

All sigmoid functions have the property that they map the entire number line into a small range such as between 0 and 1, or -1 and 1, so one use of a sigmoid function is to convert a real value into one that can be interpreted as a probability.

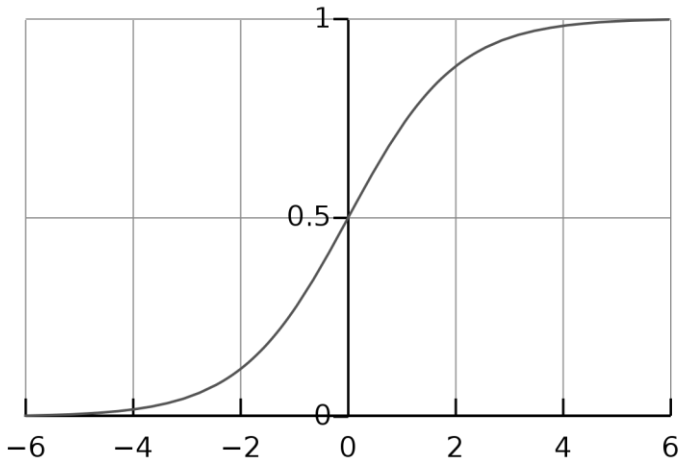


Figure 5-4 : Sigmoid Activation Function Plot

### 5.3.2 Tanh Activation Function

The range of the tanh function is from -1 to 1, making it better than the logistic sigmoid function. Tanh is sigmoidal as well (s-shaped). The positive aspect of this is that the zero inputs will be mapped near zero and the negative inputs will be highly negative in the tanh graph. The function might take numerous forms. While the derivative of the function is not monotonic, the function itself is. The tanh function is mostly used for classifying data into two groups.

$$Tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad [5\text{-}2]$$

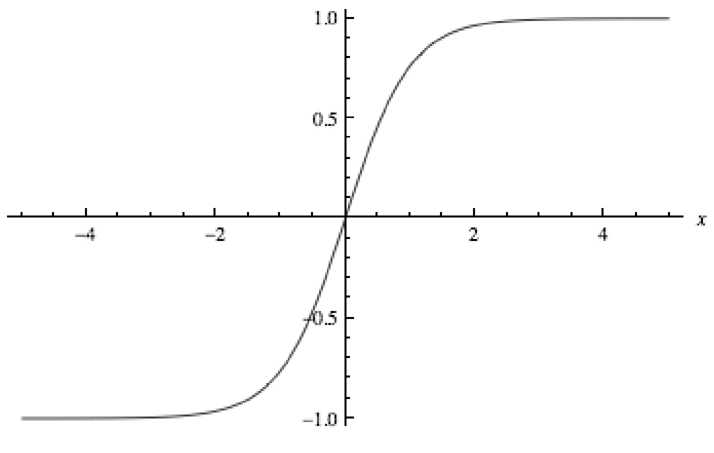


Figure 5-5 : Tanh Activation Function Plot

## 5.4 Loss Function

The loss function is the function that determines how far the algorithm's current output is from what is desired. It is a technique for assessing how well an algorithm mimics the data. Cross-entropy loss function is the type of loss function used to train neural networks.

In machine learning, this loss function is frequently employed, particularly for classification tasks. It calculates the discrepancy between the true probability distribution of the classes in the dataset and the projected probability distribution. Other names for the cross-entropy loss include log loss and SoftMax loss.

The amount of information included in a probability distribution is measured by entropy. The entropy increases with the degree of ambiguity or randomness in the distribution. Entropy is a mathematical concept that is used to quantify the randomness or uncertainty of a predicted probability distribution. The difference between the true probability distribution and the anticipated probability can be measured using the cross-entropy loss function.

The cross-entropy loss function is defined as:

$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij}) \qquad [5\text{-}3]$$

where:

- $N$ is the number of samples in the dataset
- $C$ is the number of classes
- $y_{ij}$ is a binary indicator (0 or 1) whether the true label of the sample ‘i’ is class ‘j’
- $\hat{y}_{ij}$ is the predicted probability of sample ‘i’ belonging to class ‘j’

The degree to which the projected probability distribution resembles the actual probability distribution can be determined by looking at the cross-entropy loss function. The cross-entropy loss decreases with the proximity of the two distributions. On the other hand, the cross-entropy loss increases with the degree of divergence between the two distributions.

In classification problems where the objective is to forecast the probability distribution of the classes for each input, the cross-entropy loss function is frequently utilized. In such tasks, the output of the model is often converted into a probability distribution over the classes using the SoftMax function, which is typically used to obtain the anticipated probability distribution.

The cross-entropy loss is calculated for every training example and then averaged over the whole training set throughout training. By modifying the model parameters, the training procedure aims to reduce the cross-entropy loss. Usually, an optimization technique like stochastic gradient descent is used for this.

When the target variable is a single integer encoding the class label rather than a one-hot encoded vector, the sparse categorical cross entropy loss function is utilized. In multi-class classification problems, where the objective is to predict a single class label for each input, it is frequently utilized.

The sparse categorical cross entropy loss function is defined as:

$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\hat{y}_i^{(y_i)}\right) \qquad [5\text{-}3]$$

Where:

- N is the number of samples in the dataset
- $y_i$ is the true label of the sample ‘i’ (an integer between 0 and C-1, where C is the number of classes)

The sparse categorical cross entropy loss function simply anticipates a single integer indicating the true class label, as opposed to the regular cross-entropy loss function, which anticipates a one-hot encoded vector for the target variable. The SoftMax function, which transforms the model's output into a probability distribution over the classes, is often used to determine the anticipated probability distribution from the input.

The sparse categorical cross entropy loss is calculated for every training example and averaged over the whole training set during training. By modifying the model parameters, the training procedure aims to reduce the sparse categorical cross entropy loss. Ordinarily, an optimization is used to do this.

## 5.5 Optimizer

An optimizer is an algorithm that modifies a machine learning model’s parameter to reduce the loss function and raise the model's accuracy. Based on the gradient of the loss function with respect to the parameters, the optimizer adjusts the model's parameters iteratively.

Adam (Adaptive Moment Estimation), a well-liked optimization method for neural network training, combines the advantages of stochastic gradient descent and gradient descent. For large-scale learning applications, it is an adaptive learning rate optimization algorithm.

Based on estimates of the first and second moments of the gradients, it calculates an adaptive learning rate for each parameter. The first instant is the gradients' mean, while the second moment is their uncentered variance. Scaling the learning rate by the ratio of the first moment estimate to the square root of the second moment estimate yields the adaptive learning rate for each parameter.

Mathematically, the update rule for Adam can be expressed as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad [5\text{-}4]$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad [5\text{-}5]$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad [5\text{-}6]$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad [5\text{-}7]$$

$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \quad [5\text{-}8]$$

Where:

- ‘t’ is the iteration number
- $\theta_t$ is the vector of model parameters at iteration ‘t’
- $g_t$ is the gradient of the loss function with respect to $\theta_t$
- $m_t$ is the exponentially weighted moving average of the gradients, with momentum parameter $\beta_1$ in (0,1)
- $v_t$ is the exponentially weighted moving average of the squared gradients, with decay rate parameter $\beta_2$ in (0,1)
- $\widehat{m}_t$ and $\hat{v}_t$ are bias-corrected estimates of $m_t$ and $v_t$, respectively
- $\alpha$ is the learning rate, which controls the step size of the updates
- $\epsilon$ is a small constant used to prevent division by zero

The magnitudes of the gradients and their second moments serve as the basis for how the ADAM optimizer adjusts the learning rate for each parameter. The adaptive learning rates increase the algorithm's robustness and convergence speed, while the momentum term smooths out updates and prevents oscillations.

The Adam Optimizer has several benefits, including:

- To speed up convergence and prevent becoming stuck in local minima, it adjusts the learning rate for each parameter based on the first and second moments of the gradients.
- Memory use efficiency: Compared to other optimization methods that store the entire gradient, Adam just stores the first and second moments of the gradient.
- Robustness to noisy gradients: Compared to other optimization techniques, Adam can manage noisy gradients and is less susceptible to the selection of the hyperparameters.

## 5.6 Encoder Decoder model

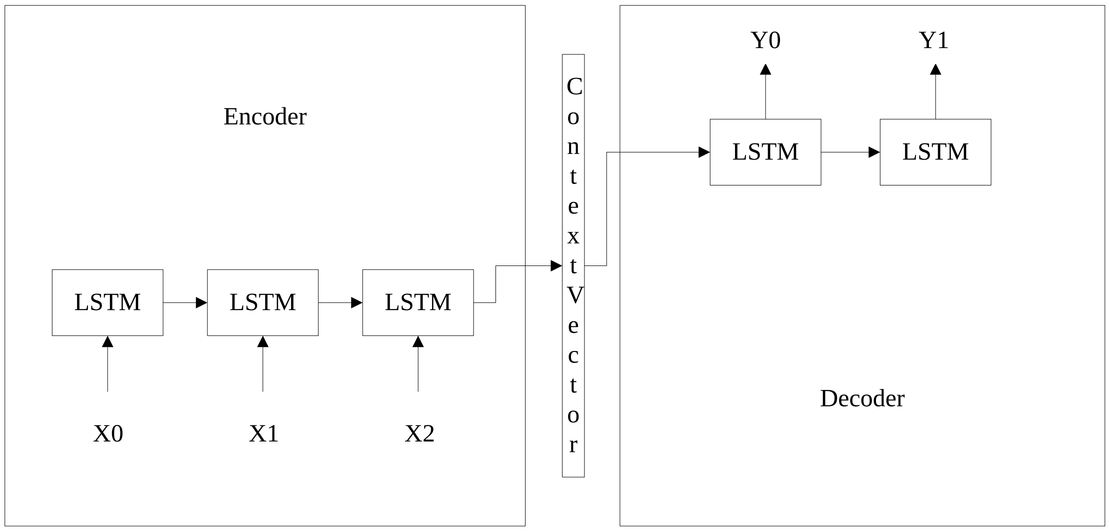


Figure 5-6 : Encoder-Decoder general structure

It is not reasonable, that the neural network for processing sequential data should output a vector from a sequence or vice versa. And while doing so, the network would not be able to manage variable length data. So, encoder decoder model was engineered, which is designed to manage sequence to sequence processing problems.

Grammar Error Correction (GEC) is surely a sequence-to-sequence problem, since a correct sequence of words is to be generated from the incorrect one. We have proposed to use encoder decoder consisting of purely LSTM cells only.

The encoder would take a sentence as an input and represents in form of hidden and cell state. After going through the layers of LSTM cells, the encoder would output a context vector/Encoded vector, which is of a certain fixed length and is built in a way that it would try to capture the whole meaning of the input-sentence and would assist the decoder make accurate corrections. The decoder is initialized by passing a start token for the sentence, expecting the model to make accurate predictions for next word in sequence. The decoder stops when it encounters an end token for the sentence.

## 5.7 Encoder-Decoder LSTM Network Architecture

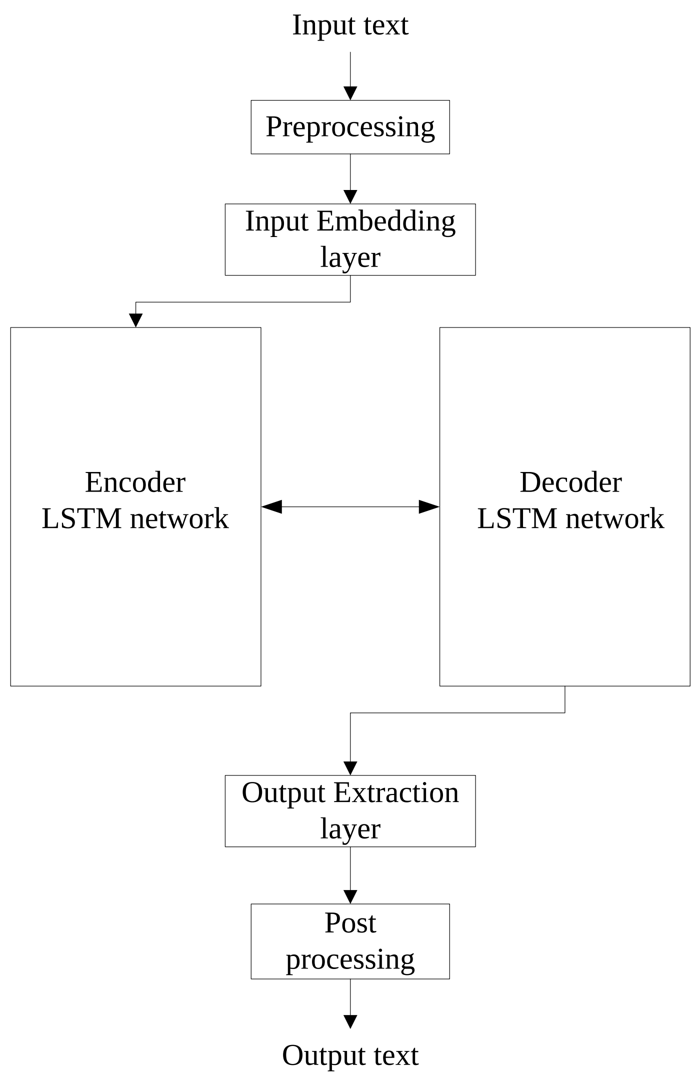


Figure 5-7 : Encoder Decoder Architecture

This block diagram basically represents the architecture of neural network, we are going to use for handling grammar correction. Grammar correction is basically a language problem, so it is required to remember the words while processing them, which could also be called context. Since a vanilla neural network is not capable of remembering context, it is necessary to use a neural network which does. Such neural networks are called Recurrent Neural Networks (RNNs). RNN is bit different from the normal neural network. Neural network contains three layers: Input layer, Hidden Layers, Output layers. When training or using this simple neural network, it does not have concern with past values. But since for GEC problem, the occurrence of past words matters. RNN, in turn, processes especially time series data, and the previous state of the hidden state, is passed to the present state, in form of a feedback, which eventually saves the context present in the given sequential data. So, neural networks like RNNs are usable in the case of this project.

But, a normal RNN also suffers, when the sequence of data to be processed is long. It suffers from vanishing gradient problem, which results in minimal learning rate, which is not a desirable situation. So, another upgraded form of simple RNN, which is called Long Short-Term Memory (LSTM) cells is used when managing large sequences of data, which does not suffer from vanishing gradient problem as much as a simple RNN suffers from.

Since, this model is aimed to correct grammar of sentences with variable length of words. A normal combination of LSTMs would not be enough. Thus, a sequence-to-sequence structure, Encoder Decoder structure of LSTMs is to be implemented. It is known as sequence to sequence because the encoder part creates a fixed length encoded vector from a variable length sequence of data and which is again converted into a variable length output sequence after the decoder decodes the fixed length vector. Hence, sequence to sequence model.

Based on the model of the neural network, the input should be processed, such that it would be accepted by the network as well, or to prevent any type of warnings or errors during implementation. And in similar manner, the output would also be the whole sentence itself at once, so a post-processing mechanism is also necessary.

## 5.8 Encoder-Decoder LSTM Network with Attention

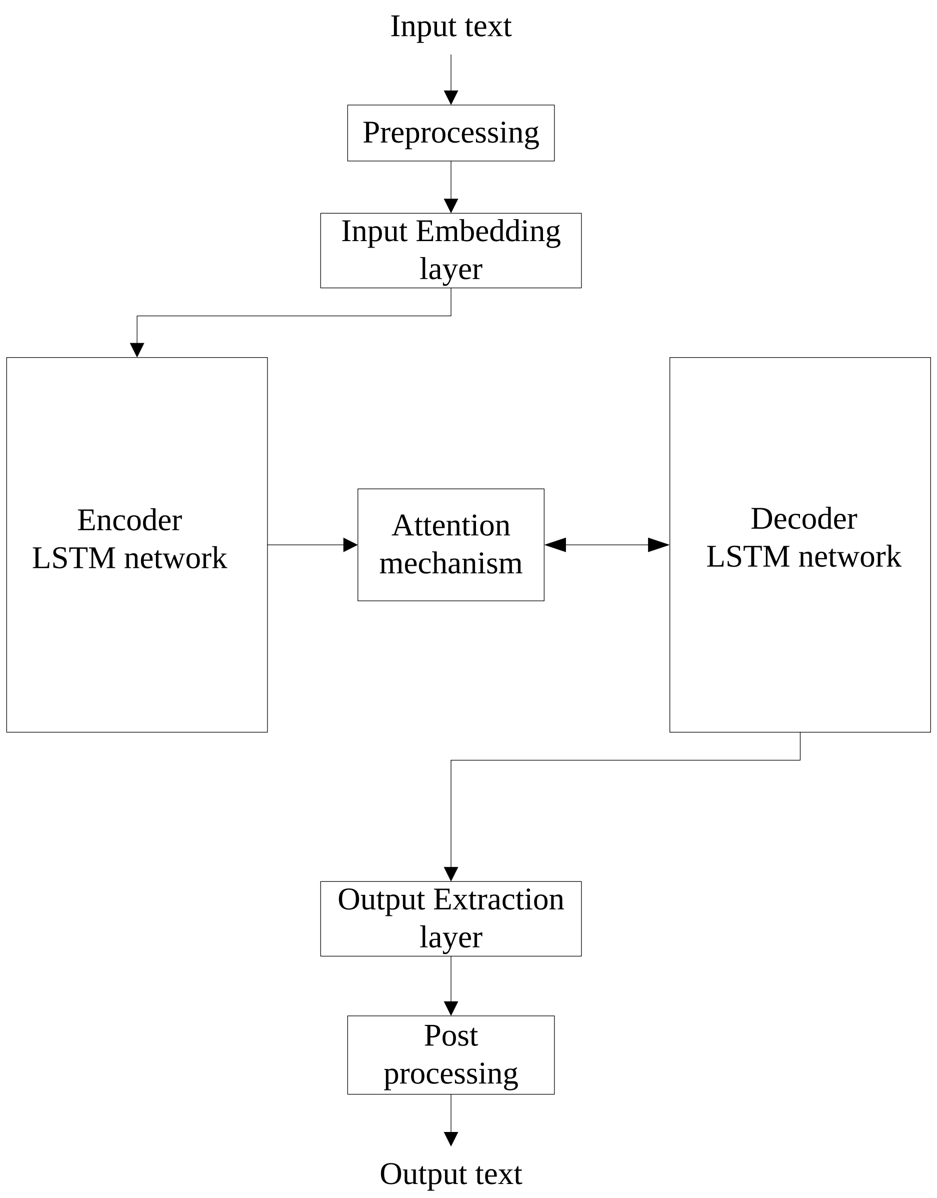


Figure 5-8 : Attention Model Architecture

Preprocessing step is same as in previous model. The basic data flow in above model is explained in points below:

- The encoder takes in the input sequence of integers as its input.
- The input sequence is then passed through an embedding layer to convert each integer into a dense vector representation.
- The embedded input sequence is then passed through an LSTM network to produce a hidden state representation.

- The decoder takes the hidden state representation from the encoder as its input.
- The attention mechanism computes a weight for each timestep of the encoder's hidden state representation, which allows the decoder to focus on different parts of the encoder's hidden state representation.
- The decoder combines the weighted encoder hidden states with its own hidden state to produce its output.
- The decoder's output is passed through a dense layer to produce the final prediction, which is a probability distribution over the target vocabulary.

Finally, the model is trained using the input and target sequences, with the goal of minimizing the difference between the model's predicted output and the target sequence.

## 5.9 Attention Mechanism

The attention mechanism is a key component of the sequence-to-sequence models. It allows the decoder to focus on different parts of the encoder's hidden state representation when producing its output, which can help improve the quality of the decoder's output.

Here is how the attention mechanism works in more detail:

- The attention mechanism computes an attention weight for each timestep of the encoder's hidden state representation. These attention weights represent the importance of each timestep of the encoder's hidden state representation for the decoder's current output.
- The attention weights are computed using a dot product between the decoder's current hidden state and each timestep of the encoder's hidden state representation, followed by a SoftMax activation to ensure that the attention weights sum to 1.
- The attention mechanism then uses the attention weights to compute a weighted sum of the encoder's hidden state representation. This weighted sum, also known as the "context vector," represents the information from the encoder that is relevant for the decoder's current output.
- The decoder combines the context vector with its own current hidden state to produce its output.

The attention mechanism allows the decoder to dynamically focus on different parts of the encoder's hidden state representation as it produces its output, which can help to improve the accuracy of the decoder's output. This is especially useful in Seq2Seq problems where the target sequence is much longer than the input sequence, as it allows the decoder to selectively attend to different parts of the input sequence as needed.

**5.10 Transformers**

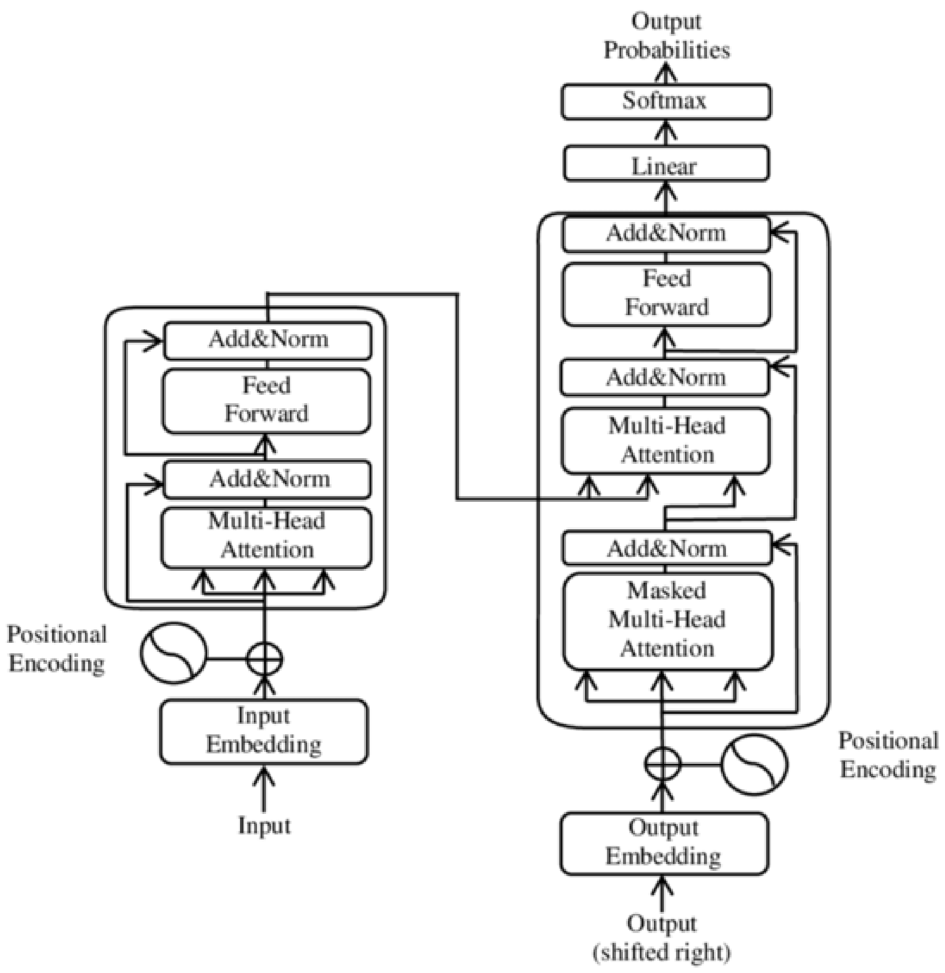


Figure 5-9 : Transformer Architecture

A transformer model is a type of deep learning architecture that has been shown to be very effective in various NLP tasks.

The transformer model can used to detect and repair grammar problems in new text inputs after being trained on a sizable corpus of text with corrected grammar. The model takes a word sequence as input and outputs a word sequence with improved grammar.

The self-attention mechanism, which enables the model to consider the relative weights of various words in the input sequence when making predictions, is the foundational element of the transformer model. The model can better comprehend the relationships between words and the context thanks to this technique, which also helps it decide which adjustments to apply. The transformer model also makes use of multi-head attention, which enables the model to pay attention to various input sequence segments concurrently, enhancing the model's capacity to recognize context. A feed-forward neural network processes the output of the self-attention mechanism before making the final predictions for each word in the output sequence.

The transformer model architecture consists of several key components, including an input layer, self-attention mechanism, feed-forward neural network, and output layer.

- **Input layer:** The input layer takes in a sequence of words, which are typically represented as word embeddings. Word embeddings are dense, continuous-valued representations of words that capture the semantic and syntactic meaning of a word.
- **Self-attention mechanism:** The self-attention mechanism is the core component of the transformer architecture. It allows the model to weigh the importance of different words in the input sequence and calculate a weighted sum of the word embeddings. This mechanism is implemented through multi-head attention, which allows the model to attend to different parts of the input sequence in parallel.
- **Feed-forward neural network:** The output from the self-attention mechanism is passed through a feed-forward neural network, which consists of multiple fully connected layers. The neural network processes the information from the self-attention mechanism and makes predictions for each word in the output sequence.
- **Output layer:** The output from the feed-forward neural network is used to make predictions for each word in the output sequence. These predictions are typically decoded into text and compared to the ground-truth text to evaluate the model's performance.

### 5.10.1 Multi-Head Attention

A crucial feature of the transformer model design is multi-head attention, which is utilized to weigh the significance of various input sequence segments when formulating predictions. Instead of needing to handle each portion of the input sequence sequentially, it enables the model to handle many sections simultaneously.

In multi-head attention, the input sequence is first transformed into queries, keys, and values, which are then used to calculate attention scores for each word in the input sequence. These attention scores represent the importance of each word in the sequence with respect to the current prediction being made.

The attention scores are then used to calculate a weighted sum of the values, which represent the word embeddings. This weighted sum is concatenated and then transformed again to produce the final output, which is used in the next layer of the model.

One of the advantages of using multi-head attention is that it allows the model to attend to different aspects of the input sequence in parallel. For example, one head may attend to the previous words in the sequence, while another head may attend to the context of the current word, and so on. This allows the model to capture different types of information about the input sequence and make more informed predictions.

### 5.10.2 Attention Scores

Attention scores are values that represent the importance or relevance of each item in a sequence with respect to a particular task or prediction. In the context of NLP and deep learning models, attention scores are used to weigh the importance of each word in a sequence when making predictions.

In the transformer model architecture, attention scores are calculated as a dot product of the queries, keys, and values that are calculated from the input sequence. The attention scores represent the similarity between the queries and keys and are used to calculate a weighted sum of the values, which represent the word embeddings.

The attention scores are used to control the flow of information in the model, allowing the model to focus on the most relevant parts of the input sequence when making predictions. The attention mechanism in the transformer model allows the model to attend to multiple parts of the input sequence in parallel, which results in improved context awareness and performance on various NLP tasks, including grammar error correction.

## 5.11 T5 Transfer Transformer Model

A potent neural network model called the T5 (Text-to-Text Transfer Transformer) is used for natural language processing (NLP) applications like translation, summarization, and question answering, among others. It was first presented by Google AI in 2019, and since then, it has grown to be one of the most popular transformer models in NLP.

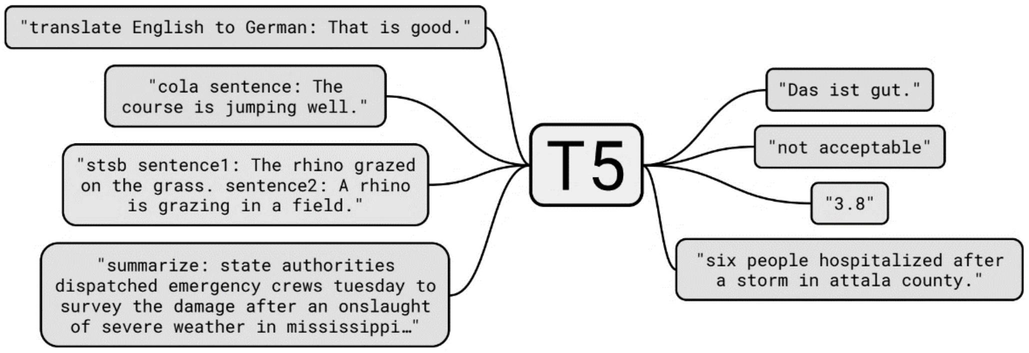

"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather in mississippi…"
T5
"Das ist gut."
"not acceptable"
"3.8"
"six people hospitalized after a storm in attala county."

Figure 5-10 : T5 Transfer Transformer Model

As a text-to-text model, the T5 model is trained to accept input text in one form and output text in another. It can be trained, for instance, to take an English sentence as input and output the same sentence in French. Unlike models like BERT, which are intended to take in text and generate a prediction or classification, this one does not. Each layer of the T5 model's encoder and decoder architecture is made up of multi-head attention and feedforward networks. The input text is processed by the encoder layers, while the output text is produced by the decoder levels.

Pretraining is a method used during training to train T5 on a huge corpus of text. Pretraining entails teaching the model to do a variety of tasks, such as predicting missing words in a sentence, producing text in response to a prompt, or translating across languages. The model can be fine-tuned for certain NLP tasks like text classification, summarization, or question answering after it has been pre-trained. Using a single model to complete different NLP tasks is one of T5's primary advantages. Multitask learning is a method that accomplishes this by simultaneously training the model on several NLP tasks. In learning shared representations across various tasks, this enables the model to perform better on each given task.

The T5 model's aptitude to perform sequence-to-sequence tasks and acquire intricate linguistic patterns makes it well-suited for grammatical error correction. Correction of grammatical errors entails taking an input sentence that contains errors and producing a corrected sentence that is grammatically sound. It is a sequence-to-sequence task, one of the T5 model's key strengths. The T5 model is ideally suited for grammatical error correction since it is built to accept a sequence of input tokens and produce a sequence of output tokens. Furthermore, the T5 model is a potent transformer model that has already been pre-trained on a sizable corpus of text, enabling it to learn intricate linguistic patterns. This pre-training gives the model the ability to comprehend the rules of grammar and syntax, enabling it to spot mistakes in phrases that do not follow them.

The T5 model's multitask learning capabilities can also be used to enhance grammatical error correction performance. The T5 model may learn to extract higher-level characteristics that are helpful for grammar error correction by being trained on numerous related tasks, such as translation or text summarization.

In conclusion, the T5 transformer model is a potent neural network model used for problems involving natural language processing. It is educated via pretraining, has several encoder and decoder layers, and can complete numerous NLP tasks with one model.

## 5.12 Evaluation Metrics

Model evaluation metrics are commonly used to evaluate the performance of Natural Language Processing (NLP) models. These metrics allow us to measure the performance of the model on text-to-text conversion task. These metrics help us to know whether the model performs as per the requirements or not and measure the standard of the model.

### 5.12.1 N-gram

An ‘n-gram’ is a widely used concept from regular text processing and is not specific to NLP. It is just a fancy way of describing the set of consecutive words in the sentence.

For the sentence: “This is the text”

1-gram (unigram): “This”, “is”, “the”, “text”

2-gram (bigram): “This is”, “is the”, “the text”.

3-gram (trigram): “The is the”, “is the text”

4-gram: “This is the text”

### 5.12.2 BLEU Score

BLEU (Bilingual Evaluation Understudy) measures the precision of text which is machine translated from one natural language to another. The concept behind BLEU is that "the closer a machine-driven output is to a human translation, the better the result is." Quality is referred to as the relationship between a machine's output and that of a human. BLEU was one of the first metrics to have high correlation with human judgement and is one of the most used computational metrics. The calculated BLEU score is in the range of 0 to 1 and higher the calculated BLEU scores the more the model output is closer to that of the reference sentence which exhibits a better performing model.

The formula for calculating the BLEU score is:

$$BLEU = BP.\exp\left(\sum_{n=1}^{N} w_n log(p_n)\right) \tag{5-9}$$

Where:

- N is the maximum n-gram order to consider (typically 4)
- $w_n$ is the weight assigned to the n-gram precision score where $\sum_{n=1}^{N} w_n = 1$
- $p_n$ is the n-gram precision score
- $BP$ is the brevity penalty term, which adjusts the score to account for translations that are shorter than the reference translations
- exp () is the exponential function

The n-gram precision score $p_n$ is calculated as follows:

$$p_n = \frac{\sum_{n-grams \in output} min\left(count_{ref}(n - grams), count_{output}(n - grams)\right)}{\sum_{n-grams \in output} count_{output}(n - grams)}$$

[5-10]

Where:

- $output$ is the set of n-grams in the machine translation output
- $count_{ref}(n - grams)$ is the maximum count of the n-gram in any reference translation
- $count_{output}(n - grams)$ is the count of the n-gram in the machine translation output

The brevity penalty term $BP$ is calculated as follows:

$$BP = \begin{cases} 1 & if\ c > r \\ \exp\left(1 - \frac{r}{c}\right) & if\ c \leq r \end{cases}$$ [5-11]

Where:

- $c$ is the length of the machine translation output in words
- $r$ is the length of the shortest reference translation in words

The BLEU score measures the overlap between the n-grams in the machine translation output and the n-grams in the reference translations, with higher scores indicating better translations. The brevity penalty term adjusts the score to account for translations that are too short.

### 5.12.3 ROUGE Score

ROUGE score also called (Recall -Oriented Understudy for Gisting Evaluation) is a set of metrics that can be used for text-to-text model. It evaluates the quality of the generated text by comparing it to one or more reference texts.

There are three sub metrics used to calculate ROUGE score which are:

- **Recall:** The recall counts the number of overlapping n-grams found in the model output and reference then decides the number by the total number of n grams in the reference. The formula to calculate the recall is:

$$Recall = \frac{(number\ of\ n - grams\ found\ in\ candidate\ and\ reference)}{(number\ of\ n - grams\ found\ in\ reference)}$$

[5-12]

- **Precision:** Precision is like recall but rather than dividing by the number of reference n-gram count we divide by the model output n-gram count. The formula to calculate precision is:

$$Precision = \frac{(number\ of\ n - grams\ found\ in\ candidate\ and\ reference)}{(number\ of\ n - grams\ found\ in\ candidate)}$$

[5-13]

- **F1-score:** The F1-score is calculated as

$$F1 - score = 2 * \frac{precision * recall}{precision + recall}$$ [5-14]

And ROUGE score is not a single metrics and there are multiple ones:

#### 5.12.3.1 ROUGE N

ROUGE-N measure the number of matchings ‘n-grams’ between the output generated by our model to the target or reference. This measures the score by applying the n gram to the precision recall and f1-score.

We specifically used ROUGE-1 and ROUGE-2 here which is for unigrams and diagrams.

For ROUGE 1:

$$Recall = \frac{(number\ of\ 1 - gram\ found\ in\ candidate\ and\ reference)}{(number\ of\ 1 - gram\ found\ in\ reference)}$$

[5-15]

$$Precision = \frac{(number\ of\ 1 - gram\ found\ in\ candidate\ and\ reference)}{(number\ of\ 1 - gram\ found\ in\ candidate)}$$

[5-16]

$$F1 - score = 2 * \frac{precision * recall}{precision + recall}$$ [5-17]

For ROUGE 2:

$$Recall = \frac{(number\ of\ 2 - grams\ found\ in\ candidate\ and\ reference)}{(number\ of\ 2 - grams\ found\ in\ reference)}$$

[5-18]

$$Precision = \frac{(number\ of\ 2 - gram\ found\ in\ candidate\ and\ reference)}{(number\ of\ 2 - gram\ found\ in\ candidate)}$$

[5-19]

$$F1 - score = 2 * \frac{precision * recall}{precision + recall}$$ [5-20]

#### 5.12.3.2 ROUGE L

Rouge-L measure the longest common sequence between the model output and the target sentence or reference. So, we count the sequence of tokens shared between both.

The basic idea with this metric is that the longer sequence that is matching between the model output and reference show more similarity between the generated text and the text that has to be generated.

In ROUGE-L we replace the number of n-grams found in the model output and reference text to the Longest Common Sequence (LCS) between the model output and reference.

$$Recall = \frac{LCS}{Count\ of\ reference\ text} \qquad [5\text{-}21]$$

$$Precision = \frac{LCS}{Count\ of\ candidate\ text} \qquad [5\text{-}22]$$

$$F1 - score = 2 * \frac{precision * recall}{precision + recall} \qquad [5\text{-}23]$$

In general, BLEU focuses on how much n-grams/ words in the model outputs appear in the reference/target. ROUGE focuses on how much n-grams in the reference/target appear in the candidate model output.

## 5.13 System Overview

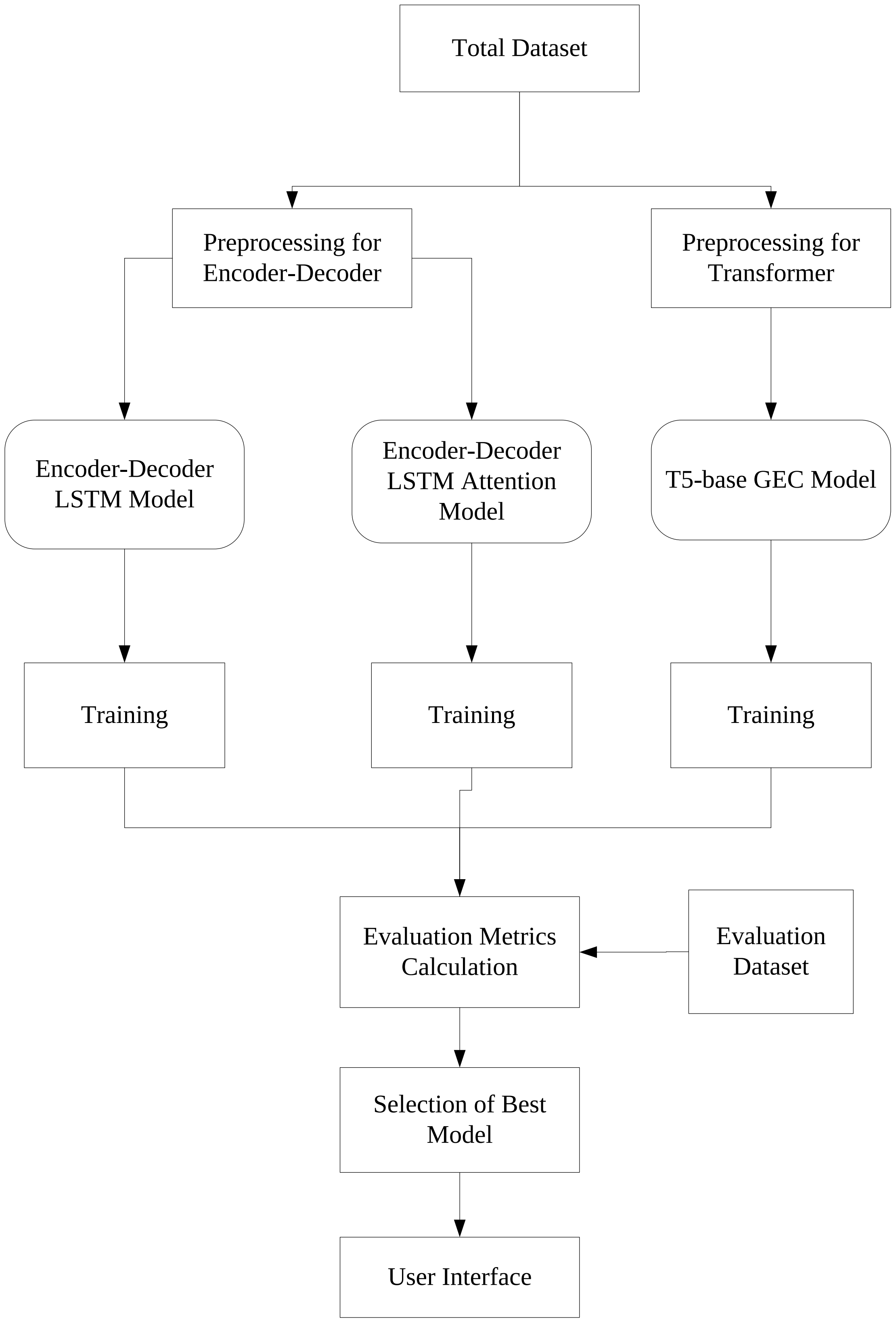

Total Dataset
Preprocessing for Encoder-Decoder
Preprocessing for Transformer
Encoder-Decoder LSTM Model
Encoder-Decoder LSTM Attention Model
T5-base GEC Model
Training
Training
Training
Evaluation Metrics Calculation
Evaluation Dataset
Selection of Best Model
User Interface

Figure 5-11 : Basic System Overview

The whole system is based on three different grammar error correction model, which are encoder-decoder LSTM model, encoder-decoder LSTM model with attention, and T5 transformer model. It is made sure that all models are trained on same dataset and evaluated on same evaluation dataset.

The data preprocessing step is same for encoder-decoder model and attention encoder-decoder model. But it is different for transformer model. Then these models are separately trained on a large training dataset which the most time-consuming part of the project. After the model is trained and ready to evaluate. The models are evaluated under different evaluation metrics. And by comparison the best model is selected to be deployed in the web application.

The web application is based on Django, an open-source python library for web development.

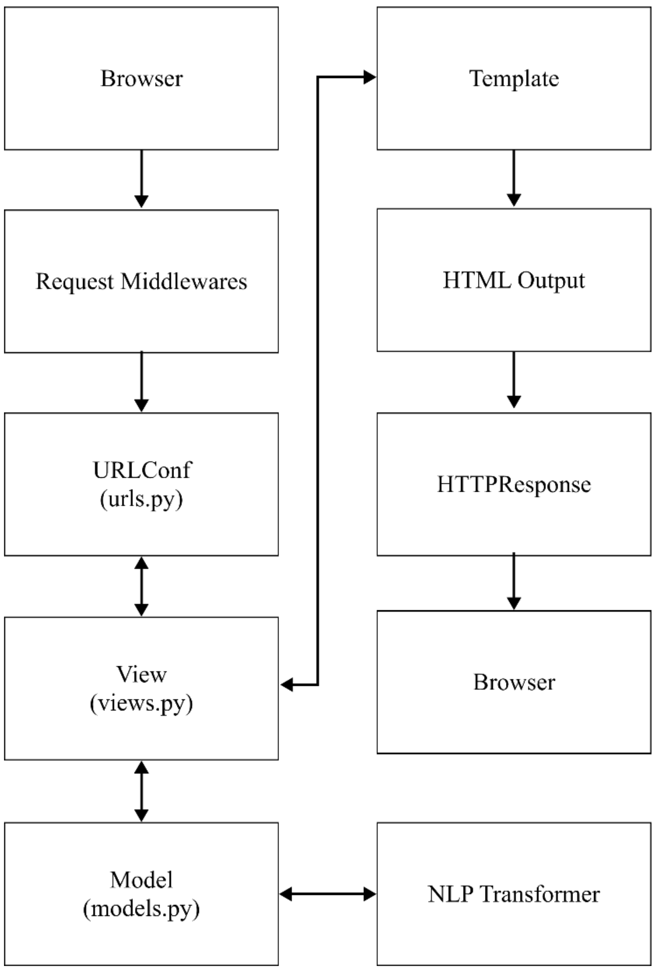


Figure 5-12 : Block Diagram of Website Implementation

Here, users request a page either through URL or a link. Then, proposal is granted Middleware for handling requests that could alter or respond to them. The request is intercepted by Django's request middleware layer, which then executes any specified request middleware functions. These features could alter the request or give a direct answer.

The ‘URLConf’ locates the associated view using urls.py. Django URL resolver searches up the proper view function and the ‘urlpatterns’ list specified in the urls.py file is used to map URLs to views.

The view function is invoked, and the request object is passed as an argument. The view function deals with the request, gaining access to information via models. If necessary, views could use unique context, it can create context dictionary, add data to it and pass to the template.

The Template receives the information before rendering it. The view function renders a template after handling the request and gathering any required data. The template receives the context dictionary from the view function and uses it to create the HTML.

The view function generates an HTTP response object using the HTML output. The response middleware layer catches the answer and executes any configured response middleware operations. The user's browser receives the completed response object, which is then displayed as a web page.

# 6. IMPLEMENTATION DETAILS

To implement a grammar error correction system using encoder-decoder LSTM neural network a free and open-source software library called TensorFlow was used. The various steps involved in the implementation of grammar error correction are listed below:

- Data Preprocessing
- Tokenization
- Encoder
- Decoder
- Model Compilation and Training

## 6.1 Data preprocessing

Data preprocessing is the initial stage in preparing the training and test sets of data that will be fed into the model to see if it will produce the desired results. Preprocessing data is done to enhance its quality to produce better results. The processes of cleaning, transformation, and reduction are involved. The handling of noisy and missing data. Data is changed into the proper forms, such as scaling or normalizing, after it has been cleansed. The study could have been more difficult because the dataset contains a big quantity of data. In order to boost efficiency by drastically lowering the cost of model data storage, the data size is reduced using the data reduction technique.

## 6.2 Tokenization

Tokenization is a technique that divides up text into individual words before mapping them to a numerical representation in vector form. The tokenization process can be carried out using a variety of word embedding techniques.

To tokenize text into numerical representations, utilize the '*Tokenizer*' class. Once each word has been given a distinct index, a list of text is inputted by calling the *'fit_on_texts'* method. The text as input is converted the text into token indices. As this method provides a 2D array, each row index represents a token, and each row represents a sentence.

## 6.3 Encoder

The encoder is a component of the neural network that uses one or more LSTM layers to encode the input sequences into a fixed-length vector representation. Sequence-to-sequence modeling is used for encoding. This model takes the sequence of integers with padding as input, transforms the sequence into dense vectors, applies an LSTM layer to the sequence, and outputs the context of the input sequence.

- **Input layer:** This layer instantiates the encoder input. It defines the shape of input to the encoder. It accepts a 2D array tensor.
- **Embedding layer:** This layer instantiates the encoder embedding This layer converts each word into a fixed length vector of defined size. This layer defines the size of the input vocabulary and the dimension of embedding.
- **LSTM layer:** 'LSTM' layer instantiates the encoder LSTM. Here, the parameters show the hidden units, and the final hidden state of LSTM layer is returned along the output. The layer returns the output and the final hidden state for each input in the batch.
- **Outputs and states:** The LSTM layer output three thing which are the hidden states generated by the LSTM layer, hidden and cell states which capture the context of the input sequence.
- **Encoder states:** This is a list of the hidden and cell states, which will be used as the initial state of the decoder in the sequence-to-sequence model.

## 6.4 Decoder

The portion of the neural network known as the decoder is responsible for translating the encoded input representation into the corrected output sequence. Like encoders, it also comprises of several LSTM layers. There are five layers in the decoder model:

- **Input layer:** This is an input layer for the decoder connected to an encoder state of encoder model.
- **Embedding layer:** When decoder model receives an input, it maps the integers to dense vector of fixed size.
- **LSTM layer:** The layer will return the output and the final hidden state for each input in the batch. The output of the LSTM layer for each time step will be returned and used as the input to the next time step in the sequence.

- **Decoder output:** The returned decoder outputs represent the output of the LSTM layer at each time step in the sequence and the final states are discarded.
- **Dense layer:** The Dense layer contains output vocabulary size which is the integer value of the number of neurons in this layer and SoftMax activation function.

The network formed by using different LSTM layers is trained using the training data which is prepared by performing necessary preprocessing of data. The optimized loss function is used to measure the difference between the predicted output and actual output.

## 6.5 Compiled model

### 6.5.1 Encoder-Decoder LSTM model

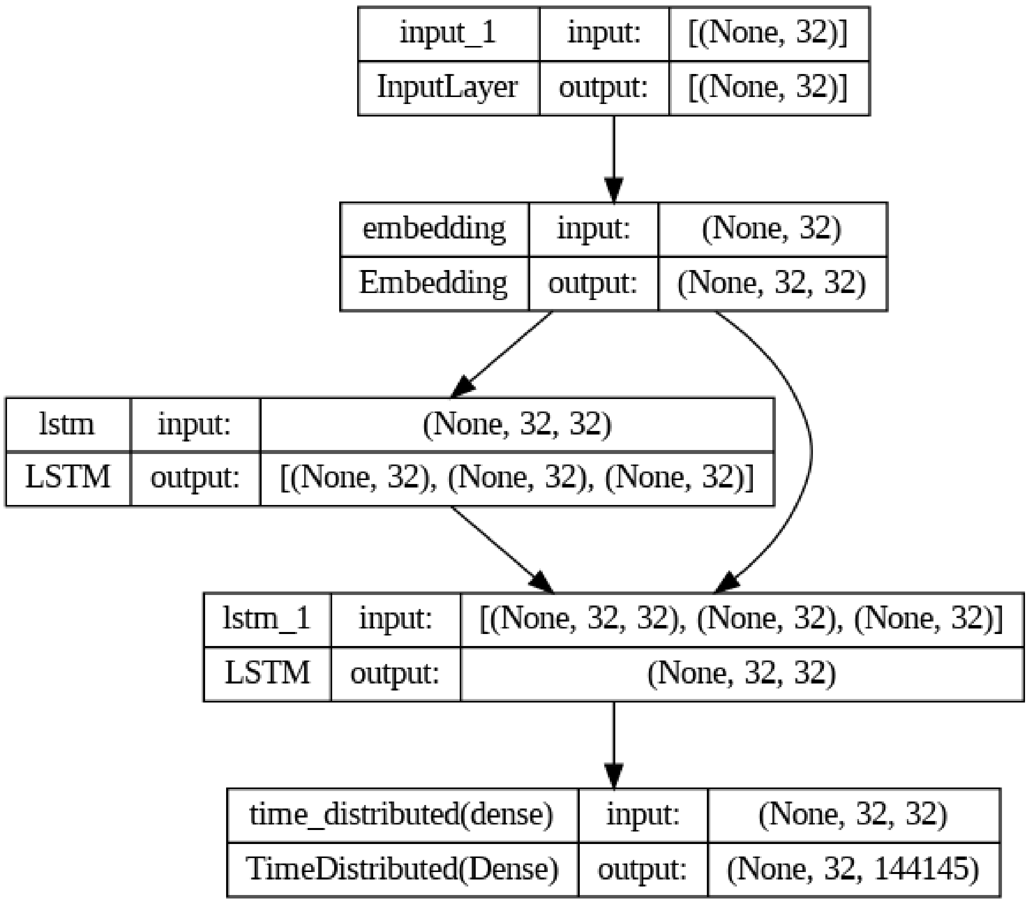

input_1 | input: | [(None, 32)]
InputLayer | output: | [(None, 32)]
embedding | input: | (None, 32)
Embedding | output: | (None, 32, 32)
lstm | input: | (None, 32, 32)
LSTM | output: | [(None, 32), (None, 32), (None, 32)]
lstm_1 | input: | [(None, 32, 32), (None, 32), (None, 32)]
LSTM | output: | (None, 32, 32)
time_distributed(dense) | input: | (None, 32, 32)
TimeDistributed(Dense) | output: | (None, 32, 144145)

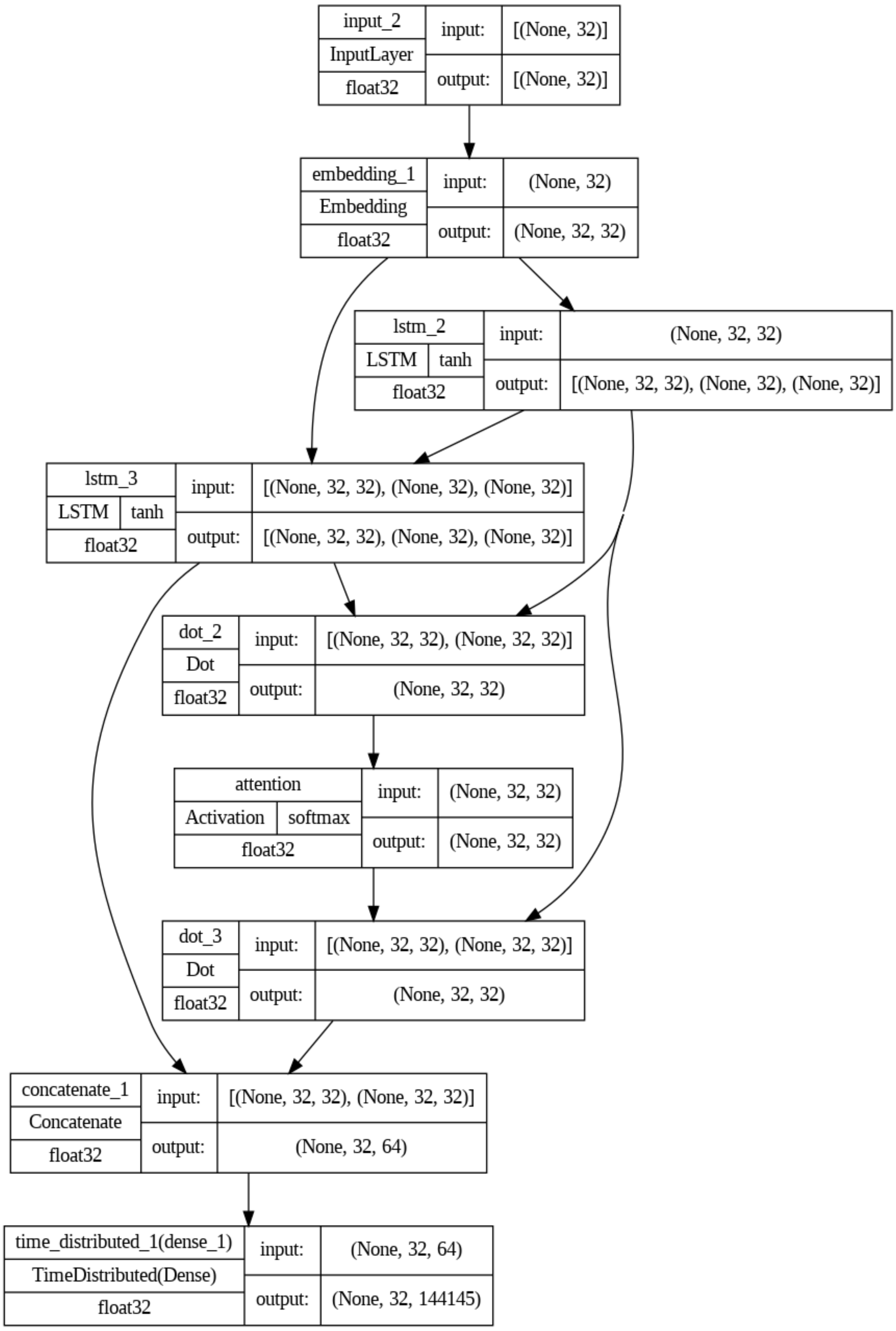
Figure 6-1 : Encoder-Decoder LSTM Model plot

The encoder consists of an input layer, which means that it takes in a sequence of tokenized items. The input is then passed through an embedding layer with 32 as embedding dimensions and vocabulary size, which would vary based on the dataset used, which is 144145 for this model when using only certain portion of Lang-8 English corpus. The output of the embedding layer is then passed through an LSTM layer with 32 units, which outputs the encoder's final hidden state and cell state.

The decoder also consists of an LSTM layer with 32 units, but in this case the layer is set to return sequences and it receives the embedded input sequence as well as the encoder's final hidden and cell states as its initial state. The output of the decoder is then passed through a time-distributed dense layer with the size of vocabulary size units and SoftMax activation, which outputs a probability distribution over the vocabulary for each time step in the decoder's output sequence.

The model is compiled using the Adam optimization algorithm, and sparse categorical loss function. This model was also configured to evaluate its accuracy during training.

## 6.5.2 Encoder-Decoder LSTM Model with Attention Mechanism



input_2 | InputLayer | float32 | input: [(None, 32)] | output: [(None, 32)]
embedding_1 | Embedding | float32 | input: (None, 32) | output: (None, 32, 32)
lstm_2 | LSTM | tanh | float32 | input: (None, 32, 32) | output: [(None, 32, 32), (None, 32), (None, 32)]
lstm_3 | LSTM | tanh | float32 | input: [(None, 32, 32), (None, 32), (None, 32)] | output: [(None, 32, 32), (None, 32), (None, 32)]
dot_2 | Dot | float32 | input: [(None, 32, 32), (None, 32, 32)] | output: (None, 32, 32)
attention | Activation | softmax | float32 | input: (None, 32, 32) | output: (None, 32, 32)
dot_3 | Dot | float32 | input: [(None, 32, 32), (None, 32, 32)] | output: (None, 32, 32)
concatenate_1 | Concatenate | float32 | input: [(None, 32, 32), (None, 32, 32)] | output: (None, 32, 64)
time_distributed_1(dense_1) | TimeDistributed(Dense) | float32 | input: (None, 32, 64) | output: (None, 32, 144145)

Figure 6-2 : Encoder Decoder Attention Mechanism

This model has an encoder with an input layer, an embedding layer and an LSTM layer that outputs the final hidden and cell states. The decoder has an LSTM layer that takes in an encoded input sequence and encoder’s final state, and outputs a sequence. Additionally, it includes a Dot product Attention between the decoder and encoder output sequence passes through a SoftMax activation function to produce weight distribution. A context vector is computed using the weights and concatenated with the decoder output sequence and passes through dense layer with SoftMax activation function to produce final output.

The implementation includes a data preprocessing pipeline for sequence-to-sequence model. The steps involved are reading the CSV data and removing the null and duplicate values and tokenizing the input and target sentences together creating a vocabulary along with the start and end tokens and converting the text sequences into fixed length of 32. Then the data is split into test and train sets.

The implementation consists of creating LSTM base encoder and decoder models for sequence-to-sequence task. The encoder model takes input tokens and converts them to dense vector representations and produces an encoded representation of the sequence. The decoder model takes the encoded representation of final hidden and cell states of the encoder as the inputs and produces a sequence of output tokens.

This model uses an attention mechanism for the sequence-to-sequence model which allows the decoder to focus on different parts of the input sequence during the decoding process. The attention mechanism computes attention weights by taking the dot product of the decoder output sequence and encoder output sequence. Then the SoftMax activation function is applied to create a probability distribution over the input sequence, indicating which part should receive more attention during decoding.

It computes the context vector by taking the dot product of the attention weights and the encoder output sequence. This weighted sum of the encoder outputs gives the decoder information about which parts of the input sequence are most relevant for generating current output token. Then, it concatenates the context vector and the decoder output sequence to create a combined representation that contains information about both the input sequence and the output generated so far. This combined representation is then used as input to the final output layer of the model to generate the next output token. Overall, this implements an attention mechanism that allows the decoder to focus on different parts of the input sequence during decoding, based on the relevance of each part to the current decoding step.

The final layer of the sequence-to-sequence model which is a dense layer predicts the next token in the output sequence based on the input and decoder output sequences. The model is compiled with Adam optimizer, sparse categorical Cross entropy loss function.

The final layer of the sequence-to-sequence model, which is a dense layer that predicts the next token in the output sequence based on the combined representation of the input sequence and the decoder output sequence.

### 6.5.3 T5 Transformer model

At first the lang-8 dataset and subset of JFLEG-CONLL-2014 dataset is loaded into the session as a pandas Data Frames in string format. The lang-8 dataset is loaded as to train the t5 mod Initially the datasets are converted into Dataset format from the ‘dataset’ library. This produces a training dataset and a testing dataset which is used to train and test the model.

In this method, a pretrained language model T5-base which is the base version of T5 transformer was utilized. The initial model was obtained by downloading it from the HuggingFace library. Additionally, the original tokenizer for the T5-base model was also obtained from the same library.

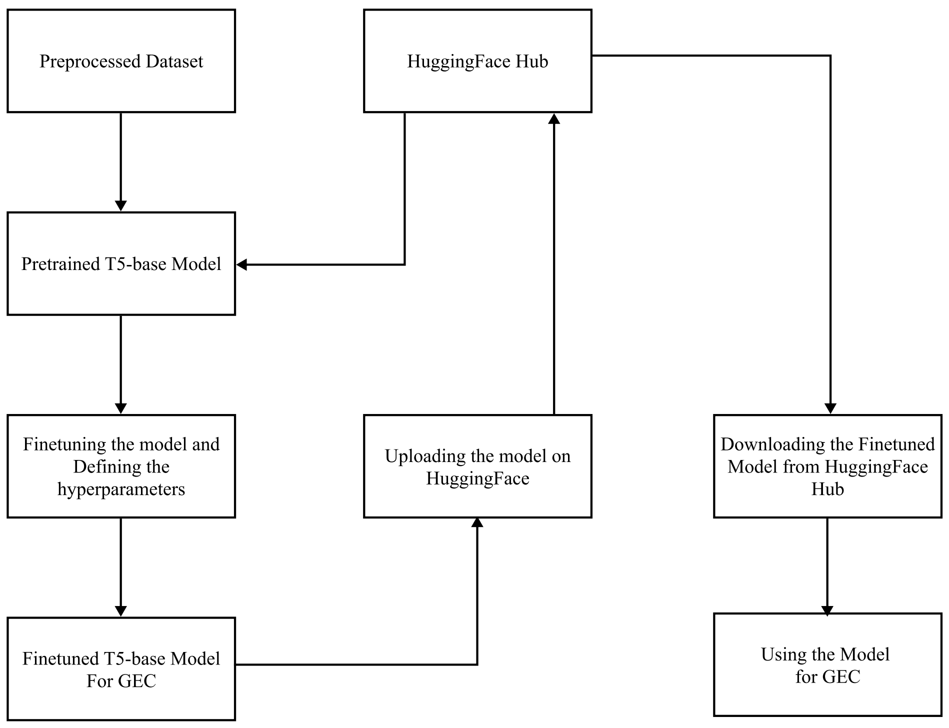

Preprocessed Dataset
HuggingFace Hub
Pretrained T5-base Model
Finetuning the model and Defining the hyperparameters
Uploading the model on HuggingFace
Downloading the Finetuned Model from HuggingFace Hub
Finetuned T5-base Model For GEC
Using the Model for GEC

Figure 6-3 : Block Diagram of T5 Overview

The several built in parameters and their values are:

- **'evaluation_strategy':** This parameter is used to determine the frequency evaluation of the data during training on the evaluation dataset. The value is set to steps.
- **'per_device_train_batch_size':** This parameter indicates how many instances should be handled on each GPU during training. The batch size per GPU was set as 16. The batch size was set to 16 because a larger batch size required more computational power and the GPU was out of memory quicker. Although larger batch size decreases the training time by a large margin smaller batch size was used due to GPU memory limitation.
- **'per_device_eval_batch_size':** This parameter indicates how many instances should be evaluated on each GPU during evaluation. Like the train batch size, the evaluation batch size was also 16.
- **'learning_rate':** The learning rate value of '2e-5', was used which is a typical default number used in many natural language processing applications

- **‘num_train_epoch’:** This argument defines how many times the full training dataset should be iterated through. The number of epochs used are 9 epochs.
- **‘weight_decay’:** This parameter regulates how much L2 regularization is applied to the model during training. NLP models having large number of parameters and tend to overfit easily, so weight decay is used. The weight decay value of 0.01 was used.
- **‘fp16’:** A technique called half precision was used which represents the weights and gradients of the model in fewer bit significantly improving the training speed and memory usage. This was used due to the GPU limitation and large number of training data for efficient training of the model.
- **‘eval_step’:** This parameter specifies the number of steps of the evaluation strategy on which the model which is training must be evaluated.

After setting these hyperparameter and declaring them in ‘Seq2SeqTrainer()’ along with the tokenizer, training dataset, evaluation dataset. The T5 model is trained in the lang-8 dataset for around 9 epochs which takes some considerable time to be trained due to large size of dataset. And finally, the model is then saved and deployed.

## 6.6 Grammar Error Correction Using the Model

### 6.6.1 Encoder Decoder Model Correction

For both the encoder decoder model and encoder decoder model using attention mechanism, the calculation was done after the incorrect sentence, or the input sentence was sent to the model first. Before sending the input sentence for evaluation the input sentence was concatenated with the start and end tokens. Then the sequence was converted into tokenized forms which is then used by the model to predict the correct output tokens then the output is converted into word format removing all the unnecessary tokens present in the sentence.

### 6.6.2 T5 Model Transformer Correction

After compiling T5-base model the model was uploaded in the hugging face hub and then again downloaded on the required device for correcting the sentences. After downloading the model, by using the generate function and supplying the model with input sentence we generated the correct sentence.

# 7. RESULTS AND ANALYSIS

In the initial phase of the project, there were limitation of resources, we decided to train the encoder-decoder models in small subset of lang-8 dataset which contained around 70000 datapoints. And at the time, we were using Google Collab to train the model. If we used the whole dataset which had around 500000 datapoints the training parameters exceeded the capacity of Collab's GPU. Hence, we trained the two models, encoder-decoder LSTM model and encoder-decoder attention LSTM model in that small dataset. The models were trained for 30 epochs and following results were obtained:

## 7.1 Small Dataset

### 7.1.1 Encoder-Decoder Model

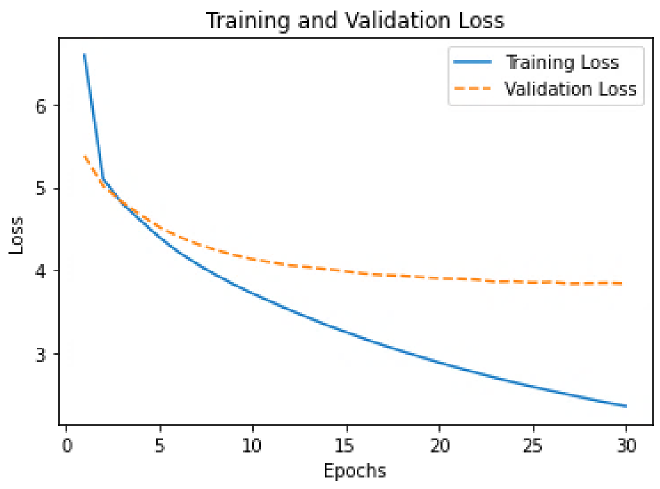


Figure 7-1 : Training and Validation Loss vs Epochs for Encoder-Decoder

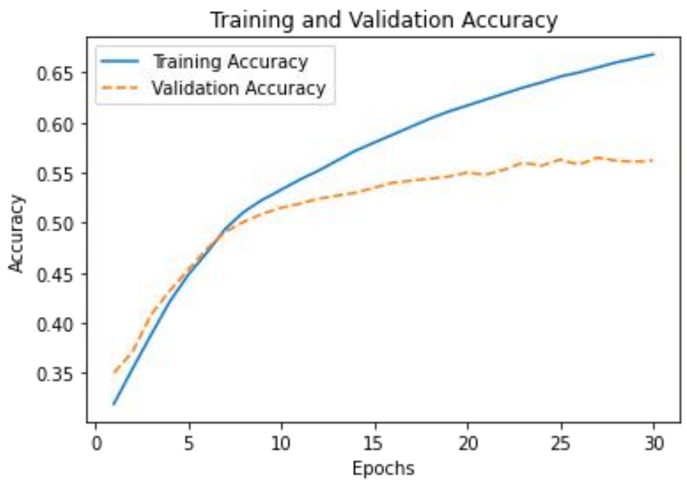


Figure 7-2 : Training and Validation Accuracy vs Epochs for Encoder-Decoder

From the above figure, we can see that after around 6-7 epochs into training, the model starts to get overfitted for the training dataset, in both the case of loss and accuracy. We can infer this result from the plot of loss and accuracy for the validation data. After a mentioned point it does not follow the plot of training data, the plots for training data overshoots and plots for the validation starts saturating.

### 7.1.2 Encoder-Decoder Attention Model

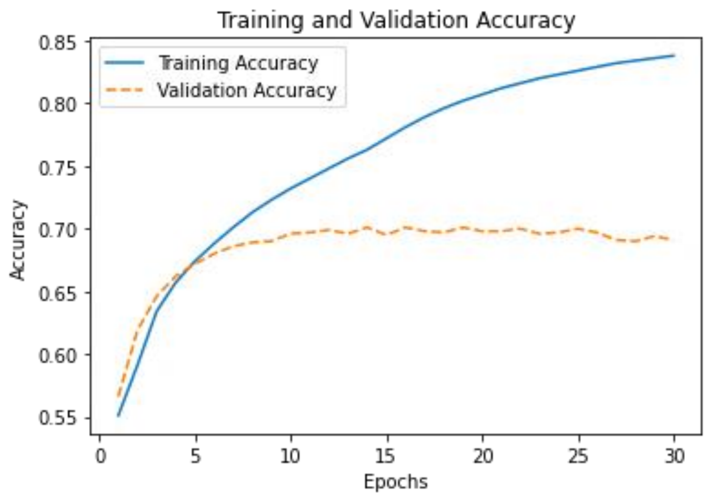


Figure 7-3 : Training and Validation Accuracy vs Epochs for Attention Model

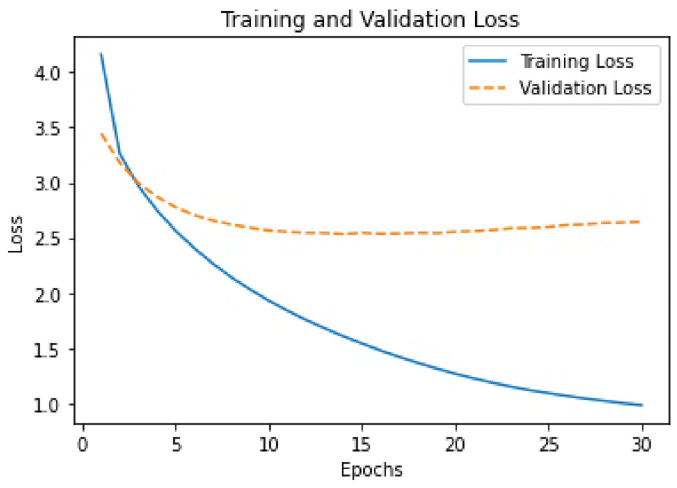


Figure 7-4 : Training and Validation Loss vs Epochs for Attention Model

The discrepancy seen in the above encoder-decoder model can also be seen in this model. And since attention mechanism is implemented, the overfitting starts early in this model, at around 3-4 epochs.

After this we found out that the model is overfitting due to smaller size of datapoints and larger number of training epochs. Along with that, the models were performing very poorly for new data. Hence, we didn't perform evaluation for these models, and we started to search for methods to use whole dataset for training.

### 7.2 Whole Dataset

Then in the later phases of the project, as we learned about the natural language processing models. We found out that when training the language models in small dataset for many epochs the model gets overfitted. So, we decided to train the whole dataset for limited number epochs and compare those models. And during this time, we were advised to use Kaggle's service to train models, which was just enough to train our model in whole dataset.

We trained three different models in the Kaggle's GPUs: Encoder-Decoder LSTM model, Encoder-Decoder Attention LSTM model and T5 Transformer model. And following results were obtained:

#### 7.2.1 Encoder-Decoder Model

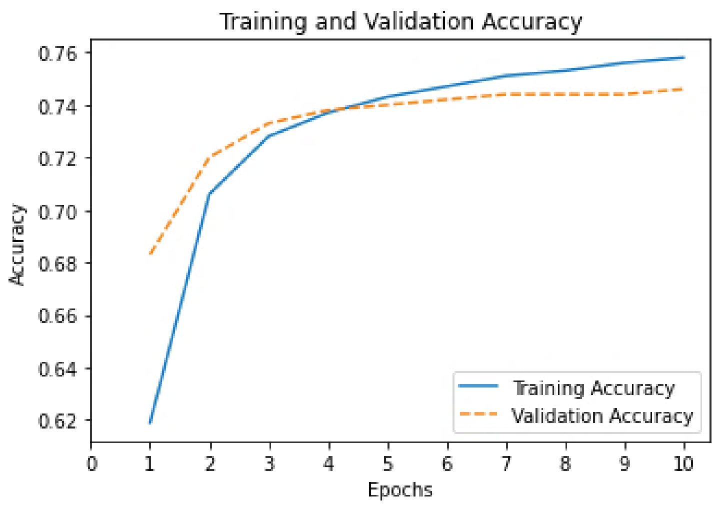


Figure 7-5 : Accuracy vs 10 Epochs for Encoder-Decoder

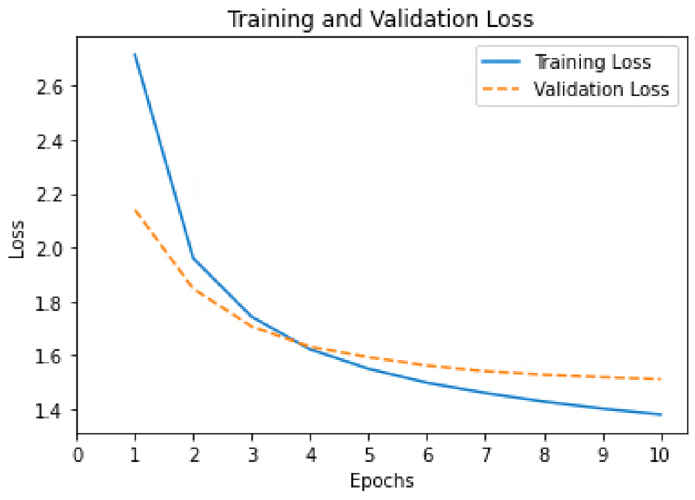


Figure 7-6 : Training and Validation Loss vs Epochs for Encoder-Decoder

Here we can see that for small number of epochs and large data, the validation loss and accuracy plot is closely following the training loss and accuracy plot, which the desired result for a model.

### 7.2.2 Encoder-Decoder Attention Model

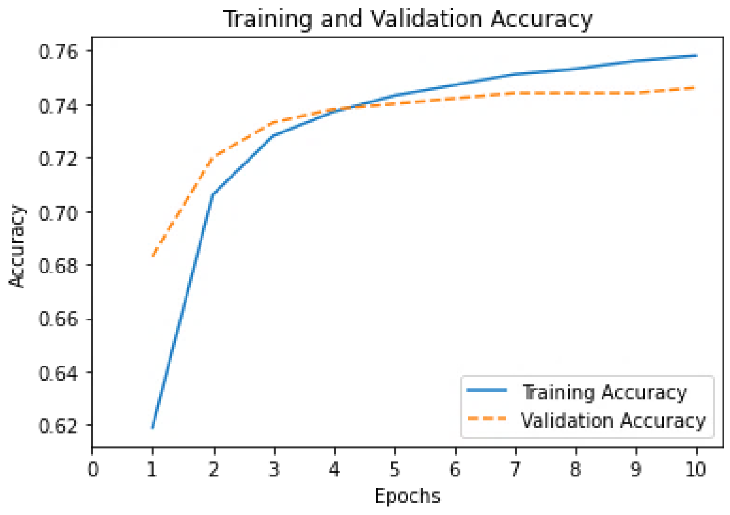


Figure 7-7 : Accuracy vs 10 Epochs for Attention Model

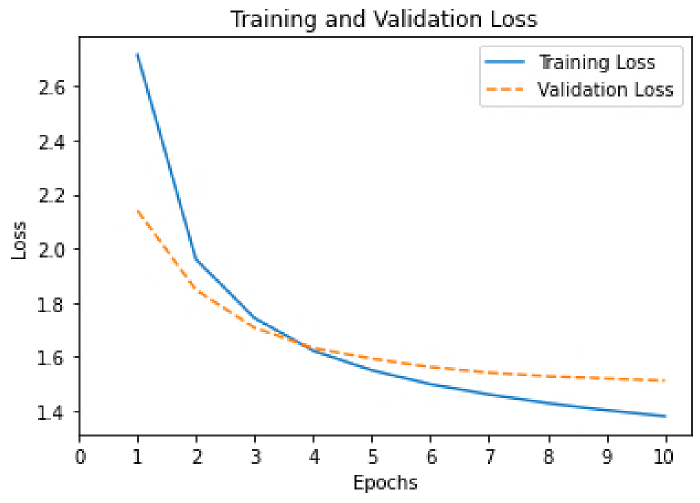


Figure 7-8 : Loss vs 10 Epochs for Attention Model

And in this model, since attention is applied, the accuracy and loss plot are changing faster than that of model with no attention. And the problems like overfitting or underfitting is not seen.

### 7.2.3 T5 Transformer Model

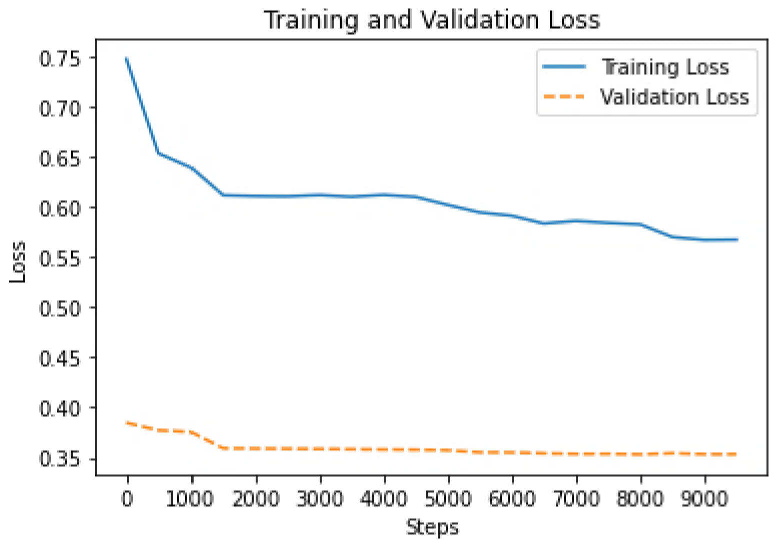


Figure 7-9 : Loss Vs Steps for Transformer Model

Since the T5 model is a transfer model, the model is already pretrained in multiple text to text datasets, so we can see that the starting loss is very low 0.75, and after training the t5-base model for around 9 epochs the loss decreases to 0.55. And we can see that validation loss is very low, which is because the validation dataset may be matching with the dataset used to train the t5-base model, thus is performing very well in validation dataset.

## 7.3 Evaluation Metrics for GEC models

Table 7-1 : Evaluation for BLEU Scores

| **Model Name** | **BLEU-1** | **BLEU-2** | **BLEU - 3** | **BLEU-4** |
|---|---|---|---|---|
| Encoder Decoder LSTM Model | 63.05 | 51.8 | 43.75 | 36.1 |
| Encoder Decoder Attention LSTM Model | 62.68 | 52.04 | 44.37 | 36.91 |
| T5-base GEC Model | 71.69 | 65.98 | 60.98 | 55.40 |

Table 7-2 : Evaluation for ROGUE Scores

| **Model Name** | **ROUGE1** | **ROUGE2** | **ROUGEL** |
|---|---|---|---|
| Encoder Decoder LSTM Model | 69.84 | 50.1 | 68.31 |
| Encoder Decoder Attention LSTM Model | 69.62 | 50.15 | 68.04 |
| T5-base GEC Model | 90.18 | 81.87 | 89.72 |

As from the above table made after the evaluation metrics, an evaluation dataset was created, and evaluation was performed for the model, and we can observe various things from the result above.

All the metrics are normalized to the range of 0 to 100 for clear results from the evaluation metrics.

The higher the score the better the model we can see that Encoder Decoder Attention LSTM Model is slightly better than Encoder Decoder LSTM Model. There is not much improvement after applying the attention layer. But after applying the evaluation on trained T5-base GEC Model we can see a huge improvement. So T5-base model was found to be the best trained model. And T5 model also performed well when encountering new data.

## 7.4 User Interface and Output

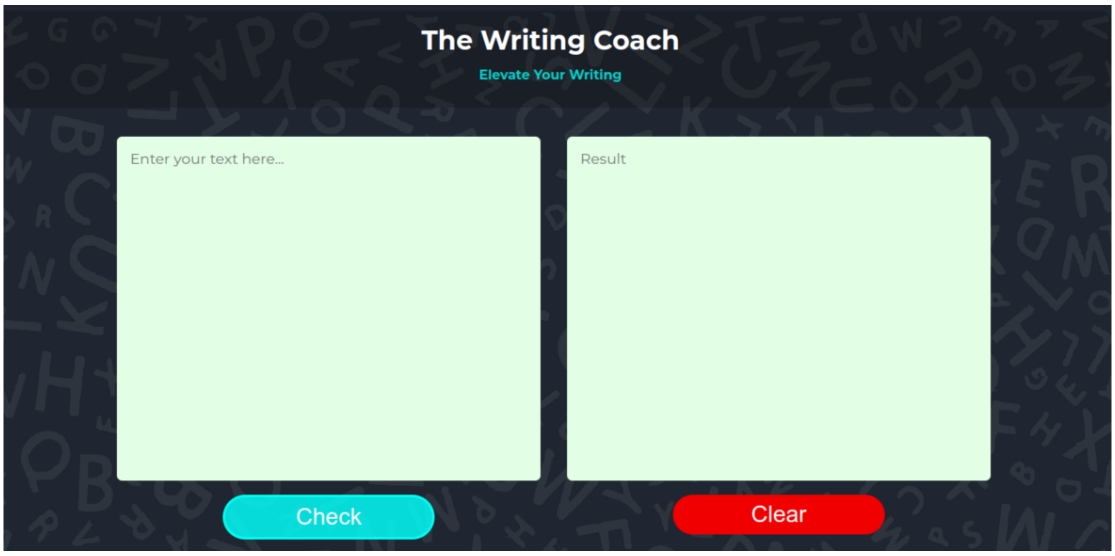
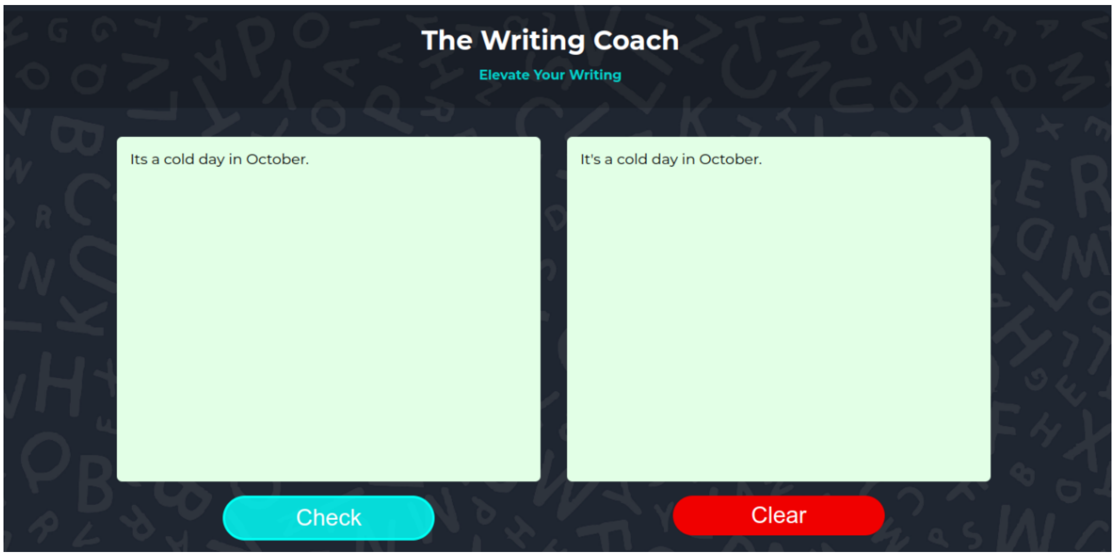


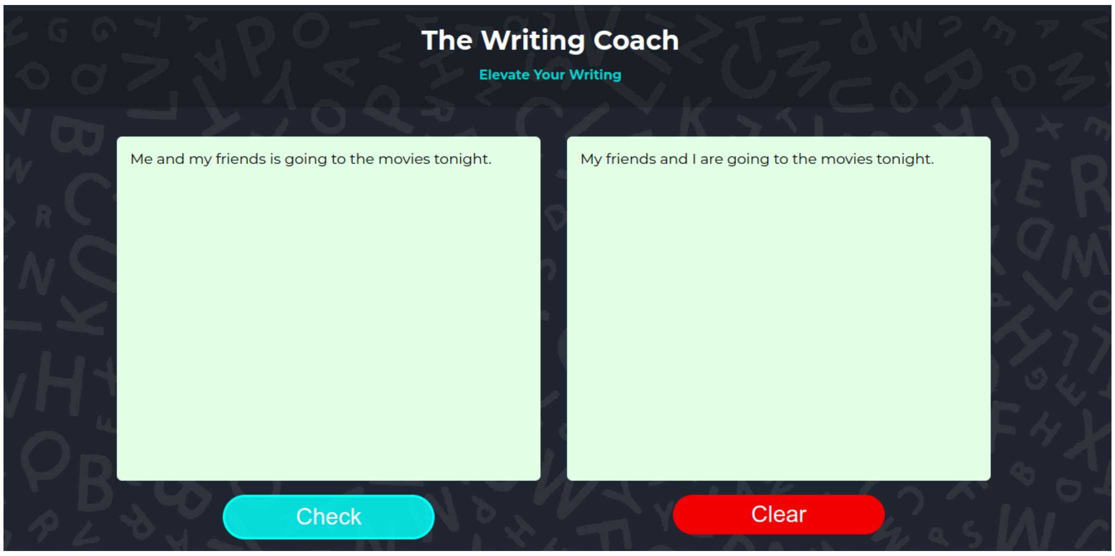
Figure 7-10 : Homepage of The Writing Coach



Its a cold day in October.

It's a cold day in October.

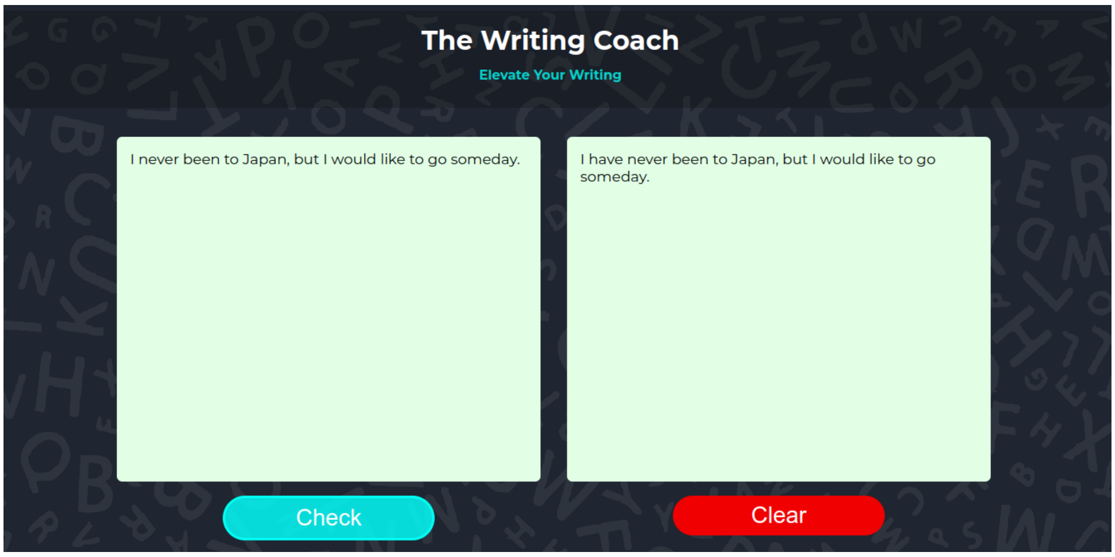
Figure 7-11 : Output I – Sentence



Me and my friends is going to the movies tonight.

My friends and I are going to the movies tonight.

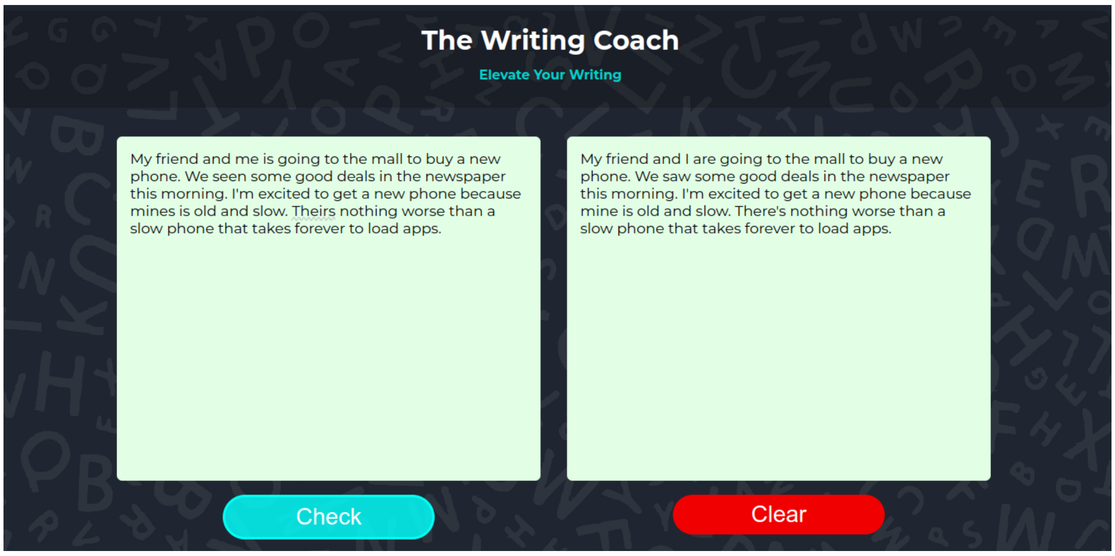
Figure 7-12 : Output II – Sentence



I never been to Japan, but I would like to go someday.

I have never been to Japan, but I would like to go someday.

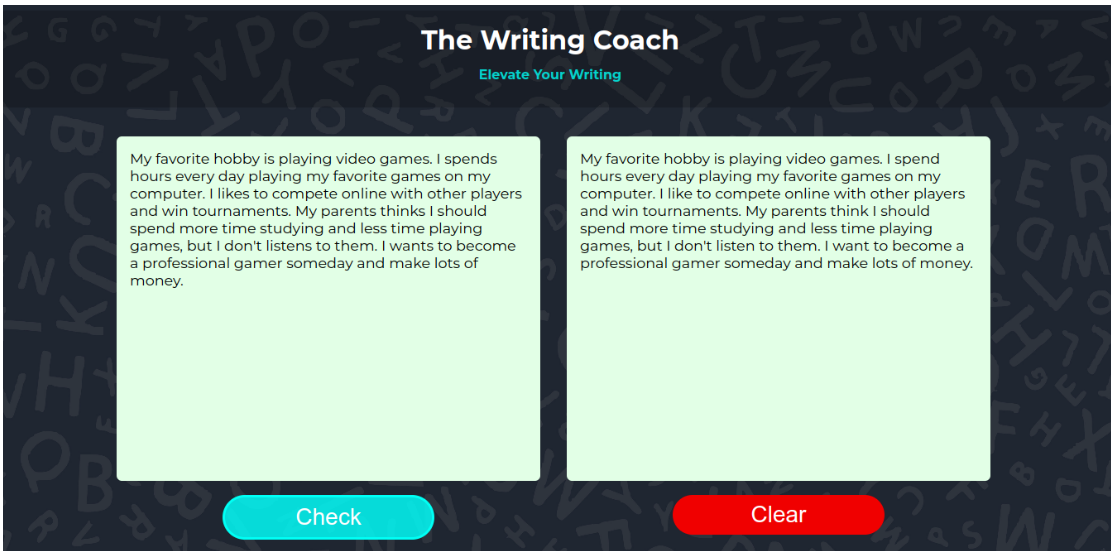
Figure 7-13 : Output III – Sentence



My friend and me is going to the mall to buy a new phone. We seen some good deals in the newspaper this morning. I'm excited to get a new phone because mines is old and slow. Theirs nothing worse than a slow phone that takes forever to load apps.

My friend and I are going to the mall to buy a new phone. We saw some good deals in the newspaper this morning. I'm excited to get a new phone because mine is old and slow. There's nothing worse than a slow phone that takes forever to load apps.

Figure 7-14 : Output IV – Passage



My favorite hobby is playing video games. I spends hours every day playing my favorite games on my computer. I likes to compete online with other players and win tournaments. My parents thinks I should spend more time studying and less time playing games, but I don't listens to them. I wants to become a professional gamer someday and make lots of money.

My favorite hobby is playing video games. I spend hours every day playing my favorite games on my computer. I like to compete online with other players and win tournaments. My parents think I should spend more time studying and less time playing games, but I don't listen to them. I want to become a professional gamer someday and make lots of money.

Figure 7-15 : Output V - Passage

# 8. FUTURE ENHANCEMENTS

Even though the model provides satisfactory result, still it is far from perfection. There are large datasets available for grammar error correction, which contains varying grammar errors, like Google C4 data. This T5 model can be trained on that data if there is no limitation of resources, as the training dataset is large, the associated training parameters are also likely going to be large. In this project, the model can only generate output sequence of 128 words due to resource limitations, even though the T5 model can output up to 512 words. We did not have enough resource to train the model for 512-word sequence.

The tokenizer used for encoder-decoder models is custom, so it cannot handle the out of the vocabulary words very well. Instead of that, pre trained word embeddings could be implemented to improve those models. And the tokenizer used in the T5 model is also pretrained and there are multiple such pre trained tokenizers. Rather than doing so, making a proper tokenizer for English language would've been better. But it is very data intensive task and could improve text to text generation models.

This model could be implemented as a browser extension, which could help many people to write correct English while using the web browser. And other services could also be provided. Like making a web application that could take in a pdf file or word file and parse it and correct the grammar and return it to the user. In this way, this project can be further implemented and enhanced in the future.

# 9. CONCLUSION

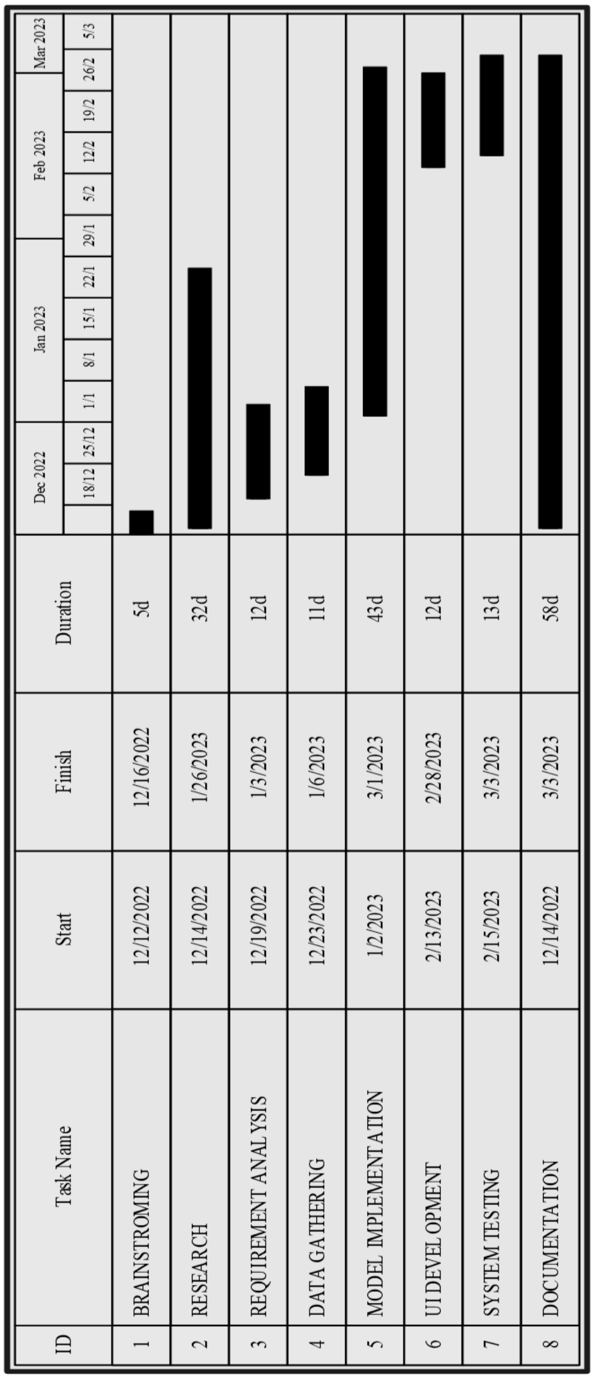
This project focuses on mitigating the problem by detecting the possible errors and correct grammar, punctuation, and syntax errors. There were three different models built for this project which are encoder-decoder LSTM model, encoder-decoder model with attention mechanism and T5 transfer Transformer model. The dataset was formed by the combination of J HU Fluency-Extended GUG Corpus, CoNLL-2014 and Colossal Clean Crawled Corpus whose size was around seventy thousand. When this dataset was used, the model starts to get overfitted after some epochs into training. Initially, the data was trained using encoder-decoder LSTM model which provides poor result. So, new methods were searched to use the whole dataset for training.

In the later phase of this project, Kaggle's service was used to train models using whole dataset. Each of the three models was trained using the training dataset. The problem like overfitting and underfitting were eliminated in attention model compared to the regular encoder-decoder model. But the T5 model being a transfer model, gives very low validation loss. Different metrics like BLEU scores and ROUGE scores were used for evaluation. The score results showed that the Encoder-Decoder Attention LSTM model was slightly better than Encoder-Decoder LSTM model. But T5-based GEC model showed significant improvement in evaluation metrics scores. So, T5-model was found to be the best trained model.

## 10. APPENDICES

### Appendix A: Project schedule GANTT chart

Table 9-1: GANTT chart of project timeline



| ID | Task Name | Start | Finish | Duration |
|---|---|---|---|---|
| 1 | BRAINSTROMING | 12/12/2022 | 12/16/2022 | 5d |
| 2 | RESEARCH | 12/14/2022 | 1/26/2023 | 32d |
| 3 | REQUIREMENT ANALYSIS | 12/19/2022 | 1/3/2023 | 12d |
| 4 | DATA GATHERING | 12/23/2022 | 1/6/2023 | 11d |
| 5 | MODEL IMPLEMENTATION | 1/2/2023 | 3/1/2023 | 43d |
| 6 | UI DEVELOPMENT | 2/13/2023 | 2/28/2023 | 12d |
| 7 | SYSTEM TESTING | 2/15/2023 | 3/3/2023 | 13d |
| 8 | DOCUMENTATION | 12/14/2022 | 3/3/2023 | 58d |

## Appendix B: Code Snippets

```
# Define the encoder model

encoder_inputs = Input(shape=(inc_train_seq.shape[1],))
encoder_embedding = Embedding(vocab_size, embedding_dim, input_length=inc_train_seq.shape[1]
(encoder_inputs)
encoder = LSTM(encoder_units, return_sequence=True, return_state=True)
encoder_outputs, state_h, state_c = encoder(encoder_embedding)
encoder_states = [state_h, state_c]
```

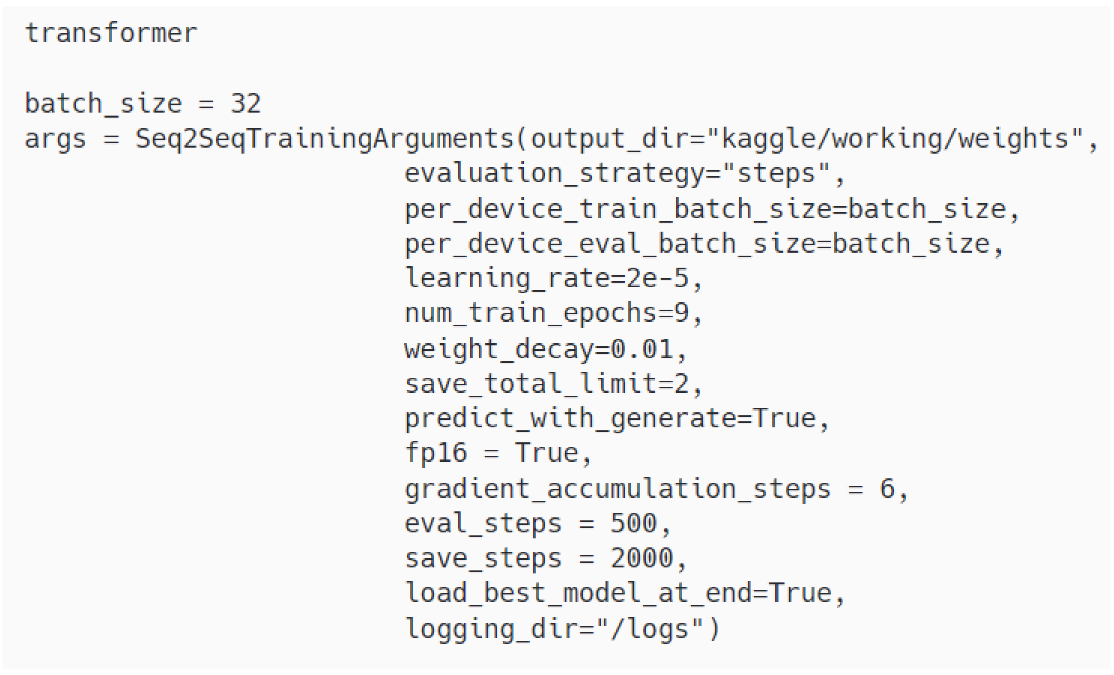
```
# Define the decoder model

# decoder_embedding = Embedding(vocab_size, embedding_dim)
# decoder_embedded = decoder_embedding(decoder_inputs)

decoder = LSTM(decoder_units, return_sequences=True, return_state=True)
decoder_outputs = decoder(encoder_embedding, initial_state=encoder_states)[0]
```

```
# Attention Mechanism
attention = dot([decoder_outputs, encoder_outputs], axes=[2, 2])
attention = Activation('softmax', name='attention')(attention)

context = dot([attention, encoder_outputs], axes=[2,1])
decoder_combined_context = Concatenate()([context, decoder_outputs])
```

```
# Dense layer for prediction
outputs = TimeDistributed(Dense(vocab_size, activation='softmax'))(decoder_combined_context)

model = Model(inputs=encoder_inputs, outputs=outputs)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

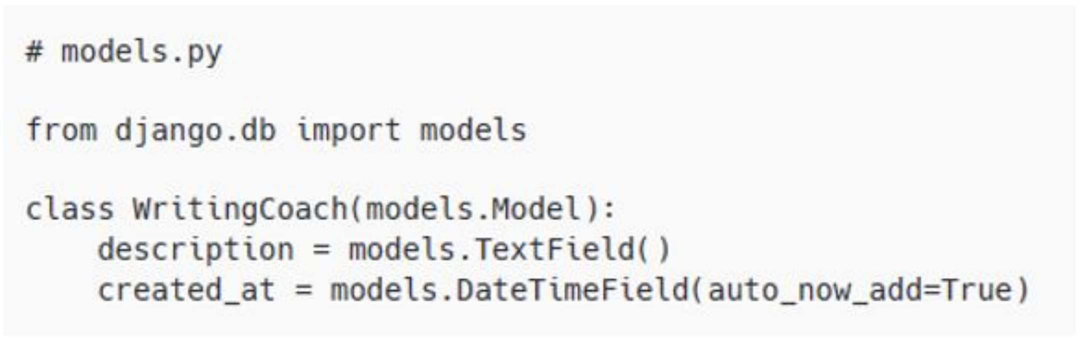
```
Encoder Decoder model

inputs = Input(shape=(inc_train_seq.shape[1],))
embedded = Embedding(vocab_size, embedding_dim, input_length=inc_train_seq.shape[1])(inputs)
encoder = LSTM(encoder_units, return_state=True)
encoder_outputs, state_h, state_c = encoder(embedded)
encoder_states = [state_h, state_c]
```

```
Encoder Decoder model
decoder = LSTM(decoder_units, return_sequences=True)
decoder_outputs = decoder(embedded, initial_state=encoder_states)
```

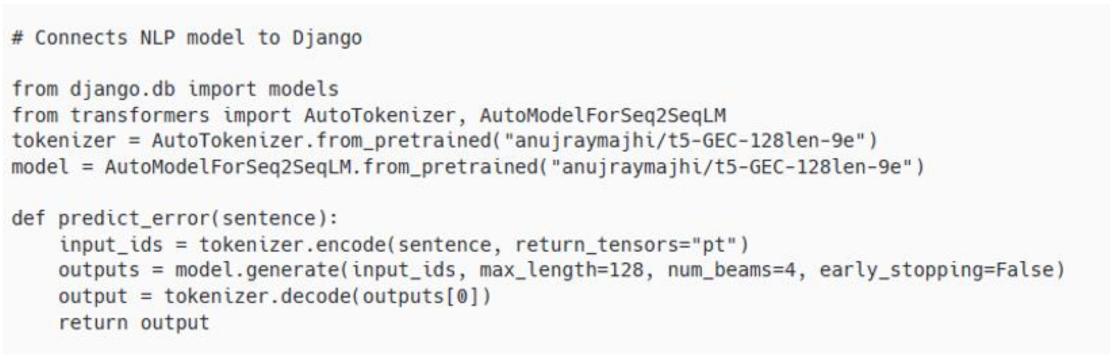


```
transformer

batch_size = 32
args = Seq2SeqTrainingArguments(output_dir="kaggle/working/weights",
                                evaluation_strategy="steps",
                                per_device_train_batch_size=batch_size,
                                per_device_eval_batch_size=batch_size,
                                learning_rate=2e-5,
                                num_train_epochs=9,
                                weight_decay=0.01,
                                save_total_limit=2,
                                predict_with_generate=True,
                                fp16 = True,
                                gradient_accumulation_steps = 6,
                                eval_steps = 500,
                                save_steps = 2000,
                                load_best_model_at_end=True,
                                logging_dir="/logs")
```

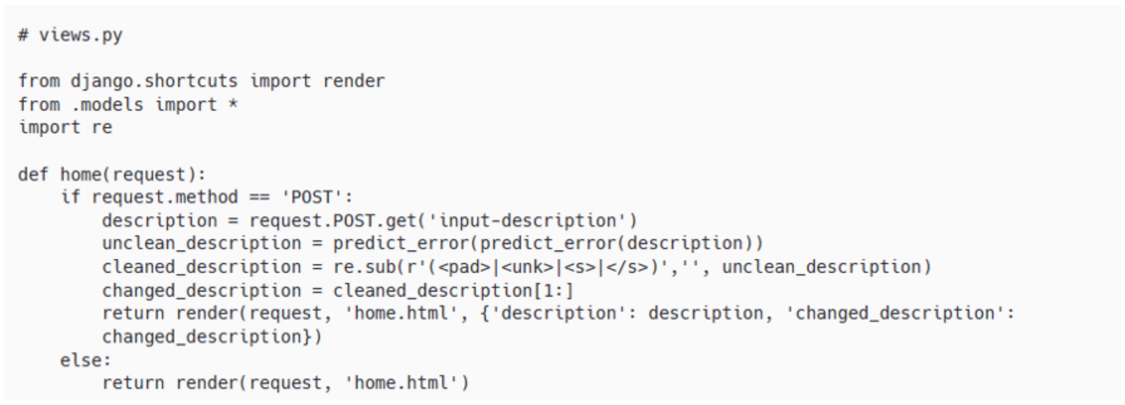
Figure 9-1 : Code Snippets of NLP Models



```
# models.py

from django.db import models

class WritingCoach(models.Model):
    description = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```



```
# Connects NLP model to Django

from django.db import models
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
tokenizer = AutoTokenizer.from_pretrained("anujraymajhi/t5-GEC-128len-9e")
model = AutoModelForSeq2SeqLM.from_pretrained("anujraymajhi/t5-GEC-128len-9e")

def predict_error(sentence):
    input_ids = tokenizer.encode(sentence, return_tensors="pt")
    outputs = model.generate(input_ids, max_length=128, num_beams=4, early_stopping=False)
    output = tokenizer.decode(outputs[0])
    return output
```



```
# views.py

from django.shortcuts import render
from .models import *
import re

def home(request):
    if request.method == 'POST':
        description = request.POST.get('input-description')
        unclean_description = predict_error(predict_error(description))
        cleaned_description = re.sub(r'(<pad>|<unk>|<s>|</s>)','', unclean_description)
        changed_description = cleaned_description[1:]
        return render(request, 'home.html', {'description': description, 'changed_description':
        changed_description})
    else:
        return render(request, 'home.html')
```

Figure 9-2 : Code Snippets of Web application

## References

[1] H. S. A.-K. Nora Madi, "Grammatical Error Checking Systems: A Review of Approaches and Emerging Directions," in *Thirteenth International Conference on Digital Information Management (ICDIM 2018),* Berlin, 2018.

[2] M. Agrawal, "medium.com," medium.com, 19 05 2021. [Online]. Available: https://medium.com/analytics-vidhya/grammatical-error-correction-using-neural-networks-aaf3e9fc91c. [Accessed 14 01 2023].

[3] Z. He, "English Grammar Error Detection Using Recurrent Neural Networks," *Scientific Programming,* vol. 2021, no. special, p. 8, 2021.

[4] G. H. a. Y. w. Jiahao Wang, "An Automatic Grammar Error Correction Model Based on Encoder-Decoder Structure for English Texts," *EAI Endorsed Transactions on Scalable Information Systems,* vol. 10, no. 1, p. 10, 2022.

[5] e. a. Ilya Sutskever, "Sequence to Sequence Learning with Neural Networks," in *NIPS'14: Proceedings of the 27th International Conference on Neural Information Processing Systems,* Montreal, 2014.

[6] H. P. C. D. M. Minh-Thang Luong, "Effective Approaches to Attention-based Neural Machine Translation," 20 September 2015. [Online]. Available: https://arxiv.org/abs/1508.04025.

[7] e. a. Ashish Vaswani, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017,* Long Beach, CA, 2017.

[8] N. S. A. R. K. L. S. N. M. M. Y. Z. W. L. P. J. L. Colin Raffel, "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer," 28 July 2020. [Online]. Available: https://arxiv.org/abs/1910.10683.

[9] Y. M. Tomoya Mizumoto, "Lang-8," NAIST, 01 01 2020. [Online]. Available: https://docs.google.com/forms/d/e/1FAIpQLSflRX3h5QYxegivjHN7SJ194OxZ4XN_7Rt0cNpR2YbmNV-7Ag/viewform. [Accessed 13 01 2023].