Grammatical Error Correction for Indian Language

A Project report submitted by

Puja Gupta

in partial fulfillment of the requirements for the award of the degree of

M.Tech

॥ त्वं ज्ञानमयो विज्ञानमयोऽसि ॥

Indian Institute of Technology Jodhpur

Department of Mathematics

May 2023

Declaration

I hereby declare that the work presented in this Project Report titled Grammatical Error Correction for Indian Languages submitted to the Indian Institute of Technology Jodhpur in fulfilment of the requirements for the award of the degree of M.Tech, is a bonafide record of the research work carried out under the supervision of Dr. Gaurav Harit and Dr. Kirankumar Hiremath. The contents of this Project Report in full or in parts, have not been submitted to, and will not be submitted by me to, any other Institute or University in India or abroad for the award of any degree or diploma.

Puja Gupta
(M21MA004)

Certificate

This is to certify that the Project Report titled **Grammatical Error Correction for Indian Language**, submitted by ***Puja Gupta (M21MA004)*** to the Indian Institute of Technology Jodhpur for the award of the degree of M.Tech, is a bonafide record of the research work done by her under my supervision. To the best of my knowledge, the contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Gaurav Harit
Department of Computer Science
Indian Institute of Technology, Jodhpur

Dr. Kirankumar R. Hiremath
Department of Mathematics
Indian Institute of Technology, Jodhpur

Acknowledgement

I would like to thank my supervisors, Dr. Gaurav Harit and Dr. Kirankumar Hiremath for their continuous support and guidance. I want to extent my gratitude towards all the faculty members of my department who make me understand all the concepts. I would also like to thank my friends, family members and lab mates for helping me in all the situations specially Rajat Bhardwaj.

Abstract

As the name suggests, Grammatical Error Correction (GEC) is the task of error detection and correction in the text. Though the problem seems easy, due to the diverse vocabulary and complex language rules, it is pretty tough to achieve. Writing is one of the most common means of communicating and sharing ideas and information; thus, a model which is capable of grammatically correct sentences in a fast and efficient way is much needed. Many models and techniques are available for grammatical error correction in English, such as Grammarly, Google docs, and numerous other text editors. However, very few works are available in the Indian languages.

In recent years, the use of transformers and other attention-based models for various multilingual tasks has motivated us to develop a state-of-the-art model for performing grammatical error correction tasks for Hindi. Three different architectures have been used to implement the GEC task: the Sequence-to-Sequence model (with and without attention using different word embeddings), the Transformer-based architecture, and Copy Augmentation Transformer model. The datasets developed by Etoori et al.[1] and Ankur et al.[2] for Hindi helped us train our model. The Non-Contextual Word embeddings such as Word2Vec, Glove, and FasText were used for training the Sequence-to-Sequence model. The Sequence-to-Sequence model trained using the attention mechanism outperformed the one without attention, as it allows to put focus on the more significant part of the input over others. Also, the best results were obtained from the model trained using the fastText word embedding.

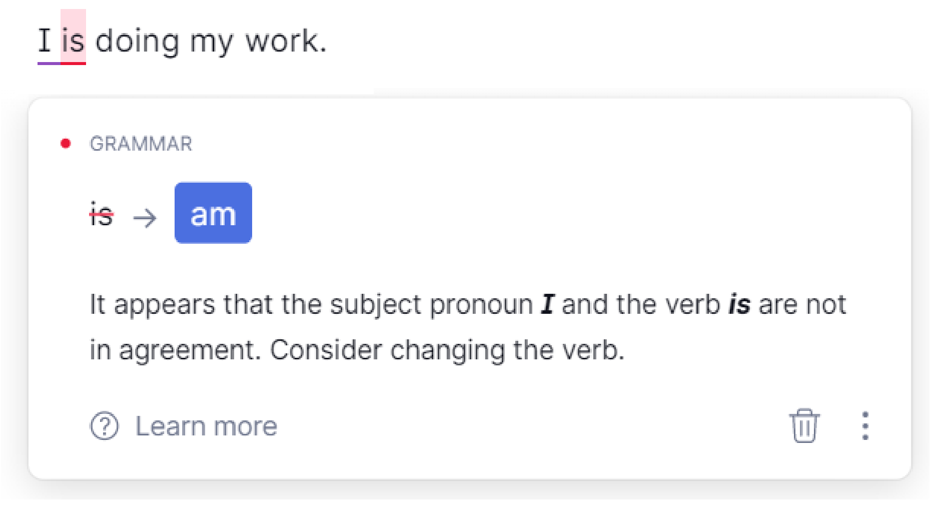
Among transformer-based architectures, the different models trained and analyzed were the Encoder-Decoder model (initialized with pre-trained "bert-base-multilingual-cased"), T5ForConditionalGeneration (initialized with pre-trained "t5-base") and mT5ForConditionalGeneration (initialized with pre-trained "mt5-small") using three different tokenizers (bert-base-multilingual-cased, MuRIL, and RoBE-RTa). Also, results were reproduced for the Copy Augmentation Transformer model used by Ankur et al. [2] in his work for comparison.

It was observed that all variations of the transformer model (with pre-trained model initialization) gave better results than the Sequence-to-Sequence model, with T5ForConditionalGeneration (using MuRIL Tokenizer), giving the best results among all. Also, the choice of tokenizer and word embedding can significantly impact the model's performance. A detailed comparative study and analysis have been done regarding all the model's performance, size, and inference time.

Contents

1 Introduction

As the name suggests, Grammatical Error Correction (GEC) is the task of error detection and correction in the text. As one of the most important research problems in NLP, GEC can be formulated as a sequence-to-sequence task, where a model is trained using grammatically incorrect sentences as input and return a grammatically correct sentence. Though the problem seems easy, but due to the diverse vocabulary and complex language rules, it is quite tough to achieve. Writing is one of the most common means (other than speaking) for sharing ideas and information or communicating with each other; thus, a model capable of grammatically correcting languages is much needed by writers to speed up their work with minimal error. Also, such a model is of great help to individuals who are not fluent in any language. With the advancement in AI and Deep Learning, along with abundant textual data being available, a lot of research work is available for GEC tasks in English. Some commonly used applications implementing GEC for English are Grammarly, Google docs, and numerous other text editors. As it is the basic need to get grammatically correct text in automatic typing, language translation, speech/image-to-text generation, or any other language problem, a similar model (fast and memory efficient) should be available for other Indian languages as well.



I is doing my work.

GRAMMAR

~~is~~ → am

It appears that the subject pronoun ***I*** and the verb ***is*** are not in agreement. Consider changing the verb.

Figure 1.1: Example of Grammatical Error Correction for English

1.1 Motivation

Many models and techniques are available for grammatical error correction in the English language, but not much work has been done for the Indian languages. The few existing models are mostly rule-based or use Statistical Machine Translation Techniques. With the expanding internet traffic with millions of Indian users online, the market for incorporating other Indian languages for content has just exploded. With this, there is a need for some tools for implementing GEC tasks for Indian languages as well which can help the writers.

In recent years, the use of transformers and other attention-based models for performing various multilingual tasks has motivated us to develop a model for performing grammatical error correction tasks for Hindi. Since GEC using neural networks is a very data extensive task, millions of parallel data should be available for training the model to ensure proper learning is done. The work by Etoori et al.[1] and Ankur et al.[2] focused on generating such datasets for Hindi, helped us to carry out our baseline experimentation and analysis in this direction and develop models based on different architectures for achieving GEC for the Hindi language.

1.2 Objective

We focus on building a state-of-the-art model for performing the Grammatical Error Correction task for Hindi Language and generalizing the process so the same task can be implemented for other low-resource Indian languages using their own language-specific dataset. As of now, unlike English, no tool is available for Grammatical Error Correction in Hindi. So building such a model, which is efficient and fast, will ease human life and can be incorporated into a range of text-generating and editing tools.

The primary focus of this project was learning the different architectures available for performing GEC tasks and comparing their performance with the present state-of-the-art model. In the project's initial phase, the main objective was to experiment and train the Sequence-to-Sequence model (with and without attention) using different word embeddings for Hindi and regenerate the benchmark results obtained by Garg[6] in her work with necessary modifications to improve the model's performance. Since the functional code implemented by Garg[6] was not available, the entire experiments and analysis are done from scratch. In the second phase, after obtaining the best-performing Sequence-to-Sequence model, the focus of the project shifted towards finding different architectures that can be used for the task, such as Transformer-based models and Copy-Augmented Transformer models.

1.3 Thesis Organization

The project work is divided into two phases: Phase 1 (Semester 3) and Phase 2 (Semester 4). In Phase 1, I conducted a comprehensive literature review of the GEC task available for Hindi and implemented Sequence-to-Sequence models (with and without attention) using three different word embeddings: Word2Vec, Glove, and FastText. Building upon this work, in Phase 2, I explored other advanced architectures for the GEC task in English such as Transformer-based models and Copy Augmentation and adapted them for Hindi to perform a detailed comparison of model performance using various architectures, tokenizers, and word embeddings. Additionally, I compared the performance of my models with the present state-of-the-art model for the GEC task in Hindi, listed in the chapters mentioned below.

There are total 8 chapters in this thesis. Chapter 1 consists of the introduction, the motivation behind the project, and the objectives we are trying to achieve. Chapter 2 consists of the literature review in the related field, a brief description of the datasets available and used for both English and Hindi GEC, and the different architectures used for this task. Chapter 3 contains information about Tokenizers and different Tokenizers available for Hindi Language and used in this project. Chapter 4 contains information about different word embeddings (Non-Contextual and Contextual) available and the ones used. Chapter 5 consists of the different architectures proposed for GEC for the Hindi language. Chapter 6 includes the complete analysis and experimentation, along with the result achieved for performing the task of Grammatical Error Correction for English and Hindi Languages using different models. Chapter 7 and 8 includes the conclusion and the list of future work that can be done in this field.

2
Literature Review

A detailed study on grammatical error correction for English has been done by Garg[6]. Unlike English, very less work has been done for Hindi and other Indian languages. Figure 2.1 summarizes all the related work done for GEC in Hindi.

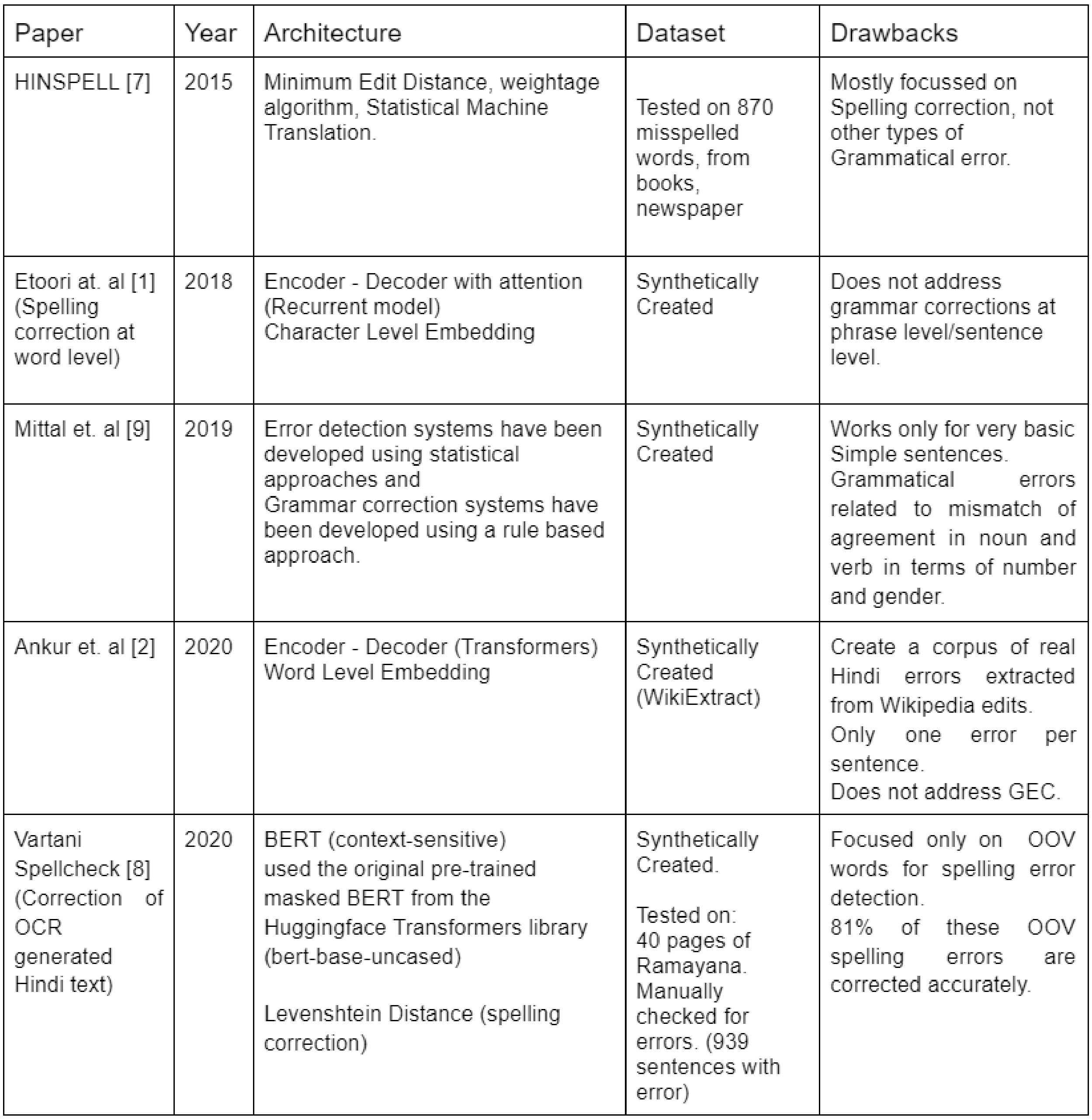
2.1 Dataset Used

Grammatical Error Correction in NLP is a data-extensive task requiring millions of data to train the state-of-the-art models and ensure proper learning. The dataset required for this task consists of parallel supervised data, which are correct and incorrect sentence pairs. Though many benchmark datasets, like LANG8, NUCLE, WI-LOCNESS, CoNLL-2014, etc., are available for English, the nonavailability of good-quality datasets for many Indian languages is a significant problem.

2.1.1 English Datasets

- **LANG8 [3]** dataset is generated using a Japanese website where users post incorrect sentences from the language they are learning, and native speakers of those languages correct them. Those sentences where both incorrect and correct sentences are available are used.
- **NUCLE [3]** corpus consists of 1,400 essays written by the National University of Singapore's students on different topics. These essays are annotated with error tags and corrections by the English instructor. This dataset has also been used in the CoNLL-2014 and BEA-2019 shared tasks.
- **WI-LOCNESS [3]** consists of two datasets: 1. LOCNESS, a corpus of essays written by native English students, which are annotated with errors. 2. Cambridge English Write and Improve (W&I), an online platform where users post incorrect sentences in the form of writings, essays, letters, stories, etc., are manually annotated and corrected.
- **CoNLL-2014 [4]** dataset is created by a system that detects the grammatical errors present in the input texts (short English essays written by non-native English speakers) and returns the corrected essays.
- **Colossal Clean Crawled Corpus (C4) [10]** is one of the largest language datasets available, with more than 156 billion tokens collected from over 365 million domains across the internet. It was used to train the T5 transformer model, which is one of the largest trained transformer models available.

For English, we have taken the LANG8 dataset to train the model.



Paper	Year	Architecture	Dataset	Drawbacks
HINSPELL [7]	2015	Minimum Edit Distance, weightage algorithm, Statistical Machine Translation.	Tested on 870 misspelled words, from books, newspaper	Mostly focussed on Spelling correction, not other types of Grammatical error.
Etoori at. al [1] (Spelling correction at word level)	2018	Encoder - Decoder with attention (Recurrent model) Character Level Embedding	Synthetically Created	Does not address grammar corrections at phrase level/sentence level.
Mittal et. al [9]	2019	Error detection systems have been developed using statistical approaches and Grammar correction systems have been developed using a rule based approach.	Synthetically Created	Works only for very basic Simple sentences. Grammatical errors related to mismatch of agreement in noun and verb in terms of number and gender.
Ankur et. al [2]	2020	Encoder - Decoder (Transformers) Word Level Embedding	Synthetically Created (WikiExtract)	Create a corpus of real Hindi errors extracted from Wikipedia edits. Only one error per sentence. Does not address GEC.
Vartani Spellcheck [8] (Correction of OCR generated Hindi text)	2020	BERT (context-sensitive) used the original pre-trained masked BERT from the Huggingface Transformers library (bert-base-uncased) Levenshtein Distance (spelling correction)	Synthetically Created. Tested on: 40 pages of Ramayana. Manually checked for errors. (939 sentences with error)	Focused only on OOV words for spelling error detection. 81% of these OOV spelling errors are corrected accurately.

Figure 2.1: Related work done for GEC in Hindi

Datasets	Training Sentences	Validation Sentences	Test Sentences
LANG8	1.9M	-	-
NUCLE	57K	-	-
WI-LOCNESS	34.3K	4.4K	4.5K
CoNLL-2014	57K	-	1.3K
Colossal Clean Crawled Corpus (C4)	200M	-	-

Table 2.1: Statistics of datasets available for the English Language

2.1.2 Hindi Datasets

Etoori et al.[1] and Ankur et al.[2] in their work, have focused on generating parallel supervised (correct and incorrect sentence pairs) datasets for Hindi.

- **Etoori's Dataset [1]** has created and released a synthetically generated parallel dataset containing incorrect and correct sentence pairs.
- **HiWikEd [2]** created a parallel corpus (HiWikEdits) of synthetic errors by inserting errors into grammatically correct sentences using a rule based process (focusing specifically on inflectional errors). The incorrect sentence generated by them has only one error per sentence. Also, a test corpus of real Hindi errors was extracted by them from Wikipedia edits for evaluating the performance of models on real-time errors.

Datasets	**Training Sentences**	**Validation Sentences**	**Test Sentences**
Etoori's Dataset	140K	30K	30K
HiWikEd	2.6M	-	13K

Table 2.2: Statistics of datasets available for Hindi

The dataset generated by Etoori et al.[1] has been used in our project to carry out the baseline experimentation and train all the model. HiWikEd data has also been used to further train the T5ForConditionalGeneration model in order to compare the performance with the existing state-of-the-art models.

2.2 Architectures

Initially, for the machine translation task, the work of text correction started and was implemented using basic Recurrent Neural Network (RNN), Gated recurrent units (GRU), and Long short-term memory (LSTM) models. With the advancement of deep learning, researchers started shifting from using a base model to more advanced sequence-to-sequence and transformer models, which showed more promising results. The models are trained using both word and character-level embeddings, which have their own advantages as per the application.

2.2.1 Sequence-to-Sequence Encoder Decoder Model

2.2.1.1 Sequence2Sequence Encoder Decoder Model Without Attention

Sutskever et al.[11] proposed a sequence-to-sequence architecture using multilayered LSTM (Long Short-Term Memory) units to map the input sequence of words fed to the Encoder in the form of a context embedding vector. This context embedding vector obtained from the final layer of the Encoder is then fed to another Decoder (LSTM-based) unit to decode the target sequence. The input and output sequence lengths can or cannot be the same. This architecture was first used to perform the English-to-French machine translation task. The sequence-to-sequence model showed better performance than the previously used Statistical Machine Translators. The LSTM units used in the Encoder and Decoder also helped in the sensible phrase and sentence representation. Though in the work proposed by Sutskever et al.[11], LSTM was used, but any other architecture such as RNN, GRU, and bi-direction LSTM can also be used in the Encoder and Decoder. The same architecture can also be used for the Grammatical Error Correction task. Figure 2.2. shows a Seq2Seq encoder decoder model without attention.

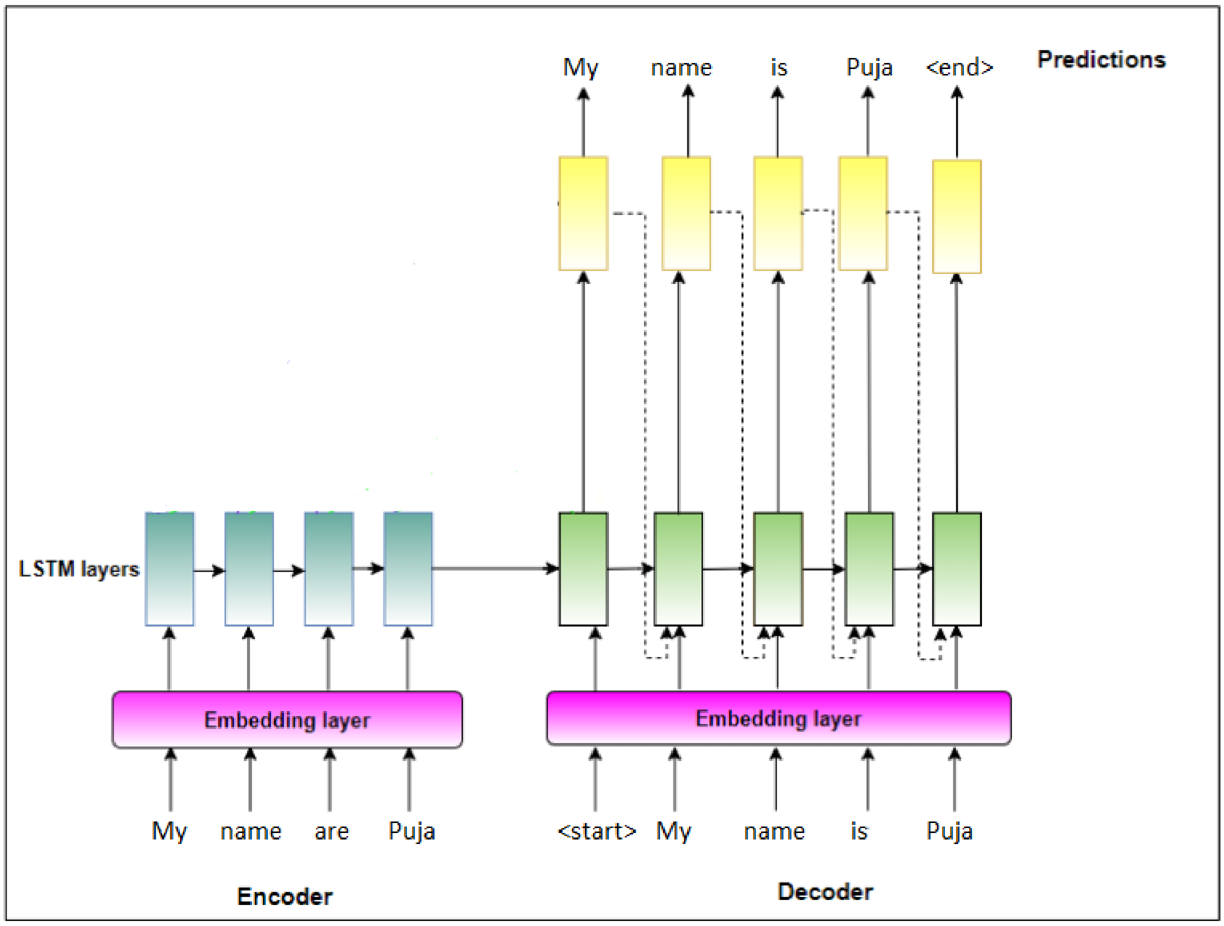


Figure 2.2: [6] Seq2Seq Encoder Decoder model without Attention

2.2.1.2 Sequence2Sequence Encoder Decoder Model with Attention

Sequence-to-Sequence model proposed by Sutskever et al.[11] has the disadvantage of not performing better when we have long sentences. The Seq2Seq model compresses the information about the input sentence into a fixed-length context vector embedding. However, if the input sentence is longer, there will be a greater information loss leading to an information bottleneck. It has been shown that the accuracy decreases as the sentence length increases with respect to the encoder and decoder. Vaswani et al.[12] proposed the concept of an attention mechanism to overcome this disadvantage.

The use of the attention mechanism lets the decoder look back to the source (gives access to the encoder's hidden states) and learn how much attention/focus needs to be given to each encoder's hidden state for this decoder timestamp. Attention weights are calculated for each decoder's timestamp, concatenated with the context vector and input embedding, and fed to the decoder for decoding the output sentence. It was seen that incorporating the attention mechanism significantly improved the performance of the encoder-decoder model. Tao et al. [13] and Schmaltz et al.[14] have used the attention-based sequence-to-sequence architecture for implementing grammatical error correction for the Chinese language. Figure 2.3. shows a Seq2Seq encoder-decoder model with attention.

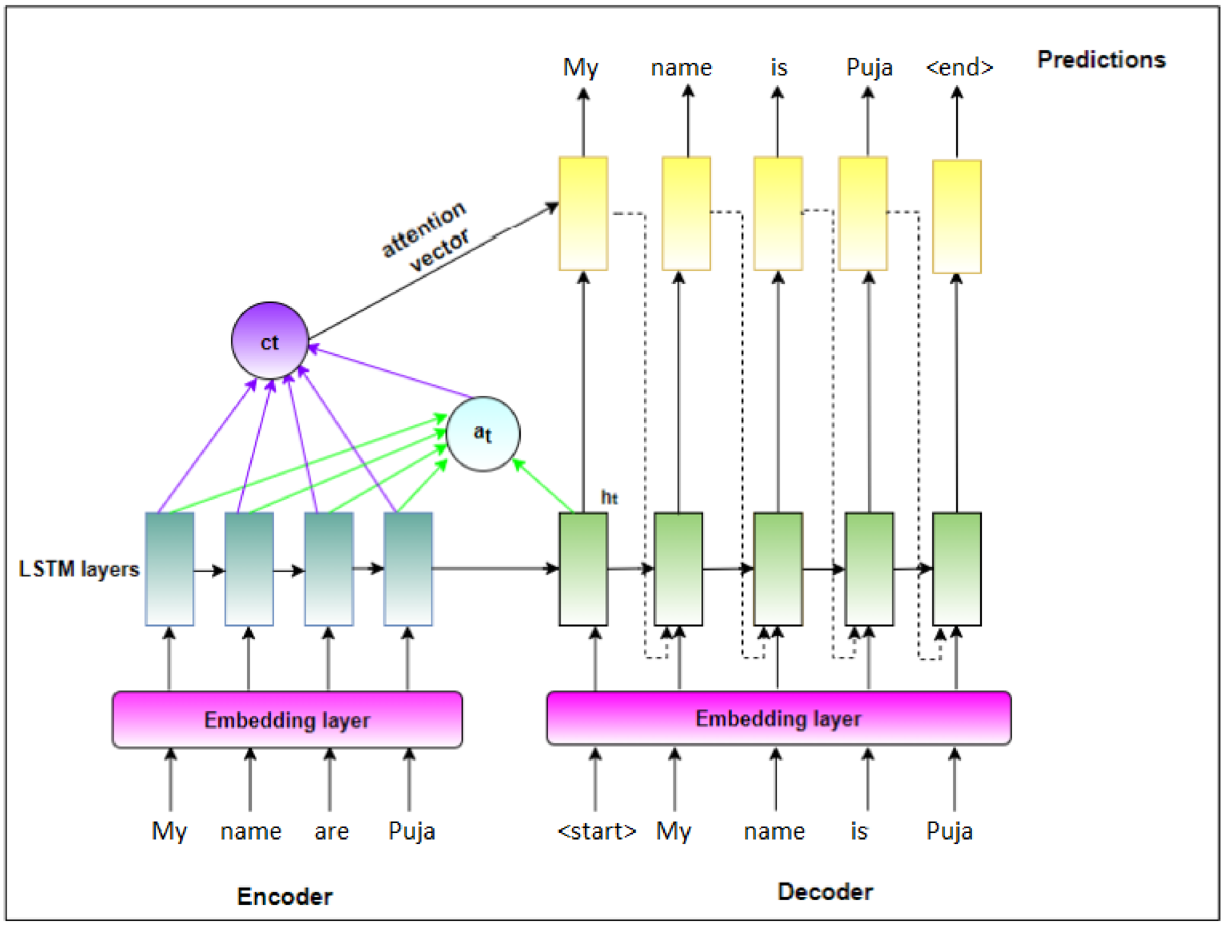


Figure 2.3: [6] Seq2Seq Encoder Decoder model with Attention mechanism for GEC in English

2.2.2 Transformer Based Models

The ”Attention is all you need” paper, published in 2017 by Vaswani et al. [12], introduced the Transformer architecture (Figure 2.4), which revolutionized NLP tasks such as question answering, machine translation, and language modeling. The Transformer architecture uses the concept of self-attention, which helps the model to find out the (weigh) importance of different parts of the input sequence when processing each element of the sequence. Unlike traditional recurrent neural network (RNN) architectures, which process the input sequence sequentially, the Transformer can process the entire sequence in parallel, making it much faster and more scalable.

The two critical components of the Transformer architecture are the self-attention mechanism and the feedforward neural network. The self-attention mechanism helps the model focus on different parts (find the weigh) of the input sequence when processing each element of the sequence. The feedforward neural network provides a non-linear mapping between the input and output spaces. One of the most significant contributions of the Transformer architecture is its ability to handle variable-length input sequences. This makes it well-suited for tasks such as machine translation, where the length of the input and output sequences can vary greatly.

The Transformer architecture has become the basis for many state-of-the-art NLP models, such as BERT, GPT-2, and T5. These models have achieved remarkable results on various NLP tasks, including language modeling, question answering, and sentiment analysis.

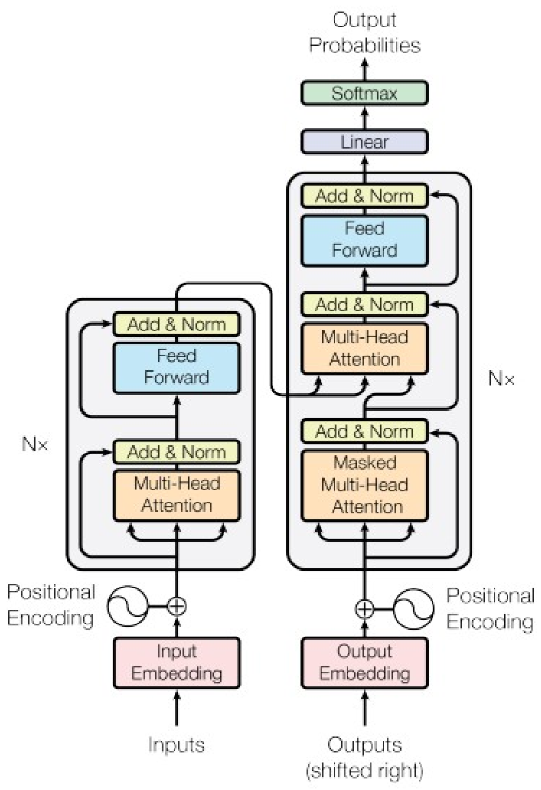


Figure 2.4: Transformer model Architecture proposed by Vaswani et al. [12]

2.2.2.1 Encoder Decoder Model

The work of Rothe et al. [20] suggests that leveraging pre-trained checkpoints can be an effective approach for improving the performance of sequence generation tasks. EncoderDecoderModel can be initialized from a pre-trained encoder checkpoint and a pre-trained decoder checkpoint. I have used a BERT model (bert-base-multilingual-cased) as both the encoder and decoder part.

- Encoder part : pretrained bert-base-multilingual-cased
- Decoder part : pretrained bert-base-multilingual-cased

BERT (Bidirectional Encoder Representations from Transformers) is a pre-trained model that can be fine-tuned for various natural language processing (NLP) tasks. ”bert-base-multilingual-cased” is one of the pre-trained BERT models provided by Google, which is designed to support 104 different languages (including Hindi). The ”cased” in its name indicates that the model distinguishes between upper and lower case characters in the input text. The ”multilingual” in its name indicates that the model can handle text from multiple languages without requiring language-specific pre-processing. This model was trained on a huge amount of text data from Wikipedia, books, and web pages in multiple languages. The pre-training process involves masking certain words in the input text, predicting the masked words based on the surrounding context, and training the model to predict the next sentence in a text sequence.

The configuration of the bert-base-multilingual-cased:

- Parameters: 179M
- Number of layers: 12
- Hidden size: 768
- Number of attention heads: 12

The fine-tuning process involves adapting the pre-trained model to a specific NLP task, such as text classification, named entity recognition, or question-answering. By fine-tuning the model on a small amount of task-specific data, the model can learn to perform the task with high accuracy. However, this model has not been pre-trained for the task of grammatical error correction.

2.2.2.2 T5 Model

Google introduced the T5 (Text-to-Text-Transfer-Transformer) architecture [18] in 2020, a unified ”Text-to-Text” framework that proposed a methodology to solve every NLP task as a text-to-text problem. With the text-to-text framework, the same model, loss function, and hyperparameters can be used for any NLP task, including document summarization, question answering, machine translation, and classification tasks (such as sentiment analysis).

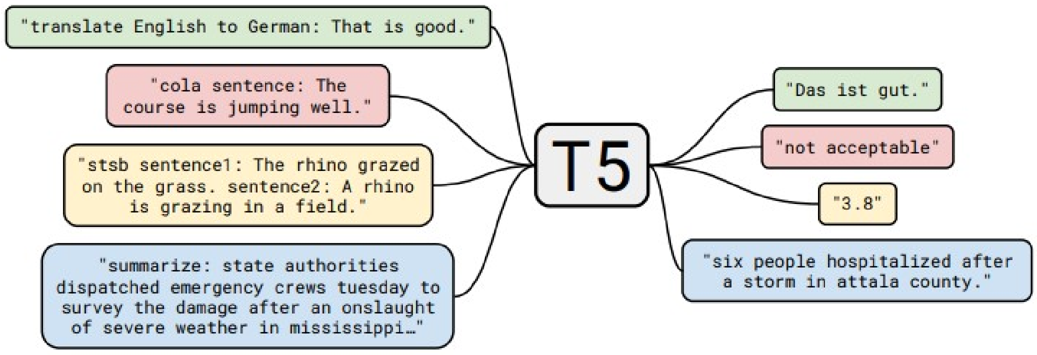

"translate English to German: That is good."
"cola sentence: The course is jumping well."
"stsb sentence1: The rhino grazed on the grass. sentence2: A rhino is grazing in a field."
"summarize: state authorities dispatched emergency crews tuesday to survey the damage after an onslaught of severe weather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

Figure 2.5: [18] T5 Text-to-Text framework

Like the standard transformer architecture, T5 is a transformer-based encoder-decoder model consisting of 12 encoder-decoder blocks. Each block contains a feed-forward network, self-attention, and optional encoder-decoder attention. The t5 model uses relative scalar embeddings. It is trained using teacher forcing, meaning for all the input sequence, we always need a corresponding target sequence for training. It has been pre-trained on multi supervised and unsupervised tasks.

The T5ForConditionalGeneration model is a T5 model variant designed explicitly for text generation tasks. It includes the same transformer encoder as the T5 model. However, it replaces the transformer decoder with a simple linear layer, which allows the model to generate text by sampling from a probability distribution over the model's vocabulary. T5ForConditionalGeneration is the complete seq2seq model with a language modeling head.

T5 library also includes other versions of the architecture for each model. For example, T5Model is the bare T5 model that outputs the raw hidden states without a specific head on top, while T5EncoderModel outputs the hidden raw states of the encoder. The T5 model is designed to perform a wide range of NLP tasks, including text classification and summarization, whereas the T5ForConditionalGeneration model is designed specifically for text generation tasks. The T5ForConditionalGeneration model is also more lightweight than the full T5 model. There are different pre-trained T5 models available which can be fine tuned for the required downstream tasks [18] : t5-small, t5-base, t5-large, t5-3B and t5-11B.

The configuration of different pre-trained T5 models are listed in the Table 2.3:

Configuration	**t5-small**	**t5-base**	**t5-large**	**t5-3B**	**tr-11B**
Parameters	60M	220M	770M	2.8B	11B
Encoder-decoder layers	6	12	24	24	24
Hidden states	512	768	1024	1024	1024
Feed-Forward Hidden states	2048	3072	4096	16384	65536
Attention heads	8	12	16	32	128

Table 2.3: Configuration of different pre-trained T5 models.

mT5 transformer architecture is also available [19], which is the multilingual variant of T5 that was pre-trained on a new Common Crawl-based dataset covering 101 languages. Unlike T5, the mT5 model was pre-trained only on the mC4 dataset; therefore, it has to be fine-tuned for any of the required downstream tasks.

2.2.3 Copy Augmented Transformer Model

Zhao et al. [21] in their work propose a pre-training framework to improve the performance of grammatical error correction models for English. They introduced a new copy-augmented architecture for GEC that incorporates copy mechanisms to allow the model to copy input tokens to the output sequence. This architecture allows the model to handle rare or out-of-vocabulary words better and improves the overall performance of the GEC model. They used a FairSeqModel by Facebook AI Research as the base Architecture which is a transformer model. They augmented the current architecture to incorporate the copying mechanism that copies the unchanged and out-of-vocabulary words directly from the source sentence. The configuration of this architecture is mentioned in Chapter 5, Section 5.4.

As shown in the Figure 2.6, the decoder model is a mixture of P_{vocab} and P_{copy} models. P_t is the Final Probability distribution, P_t^{gen} is the Generation Distribution, P_t^{copy} is the Copy Distribution and $\alpha_t^{copy} \in [0, 1]$ is the Balancing factor (Controls the balance between copying and generating at each time stamp t.

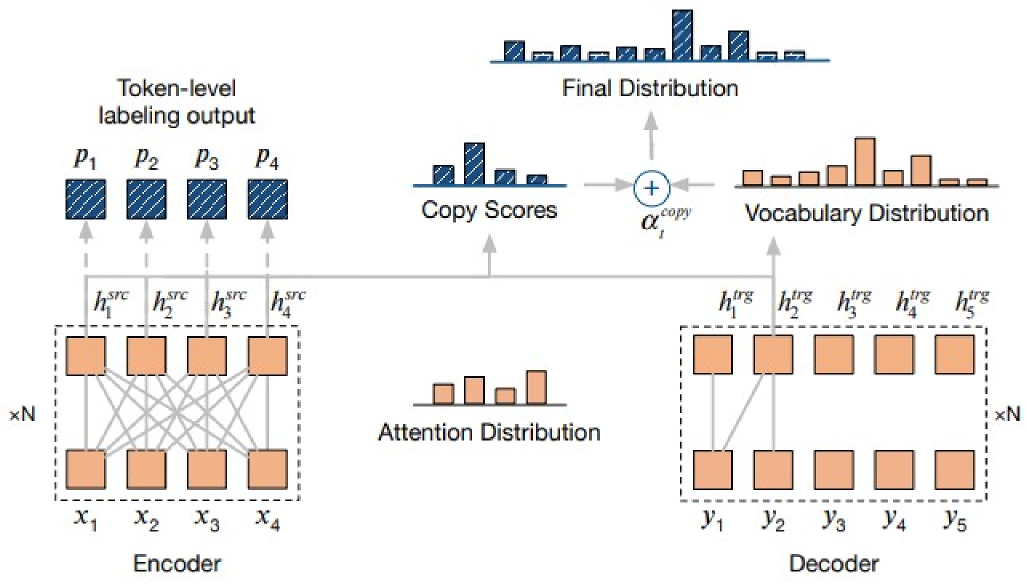

Final Distribution
Token-level labeling output
p_1 p_2 p_3 p_4
Copy Scores
Vocabulary Distribution
α_t^{copy}
h_1^{src} h_2^{src} h_3^{src} h_4^{src}
h_1^{trg} h_2^{trg} h_3^{trg} h_4^{trg} h_5^{trg}
×N
Attention Distribution
×N
x_1 x_2 x_3 x_4
Encoder
y_1 y_2 y_3 y_4 y_5
Decoder

Figure 2.6: Copying Augmented Transformer model proposed by Zhao et al. [21]

$$P_t(w) = (1 - \alpha_t^{copy}) * P_t^{gen}(w) + \alpha_t^{copy} * P_t^{copy}(w)$$

Depending upon the value of P^t_{copy}, the model can dynamically switch between either generating words from vocabulary or directly from source input.

Ankur et al. [2] has already implemented the above Copy Augmented Transformer-based architecture for the Grammatical Error Correction Task in Hindi.

3
Tokenizers

Tokenization, one of the important concepts in NLP, is the process of breaking down the text into smaller units called tokens. The most common way of tokenizing text is to split it into words. However, other types of tokenization, such as character-level tokenization or subword-level tokenization, can also be used. Tokenization is usually the first step in most NLP tasks as it helps to convert the raw text data into a structured format that algorithms can efficiently process.

The choice of tokenizer can impact how the text data is processed and represented, which can affect the accuracy and efficiency of the models trained on this data. For example, different tokenizers may use different rules for splitting text into tokens. Some tokenizers may split words based on white spaces, while others may consider punctuation marks or special characters as separate tokens. As a result, the same text input may be represented differently depending on the tokenizer used. Moreover, certain tokenization methods may be more appropriate for specific tasks or languages. For instance, some languages have complex compound words that may need to be split into multiple subwords to capture their meaning accurately. Using a tokenizer that is designed to handle such cases may result in better performance for models trained in that language.

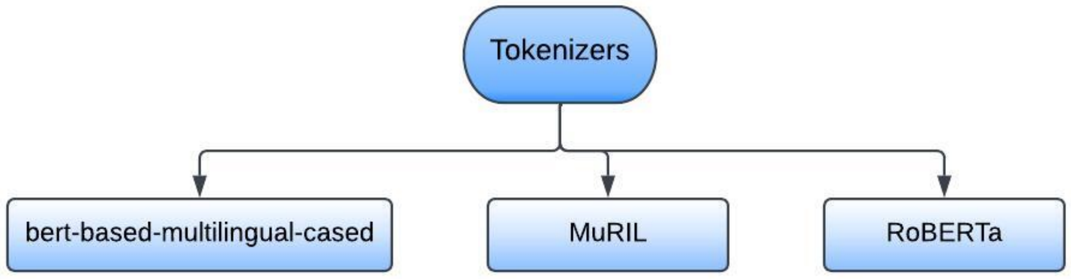


Figure 3.1: Different Tokenizers used to train the model

3.1 Bert-base-multilingual-cased

Bert-base-multilingual-cased [15] is a pre-trained language model provided by the Hugging Face Transformers library. It has been trained on a very large corpus of text data from multiple (104) languages and uses WordPiece tokenization algorithm to split words into subword units. This tokenizer is designed to handle text data from multiple languages and can be helpful for tasks that involve multilingual text data. The vocabulary size is 119547.

3.2 MuRIL

MuRIL is a pre-trained language model (for 17 Indian languages) [16] that is specifically designed for Indic languages. It has been trained on a very large corpus of text data from various Indic languages and uses a specific tokenization approach tailored for these languages. The MuRIL tokenizer uses a combination of morphological analysis and rule-based methods to split words into subword units, which can better capture the complex morphology of Indic languages. The vocabulary size is 197285 (only Indian Languages).

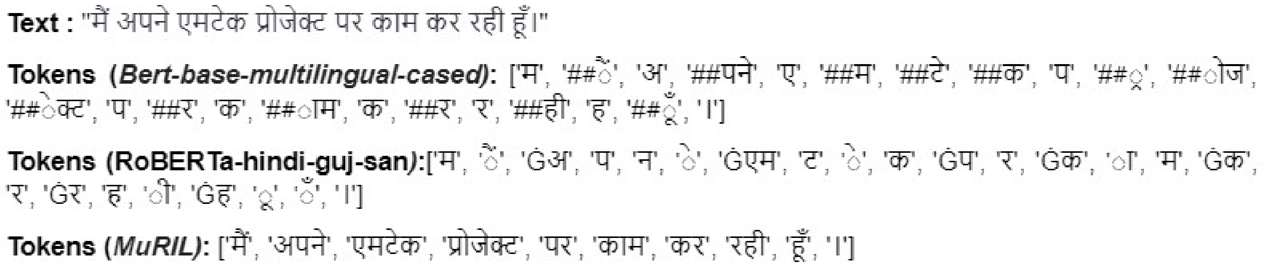
3.3 RoBERTa

The RoBERTa tokenizer is a subword tokenizer used for tokenizing text into smaller subword units. It is based on the Byte Pair Encoding (BPE) algorithm.

The tokenizer works by breaking down the input text into smaller subword units based on the frequency of occurrence of each subword. It starts by initializing a vocabulary of single characters and then iteratively combines the most frequent pairs of consecutive characters until the desired vocabulary size is reached. During tokenization, the tokenizer first converts the input text into Unicode codepoints and then applies normalization and tokenization. The normalization step involves applying Unicode Normalization Form C (NFC) to the input text to convert any equivalent characters to a single form. The tokenization step then breaks down the normalized text into subword units based on the RoBERTa vocabulary.

RoBERTa-hindi-guj-san is a Multilingual RoBERTa-like model trained on Wikipedia articles of Hindi, Sanskrit, and Gujarati languages (https://huggingface.co/surajp/RoBERTa-hindi-guj-san). The vocabulary size of the pre-trained model is 30522 (much less than the other two).

No other pre-trained model based on RoBERTa architecture is available for Hindi.



Text : "मैं अपने एमटेक प्रोजेक्ट पर काम कर रही हूँ।"

Tokens (*Bert-base-multilingual-cased*): ['म', '##ैं', 'अ', '##पने', 'ए', '##म', '##टे', '##क', 'प', '##्', '##ोज', '##ेक्ट', 'प', '##र', 'क', '##ाम', 'क', '##र', 'र', '##ही', 'ह', '##ूँ', '।']

Tokens (RoBERTa-hindi-guj-san*)*:['म', 'ैं', 'Ġअ', 'प', 'न', 'े', 'Ġएम', 'ट', 'े', 'क', 'Ġप', 'र', 'Ġक', 'ा', 'म', 'Ġक', 'र', 'Ġर', 'ह', 'ी', 'Ġह', 'ू', 'ँ', '।']

Tokens (*MuRIL*): ['मैं', 'अपने', 'एमटेक', 'प्रोजेक्ट', 'पर', 'काम', 'कर', 'रही', 'हूँ', '।']

Figure 3.2: Sentence Tokenization using different tokenizers

Tokenizer	Algorithm Used	Vocabulary Size	Tokenization Type
Bert-base-multilingual-cased	WordPiece	119547	Sub word level
MuRIL	WordPiece + Rule based method for Indian languages	197285	Word/Sub word level
RoBERTa-hindi-guj-san	Byte Pair Encoding	30522	Sub word level

Table 3.1: Comparison of different Tokenizers Used

Larger vocabulary allows to capture more fine-grained linguistic nuances and to handle rare or out-of-vocabulary words more effectively.

4
Word Embeddings

In NLP, the input we have (dataset) is in the form of textual data, which cannot be directly fed to the model, as the machine does not understand the textual data. So we convert the textual data into some word vector first (numerical form) and feed that vector representation of the word to the model for processing. Word embeddings are used to convert words into vector representation to make them understandable by the machines by capturing the different morphological, contextual, semantic, or syntactic relationships between the words. Models learn these word vectors through different unsupervised learning techniques, which enables them to store contextual information of words in a low-dimensional vector. We have various pre-trained state-of-the-art word embedding models available for English and Hindi. These can convert any textual sentence into the corresponding vector representation, which can be fed to the model to learn. There are two different types of word embeddings : Non Contextual Word Embeddings and Contextual Word Embeddings. Figure 4.1 shows different word embeddings used in the project along with the examples.

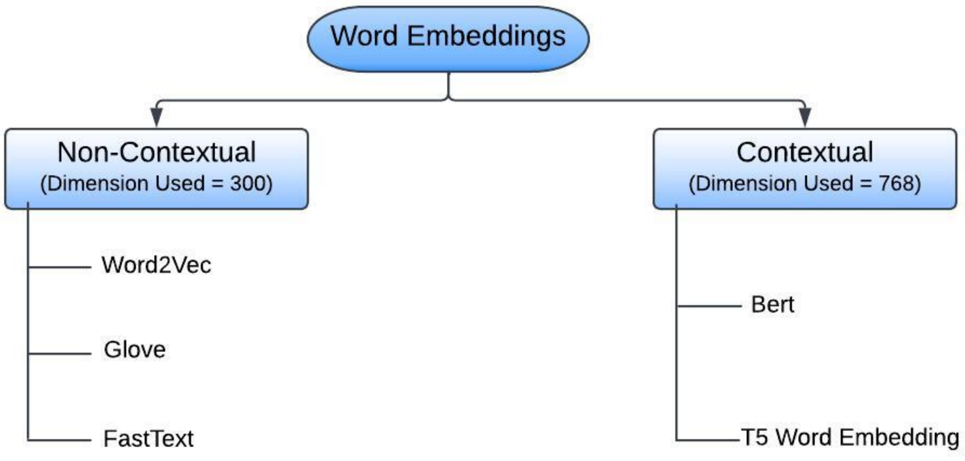


Figure 4.1: Different Word Embeddings used

4.1 Non-Contextual Word Embeddings

4.1.1 Word2Vec

Word2Vec provides the N-dimensional vector representation of the document vocabulary. If we plot the word embeddings in a 2D space, similar words are plotted together, and words unrelated to each other are plotted at a further distance. The spatial position of two words is calculated based on the Cosine Similarity between them.

The two different methods to obtain Word2Vec word embeddings (both processes involve training Neural Networks with very large datasets such as big long sequence of texts from Wikipedia) are Skip Gram and Common Bag of Words.

4.1.1.1 Continuous Bag-of-words

In Continuous Bag-of-words (CBOW) we take the context and predicts the central word based on the highest probability. Figure 4.2. shows a block diagram of CBOW. For simplicity, let's consider a language that has only five words (My, Final, M.Tech, Project, On, GEC). We initially encode each word as one-hot vector (one bit as one and the rest all 0). To generate word vectors from this vocabulary, we iterate through a corpus. For simplicity, let's say our corpus has just one sentence: ”My final M.Tech project on GEC.” Now we need to select a window size for iterating through this sentence (m=3). Since the window size is 3, we need to predict the center word using the two surrounding words. We need a simple neural network for this. In this case, the neural network will have the context word ”My” and ”Final” as input, fed to the neural network to predict ”Final.” The size of each vector also needs to be chosen (say 3), then the size of the hidden layer for our neural network will also be 3. Figure 4.3. shows the basic architecture and working of this neural network. The neural network tries to predict the central word given the context words. We pass it to the softmax function to get the probability and compare it with the actual word. The error is used to update the weights and then we slide the window to get all the weight matrix, which helps us to get is the set of vectors.

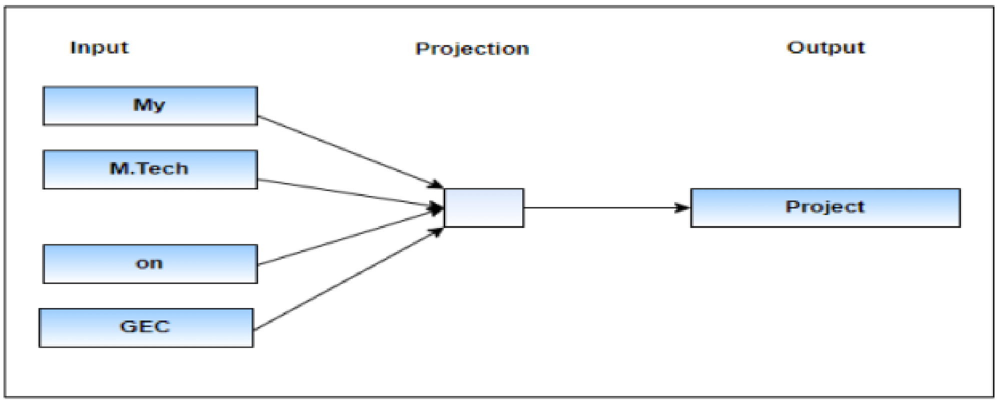


Figure 4.2: Common Bag Of Words

4.1.1.2 Skip Gram

In contrast to CBOW, in SkipGram (SG) we predict the the context (single or combination of words) based on the central word. So basically, for each central word, t = 1,...m, we predict the surrounding words in the ”m” window size by maximizing the objective/loss function, which is defined as the probability of any context word given the surrounding word. Figure 4.4. shows a

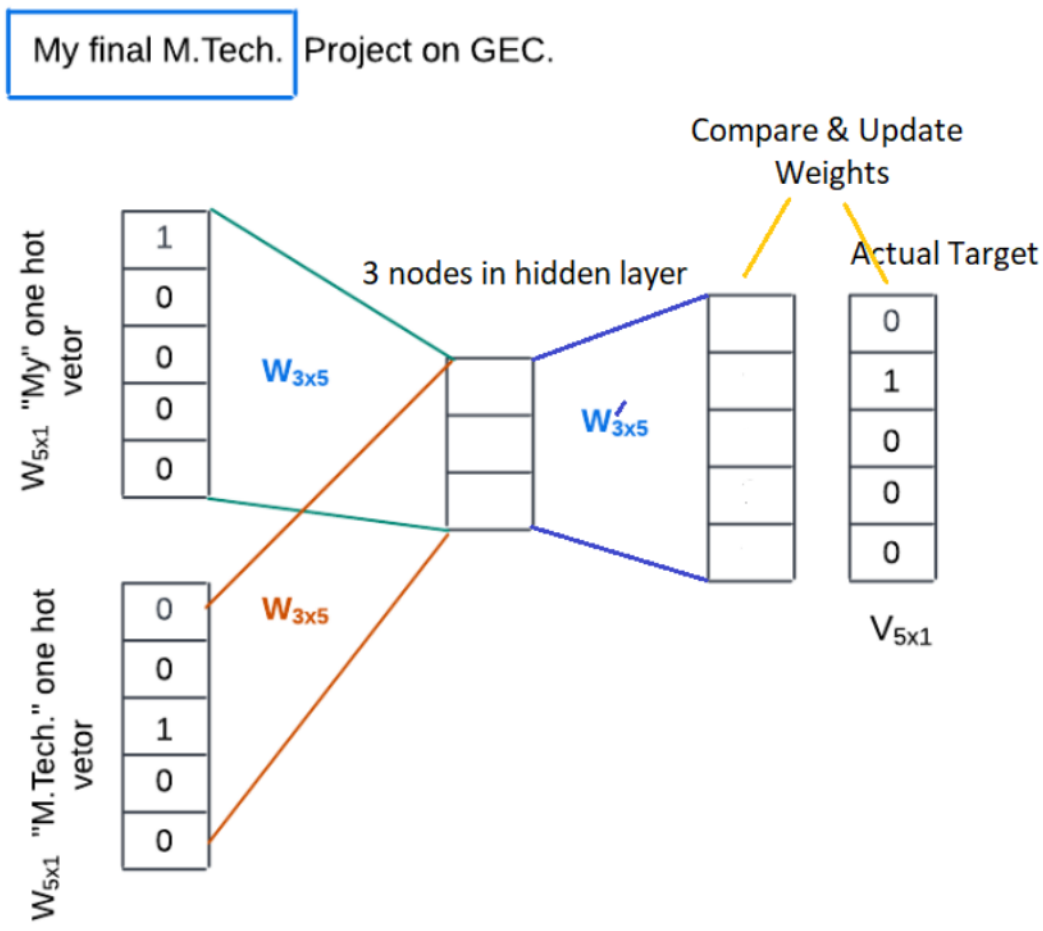


Figure 4.3: Working of CBOW Algorithm

block diagram of Skip Gram and Figure 4.5. shows the working of the SG architecture for the example discussed in section 4.1.1.1.

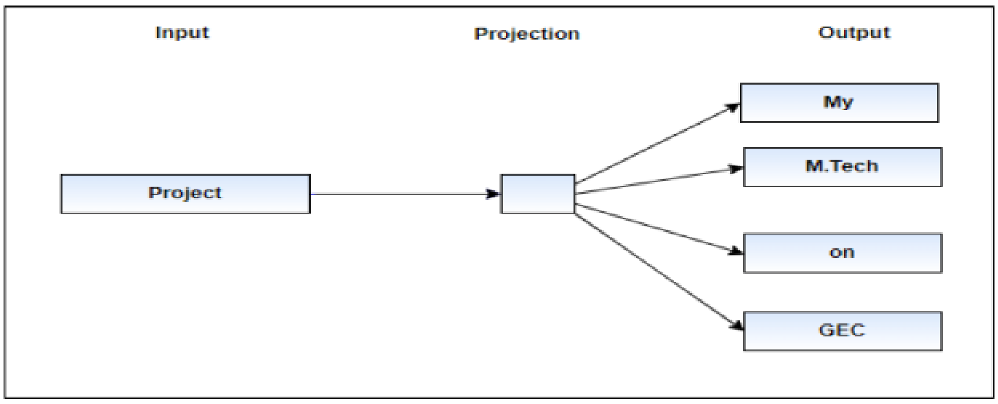


Figure 4.4: Skip Gram

Few drawbacks of Word2Vec are:

- It does not handle disambiguation i.e. it does not capture the context (sense) in which a word is being used.

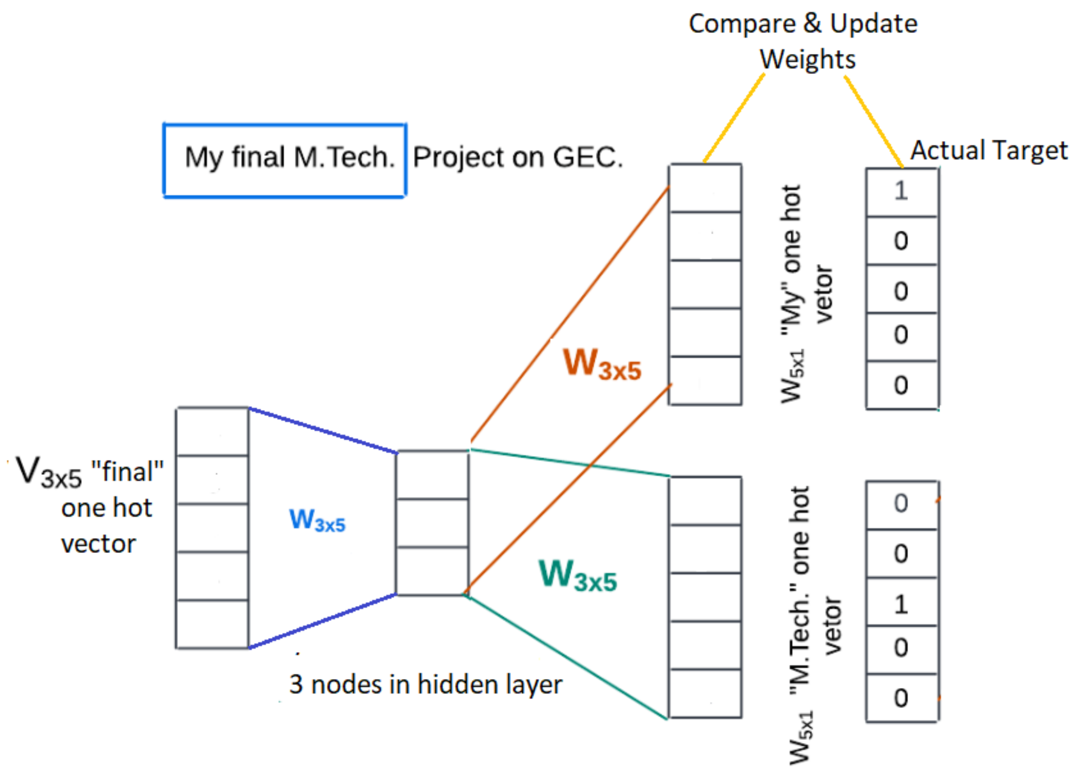


Figure 4.5: Working of Skip Gram

- It cannot handle OOV (Out of Vocabulary) words i.e. it won't be able to generate a vector representation of the words which are not seen during the training.
- It does not capture sub-word information of a word.

4.1.2 Glove

Glove, a refinement technique of Word2Vec, is another method of finding word embeddings. Unlike Word2Vec, global statistics are also considered to obtain the vector representation of words instead of just relying on the local context information of the words. Glove provides better performance than Word2Vec as it also captures the sub linear relationships of the word vectors. In Glove, relationships between word pairs are also taken into account rather than capturing just word and word relationships. Also, less weight is given to the most frequent word pairs to prevent meaningless stop words like a, an, the, etc., from dominating the training progress. One drawback of Glove is that it takes a lot of memory space as the model is being trained (using unsupervised learning) on the co-occurrence word matrix.

4.1.3 Fasttext

Fasttext is another technique for generating word embeddings and was created by the Facebook Research Team. Unlike Word2Vec, the n-gram technique is used to generate the word representation in Fasttext. This n can vary from 1 to length on word, and these n-gram words are used along with the original word vocabulary for training the model. We basically break the word into multiple chunks and add two tokens at the start and end of the word chunks. The new chunks formed and the whole word are combined and fed to the neural network to predict the next word. For simplicity, let's say we have one and given one word fed to the neural network want to predict the next word. sentence: ”My final project on GEC.” So if n=3, the word ”final” will be broken down into [fin,ina,nal] and along with complete f’final” will be fed to the neural network to predict the next word ”project”. Figure 4.6 shows the working of above example using Fasttext

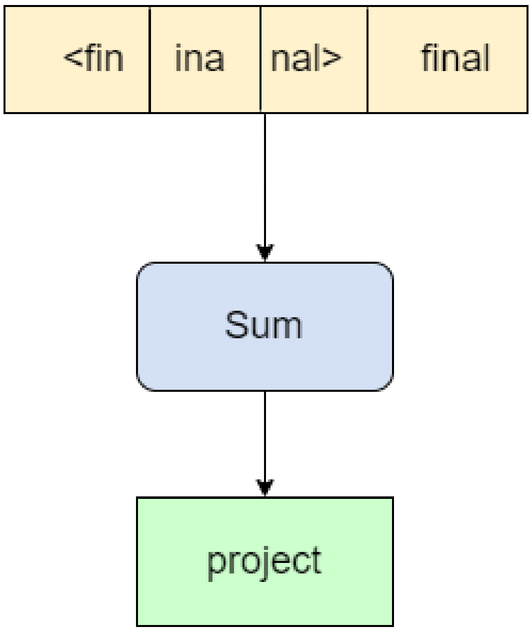


Figure 4.6: Working of FastText

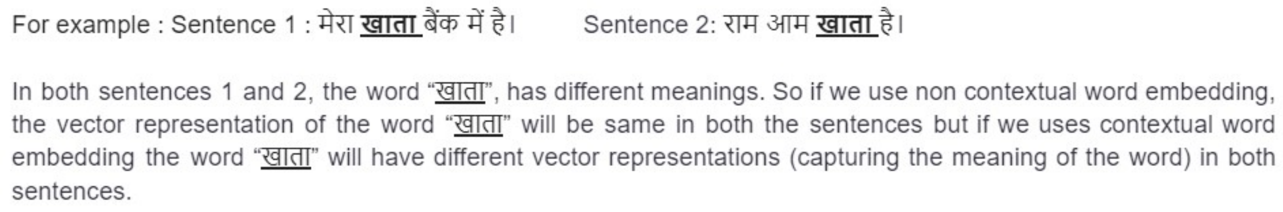
Advantage of using Fasttext over Word2Vec and Glove:

- Unlike Word2Vec and Glove, it can give word vectors for OOV (Out of Vocabulary) and rare words because a single word is further broken into character n-grams.
- While learning word vector representation, it also considers the internal structure of a word and can be used with smaller datasets.

4.2 Contextual Word Embeddings

Non-Contextual Word Embeddings such as Word2Vec, Glove, and Fasttext usually suffer the drawback of not capturing the sense or context of a word used in a sentence, as the same word can have different meanings in different sentences. However, each word will have a single representation in these word embeddings. Contextual Word Embeddings such as ELMo, BERT, MuRIL, etc., are dynamic embeddings and can be used to overcome these drawbacks. These models consider the context in which a word has been used so that the same word can have different vector representations in different sentences (explained with example in Figure 4.7).

The main advantage of contextual word embeddings is their ability to capture more nuanced semantic relationships between words as they consider the context in which the words appear. This makes them particularly useful for tasks such as sentiment analysis, named entity recognition, and machine translation. Non-contextual word embeddings, on the other hand, are simpler and faster to train, making them a good choice for tasks such as document classification and topic modeling where context is less important.



For example : Sentence 1 : मेरा **खाता** बैंक में है। Sentence 2: राम आम **खाता** है।

In both sentences 1 and 2, the word "खाता", has different meanings. So if we use non contextual word embedding, the vector representation of the word "खाता" will be same in both the sentences but if we uses contextual word embedding the word "खाता" will have different vector representations (capturing the meaning of the word) in both sentences.

Figure 4.7: Difference between Non-Contextual and Contextual Word Embedding

4.2.1 BERT

BERT is a pre-trained language model widely used in natural language processing tasks. The embedding layer in BERT is the first layer in the model, and it takes as input a sequence of tokens, where each token is represented as an integer index. The embedding layer then maps each token index to a dense vector representation, which captures the meaning of the token in the context of the input sequence.

The BERT embedding layer is different from traditional embedding layers in that it combines position embedding and segment embedding to capture the context of the tokens. Position embeddings encode the position of each token in the sequence. In contrast, segment embeddings encode whether a token belongs to a sequence's first or second segment (e.g., in a sentence pair classification task). During pre-training, the BERT embedding layer also uses masked language modeling (MLM), which randomly masks some of the input tokens and requires the model to predict the masked tokens based on the context. This helps the model learn more robust representations to handle missing or noisy input.

In BERT word embedding, after the input sequence is tokenized into subword units using different tokenizers, each subword unit is assigned a learned embedding vector. These subword embeddings are then fed into the model's transformer layers, which perform a series of computations to generate contextualized word embeddings. The contextualized word embeddings in BERT are generated by considering the entire input sequence in both forward and backward directions. This allows the model to capture complex dependencies between words in a sentence and generate more accurate embeddings that capture the meaning of each word in the context of the whole sentence.

Bert-base-multilingual-cased is a variation of bert model. The word embedding layer in bert-base-multilingual-cased is pre-trained on a large corpus of text data in multiple languages, which allows it to learn language-agnostic representations that can be used for a wide range of NLP tasks in various languages.

4.2.2 T5 Embedding Layer

The T5 model includes an embedding layer on top of the encoder that converts the input tokens into dense vector representations using the concept of Relative scalar embeddings. Relative scalar embeddings are a type of positional encoding used in the T5 model. In traditional positional encoding, the position of each token in the input sequence is represented by a fixed vector. In contrast, relative scalar embeddings use learnable scalar weights relative to the query's position and the key vectors in the multi-head attention mechanism.

Using relative scalar embeddings in T5 allows the model to better capture the relationships between tokens in the input sequence, as the scalar weights can be learned based on the specific task and input data. This can lead to improved performance on a wide range of NLP tasks. So, relative scalar embeddings are an important component of the T5 model architecture, and they help the model to better capture the context and relationships between tokens in the input sequence.

5
Proposed method for GEC

5.1 Sequence2Sequence Model with Attention

Many models and techniques are available for grammatical error correction in the English language, but not much work has been done for the Indian languages. The few existing models are mostly rule-based or use Statistical Machine Translation Techniques which is the motivation behind this project to propose state-of-the-art grammatical error correction tool for Hindi. A sequence-to-sequence word level model, as discussed in section 2.2.2, shown in figure 4.1 has been implemented with 1 Long short-term memory (LSTM) layer each in the Encoder and Decoder with an attention mechanism. Word-level input using different word embeddings, as discussed in the below subsection, has been used. The dataset proposed by Etoori. et al.[1] is used to train the above model, containing 140K parallel sentences and 10K test sentences. Exploratory Data Analysis was performed to get a sense of the dataset. Duplicate data entries and unnecessary spaces were removed from the dataset, leaving 1,39,798 training sentences and 10K test sentences. Indic NLP library for Tokenization was used to create the word vocabulary for the encoder and decoder. The encoder word vocabulary consists of 73829 words, and the decoder word vocabulary consists of 72971 words. The word embedding of incorrect sentences in the dataset is given to the encoder, and the word embedding of correct sentences along with the context word embedding generated from the final Encoder layer and learnt attention weights are given to the decoder to train the model. All models are trained for 50 epochs with Early stopping criteria on validation loss with patience 5. All the codes are written in TensorFlow using Keras libraries. Cuda version 11.1 and cuDNN version 8.1.1. are used. The experiment and analysis code repository is available on the 172.25.0.209 server under the user id gupta92.

5.1.1 Word2Vec

A pre-trained Word2Vec CBOW (Common Bag of Words) model for the Hindi language of size 4 GB by Kumar et al. [5] was downloaded and used in the project for generating the word embeddings. The dimension of embeddings used is 300. The word embeddings generated using this model are fed to both the encoder and decoder of the Seq2Seq model for training. The model is being trained for 50 epochs with an early stopping criterion monitoring the validation loss with patience five, i.e., the number of epochs monitored with no improvement, after which the model training is stopped. A batch size of 64 and 32 attention units is used to train the model.

5.1.2 Glove

A pre-trained Glove model on the Hindi language by Kumar et al. [5] of size 4.6 GB is used for generating the word embeddings. The word embedding generated for each word is of dimension

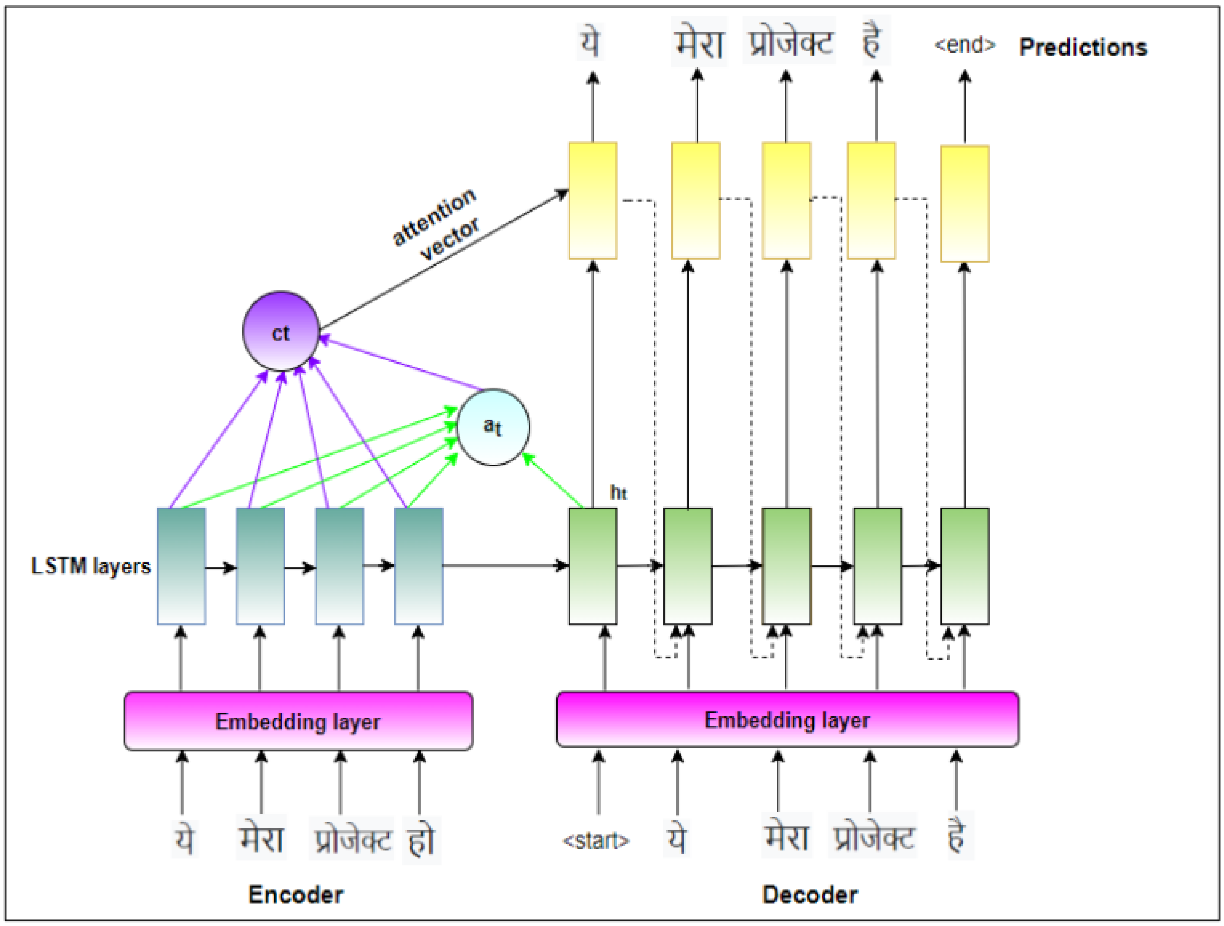


Figure 5.1: [6] Seq2Seq Encoder Decoder model with Attention for GEC in Hindi

300. The model is trained using these embeddings for 50 epochs with an early stopping criterion. A batch size of 32 and 32 attention units is used to train the model.

5.1.3 Fasttext

A pre-trained Fasttext model on the Hindi language of size 2.6 GB is downloaded using fastText library and used in our project for generating the word embeddings. The word embedding with dimension 300 is generated for the words of encoder-decoder vocabulary and is used to train the model for 50 epochs with an early stopping criterion (monitoring the validation loss) to avoid overfitting the model. A batch size of 32 and 32 attention units is used to train the model.

5.2 Transformer Based Models

The introduction of Transformer architecture by Vaswani et al. [12] revolutionized the NLP tasks as discussed in Chapter 2, section 2.2.2. Many transformer-based models are available for the Sequence-to-Sequence problem (translation) on Hugging Face. However, none of these transformer models have been used for the Grammatical Error Correction task for Hindi or other Indian languages. The work of Rothe et al. [20] suggests that leveraging pre-trained checkpoints can effectively improve the performance of sequence generation tasks.

Pre-trained multilingual-bert-based transformer models are available, which can be used with the EncoderDecoder model as discussed in Chapter 2, Section 2.2.2.1 for the GEC task. Rothe et al., in their work [22], uses the T5 architecture to achieve state-of-the-art results for the GEC task in four languages: English, German, Czech, and Russian. A multilingual variation on T5 (mT5), pre-trained in 104 languages, is also available, which can be used as well to achieve the task. All these transformer models have their baseline architecture similar to the model suggested by Vaswani et al. [12], with different numbers of the encoder, decoder, and attention layers. Also, these models are pre-trained on different specific tasks, which is mentioned in sections 5.2.1 and 5.2.2. All the trained model and are available on the 172.25.0.209 server under the user id gupta92.

5.2.1 Encoder Decoder Model

As discussed in Chapter 2, section 2.2.2.1, EncoderDecoder Model, a transformer based architecture can be initialized with pre-trained encoder and decoder checkpoints. I have used a pre-trained BERT model (bert-base-multilingual-cased) with 179M parameters as both the encoder and decoder parts. The model has been pre-trained on Self-supervised tasks (Masked language Modelling & Next Sentence Prediction)for 104 languages, including Hindi. It has 12 encoder, decoder, and attention layers, with the topmost layer of the encoder as a word embedding layer, which is used to generate the contextual vector representation of input source and target tokens. The word embedding dimension is 768. The dataset proposed by Etoori. et al. [1] is used to train the above model, containing 140K parallel (source and target) sentences and 10K test sentences. The bert-base-multilingual-cased tokenizer is used to tokenize the input sequence, with a vocabulary size of 119547. The model has been trained for two epochs with batch size 8, taking around 3.5 hrs to train. All the codes are written in PyTorch. Cuda version 11.1 and cuDNN version 8.1.1. are used. The experiment code repository and trained model are available on the172.25.0.209 server under the user id gupta92.

5.2.2 T5ForConditionalGeneration Model

Rothe et al., in their work [22], achieved state-of-the-art results for the GEC task in four languages: English, German, Czech, and Russian, using the T5 model, which is a Text-to-Text-Transfer-Transformer architecture. Taking this as motivation, I have used a pre-trained t5-base model to initialize the model weights for the GEC task in Hindi. The model is pre-trained on multitask mixture of unsupervised denoising & supervised learning tasks such as Sentence acceptability judgment, sentiment analysis, paraphrasing/sentence similarity, natural language inference, sentence completion, question answering, word sense disambiguation, etc. The pre-trained model has 220M parameters with 12 encoder, decoder, and attention layers, with the topmost layer of the encoder as a word embedding layer, which is used to generate the contextual vector representation of input source and target tokens using the concept of Relative scalar embeddings. The word embedding dimension is 768. MuRIL (a multilingual tokenizer by Google) and RoBERTa were used to tokenize the input sequence, with a vocabulary size of 197285 and 30522, respectively.

The dataset proposed by Etoori. et al. [1] is used to train the above model, containing 140K parallel (source and target) sentences and 10K test sentences. All the codes are written in PyTorch.

Cuda version 11.1 and cuDNN version 8.1.1. are used to train the model with batch sizes 8 and 7 epochs. Also, to compare the model's performance with results obtained by Ankur et al. [2] on the HiWikEd dataset, I have trained the model on an additional 2.6M Artificially generated dataset for one epoch.

Additionally, the multilingual variant of the T5 (mT5) model, which was pre-trained only on the mC4 dataset (it has to be fine-tuned for any of the required downstream tasks) is also trained on the 140K Etoori train dataset to check whether pre-training in multilingual languages improves the model performance. The model was initialized with pre-trained mT5-small (300M parameters), with 8 encoders, 8 decoders, and 6 attention layers. The word embedding dimension being used is 512, with a vocabulary size of 250112. All the codes are written in PyTorch. Cuda version 11.1 and cuDNN version 8.1.1. are used to train the model with batch sizes 1 and 5 epochs.

The experiment code repository and trained model are available on the 172.25.0.209 server under the user id gupta92.

5.3 Copy Augmented Model

Ankur et al. [2] has already implemented the Copy Augmented Transformer-based architecture, discussed in Chapter 2, Section 2.2.3 for the Grammatical Error Correction Task in Hindi. They have trained the model for 2.6M Artificially generated training dataset. In order to understand the implementation and pipeline of the model training, I have referred to their official GitHub repository. Using the command given in their official repository for cloning the project, training the model, and evaluating the results, I was able to train the model for the Etoori Training dataset instead, as the performance of the model obtained by training on 2.6M Artificially generated dataset is already mentioned in their work (Also it was taking a lot of time to train). Cuda version 10.0 (its not compatible for Cuda Version 11.1 used to train the other models) and cuDNN version 7.6.5. are used to train the model for 5 epochs.

The experiment code repository and trained model are available on the 172.25.0.209 server under the user id gupta92.

5.4 Model Comparison

Sequence-to-Sequence (Seq2Seq), Transformer-based, and Copy Augmented Transformer models are all neural network models used for natural language processing (NLP) tasks, but they differ in their architectures and use cases.

A Sequence-to-Sequence (Seq2Seq) model is a neural network model that maps an input sequence to an output sequence. It consists of two main components: an encoder and a decoder. The encoder takes the input sequence and encodes it into a fixed-size context vector, which is then passed to the decoder. The decoder generates the output sequence one token at a time, using the context vector and the previously generated tokens as input.

On the other hand, T5ForConditionalGeneration is a specific type of Seq2Seq model that is based on the Transformer architecture. It was introduced by Google and is pre-trained on a huge

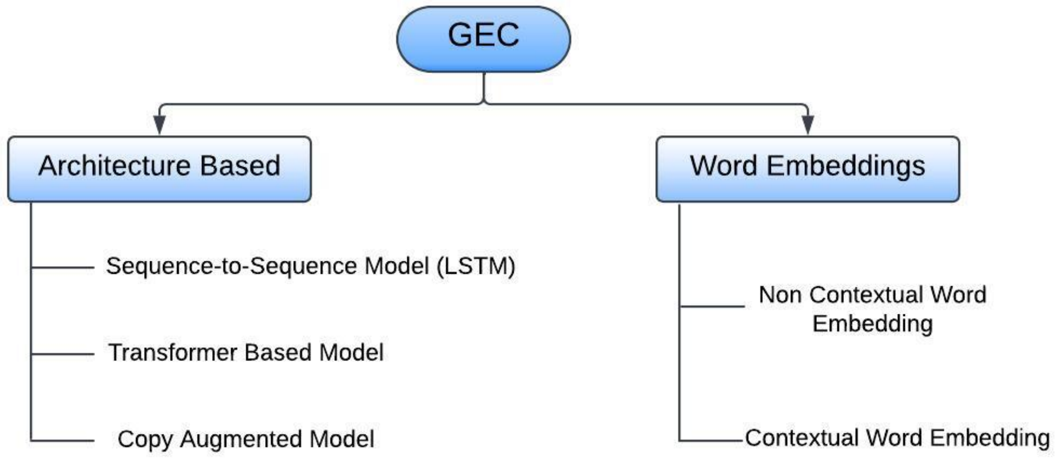

GEC
Architecture Based
Word Embeddings
Sequence-to-Sequence Model (LSTM)
Transformer Based Model
Copy Augmented Model
Non Contextual Word Embedding
Contextual Word Embedding

Figure 5.2: Complete Task

corpus of text data using a set of unsupervised learning objectives, including masked language modeling, denoising auto-encoding, and sequence-to-sequence pre-training. The T5 model can be fine-tuned on various supervised NLP tasks, such as text classification, question answering, and summarization. The T5ForConditionalGeneration model is a variant of the T5 model specifically designed for text generation tasks, such as machine translation and text summarization.

Whereas Copy-augmented Transformer architecture incorporates copying mechanisms that allows the model to copy input tokens directly to the output sequence. This architecture allows the model to handle rare or out-of-vocabulary words better and improves the overall performance of the GEC model. A detailed comparison of the Configurations of these Models are shown in the Figure 5.3.

5.5 Loss Function

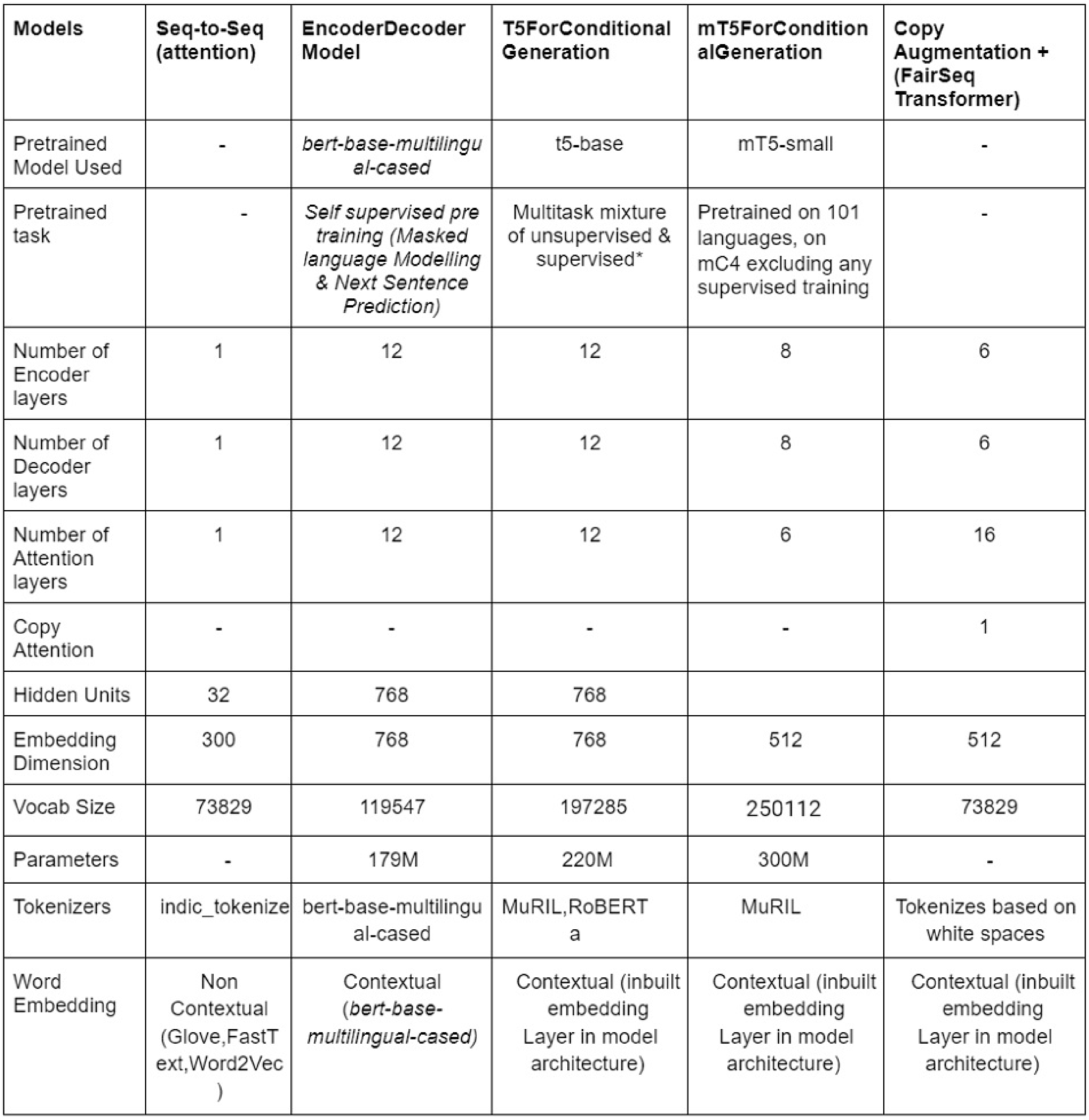
Sparse Categorical Cross Entropy Loss is used for calculating the loss while training the model. In the below equation y_i refers to true label and y_i' refers to predicted label.

$$J(w) = -\frac{1}{N}\sum_{i=1}[y_i log(y_i^{'}) + (1 - y_i) log(1 - y_i^{'})]$$

Unlike Categorical cross-entropy, where the true labels are one-hot encoded, in Sparse Categorical cross-entropy, the true labels are integer encoded.

5.6 Optimizer

Adam (Adaptive Moment Estimation) optimizer is a stochastic gradient descent (SGD) optimization algorithm widely used in machine learning, deep learning and NLP task. It is an adaptive



Models	**Seq-to-Seq (attention)**	**EncoderDecoder Model**	**T5ForConditional Generation**	**mT5ForCondition alGeneration**	**Copy Augmentation + (FairSeq Transformer)**
Pretrained Model Used	-	*bert-base-multilingu al-cased*	t5-base	mT5-small	-
Pretrained task	-	*Self supervised pre training (Masked language Modelling & Next Sentence Prediction)*	Multitask mixture of unsupervised & supervised*	Pretrained on 101 languages, on mC4 excluding any supervised training	-
Number of Encoder layers	1	12	12	8	6
Number of Decoder layers	1	12	12	8	6
Number of Attention layers	1	12	12	6	16
Copy Attention	-	-	-	-	1
Hidden Units	32	768	768		
Embedding Dimension	300	768	768	512	512
Vocab Size	73829	119547	197285	250112	73829
Parameters	-	179M	220M	300M	-
Tokenizers	indic_tokenize	bert-base-multilingu al-cased	MuRIL,RoBERT a	MuRIL	Tokenizes based on white spaces
Word Embedding	Non Contextual (Glove,FastT ext,Word2Vec)	Contextual (*bert-base-multilingual-cased)*	Contextual (inbuilt embedding Layer in model architecture)	Contextual (inbuilt embedding Layer in model architecture)	Contextual (inbuilt embedding Layer in model architecture)

Figure 5.3: Difference in Configuration of Used Models.
* : Unsupervised denoising objective , Supervised text-to-text language modeling objective (Sentence acceptability judgment, sentiment analysis, paraphrasing/sentence similarity, natural language inference,sentence completion, question answering, word sense disambiguation)

learning rate optimization algorithm which is used to compute an adaptive learning rates for each parameter, based on the estimate of 1st and 2nd moments of the gradients. The algorithm combines the advantages of two other optimization algorithms: AdaGrad and RMSProp. The Adam optimizer maintains an exponentially decaying average of past gradients and their squares, called the first and second moments, respectively. These estimates are used to calculate an adaptive learning rate for each parameter, which is then used to update the parameter values in the opposite direction of the gradient. The main advantages of Adam optimizer are that it requires less memory than other optimization algorithms and is well-suited for large-scale, high-dimensional problems.

AdamW optimizer is a variant of the Adam optimizer that incorporates weight decay, which is a regularization technique that penalizes large weights to prevent overfitting. The ”W” in AdamW stands for ”Weight decay”. Weight decay is commonly used to prevent overfitting and improve generalization performance. A penalty term is added to the loss function that is proportional to the squared magnitude of the weights. The regularization term encourages the weights to stay small, which reduces the complexity of the model and improves its generalization performance. The original Adam optimizer does not incorporate weight decay directly, which can lead to suboptimal performance and instability when the learning rate is high. AdamW optimizer adds weight decay to the parameter update step, which effectively scales down the weight parameters and reduces their magnitude, preventing overfitting and improving the generalization performance. The AdamW hyperparameters are similar to those of the original Adam optimizer, including the learning rate, the decay rates of the first and second moments, and an epsilon value to avoid division by zero.

5.7 Evaluation

The model’s performance is evaluated in terms of accuracy metrics, i.e., training accuracy, validation accuracy, training loss, and validation loss. The trained Seq2Seq model (using different word embeddings) is then used to calculate the BLEU score on the 10k test data.

5.7.1 Sentence Prediction

Greedy decoding is used for correct sentence prediction in the project. Greedy decoding goes with whichever word has the highest probability at hand (does not consider what comes before that word) and might end with a sentence with a lower probability as a whole. BEAM search can be used to overcome this issue. In contrast to Greedy Search, using Beam search, the best N sequences so far are considered, and the probabilities of all the combinations of the preceding words along with the current word are used to predict the sentence.

5.7.2 Accuracy

It tells how well the model is performing in terms of correctly predicting the sentence. The accuracy can be calculated using the formulae:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

where TP = True Positive, TN = True Negative, FP = False Positive and FN = False Negative. Though accuracy might not always give the correct picture of how well the model is performing for NLP tasks. Other Evaluation techniques like BLEU score and GLEU Score gives a better understanding of the model's performance.

5.7.3 BLEU Score

BLEU (Bilingual Evaluation Understudy Score) Score is used to evaluate machine-translated text and ranges between the value 0 to 1. It uses the similarity between the translated and original sentences to calculate the score. It is a precision focused metric that evaluates the n-gram overlap of reference and predicted sentences. The matching of n-grams is position-independent. Inbuilt $nltk.translate.bleu_score$ function provided by the Natural Language Toolkit (NLTK) library has been used to calculate the BLEU for evaluating the GEC task.

5.7.4 GLEU Score

GLEU (Grammar-based Lexicalized Evaluation Metric) score is a metric for evaluating the quality of machine-translation output. It measures the similarity between the output of a machine translation system and a reference translation in terms of both grammar and vocabulary.

The GLEU score is calculated by comparing the n-grams (sequences of n consecutive words) in the machine translation output to those in the reference translation and assigning a score to each n-gram based on its exact match, presence in the reference translation or presence in the output. The score is then normalized by the total number of n-grams in the output. The GLEU score ranges from 0 to 1, with higher scores indicating better translation quality. Overall, GLEU is considered to be a more comprehensive metric than BLEU, as it takes into account both grammatical correctness and lexical diversity of the translation output. Inbuilt $nltk.translate.gleu_score$ function provided by the Natural Language Toolkit (NLTK) library has been used to calculate the GLEU for evaluating the GEC task.

6
Experiments and Analysis

The experiment code repository and trained models are available on the 172.25.0.209 server under the user id gupta92.

6.1 GEC For English

The Seq2Seq Word level Models for English have been implemented in the Phase 1 (Semester 3), whereas the Transformer based Models have been implemented in the Phase 2 (Semester 4).

6.1.1 Seq2Seq Word level Models for English

In the project's initial phase, experiments were carried out using Seq2Seq Encoder-Decoder model as discussed in Chapter 2, section 2.2.1.1 and 2.2.2.1, for English to study the model's performance, as this technique is already available. The dataset used is LANG8 with 80624 training sentences, 20156 validation sentences, and the model's performance was tested on 2000 test sentences.

Model	**Embedding**	**Training Loss**	**Validation Loss**	**BLEU Score**
Encoder Decoder	Trainable	0.5163	1.1652	0.1294
Encoder Decoder (Attention)	Trainable	0.2610	0.6519	**0.4426**

Table 6.1: Test results obtained for different models for English

These results motivated us to implement the same method for GEC in Hindi as well.

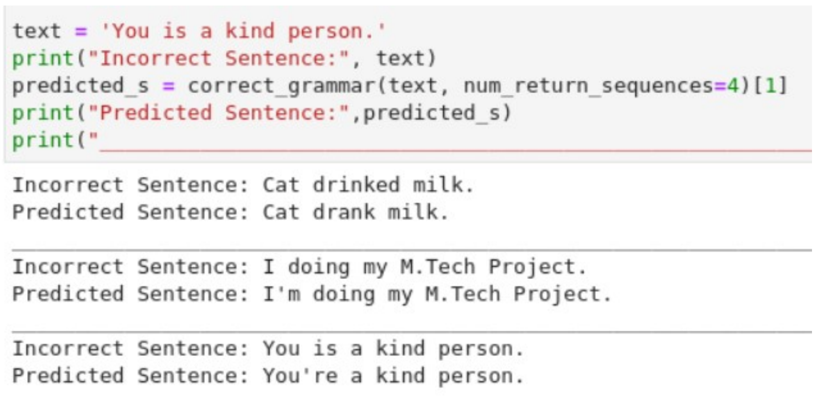
6.1.2 Transformer based Models for English

As discussed in Chapter 5, section 5.2.2, Rothe et al., in their work [22], achieved state-of-the-art results for the GEC task in four languages: English, German, Czech, and Russian, using the T5 transformer-based model. In order to understand the pipeline of model training and its performance for the GEC task in English, the model was trained for 1 epoch using 45000 training sentences and 5000 validation sentences from the C4 dataset. The T5ForConditionalGeneration, a variant of the T5 model designed explicitly for text generation tasks, was initialized with the pre-trained t5-base model using the ”t5-base” pre-trained tokenizer and was fine-tuned for the GEC task. Results are shown in Table 6.2 and Figure 6.1.

These results motivated us to implement transformer-based architecture for the for GEC in Hindi as well.

Model	Pre-trained	Tokenizer	Epoch	Training Loss	Validation Loss
T5ForConditionalGeneration	t5-base	t5-base	1	0.78800	0.655472

Table 6.2: Test results obtained for T5 transformer models for English



```
text = 'You is a kind person.'
print("Incorrect Sentence:", text)
predicted_s = correct_grammar(text, num_return_sequences=4)[1]
print("Predicted Sentence:",predicted_s)
print("_______________________________________________________
```

```
Incorrect Sentence: Cat drinked milk.
Predicted Sentence: Cat drank milk.
________________________________________________________________
Incorrect Sentence: I doing my M.Tech Project.
Predicted Sentence: I'm doing my M.Tech Project.
________________________________________________________________
Incorrect Sentence: You is a kind person.
Predicted Sentence: You're a kind person.
```

Figure 6.1: Grammatical Error Correction using T5 model for English

6.2 GEC For Hindi

Figure 6.2. shows different architectures used to train the model in order to implement the GEC task for Hindi. Among these, only Copy Augmented transformer model is used by Ankur et al. [2]; the rest of the two architectures discussed in Chapter 2, Section 2.2.1 and 2.2.2, have never been used for the GEC task in Hindi. Detail explanation of working of the architectures are given in Chapter 2, Section 2.2 and Chapter 5. The difference in the configurations of the models used is explained in Chapter 5, Section 5.4.

The Seq2Seq Word level Models have been implemented in Phase 1 (Semester 3), whereas the Transformer-based and the Copy Augmented Models have been implemented in Phase 2 (Semester 4).

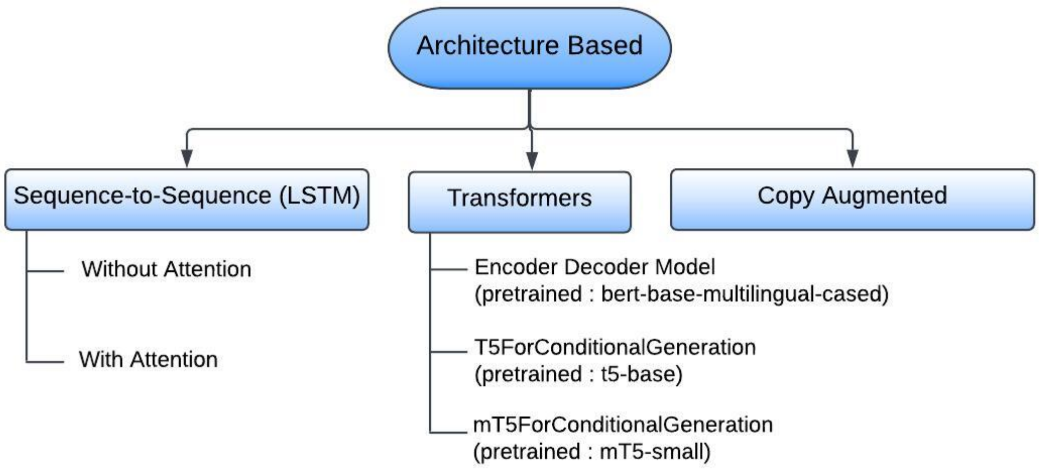


Figure 6.2: Different Architectures used to implement GEC task for Hindi

6.2.1 Exploratory Data Analysis

Exploratory Data Analysis was performed on Etoori's dataset to get a sense of the data. Duplicate data entries and unnecessary spaces are removed from the dataset. Analysis of word distribution of incorrect and correct sentences indicates that the maximum number of characters present in the input is 1823 with 410 words. Also, more than 99 percentile of the inputs in the dataset has 51 words with 248 characters. The same kind of distribution has been observed for both incorrect and correct sentences. Also, the visualization of most occurring words in the incorrect and correct sentences has been observed using the word cloud.

[a] [b]

Figure 6.3: [a] Word cloud for incorrect sentences and [b] Word cloud for corresponding correct sentences in the Etoori's dataset.

6.2.2 Seq2Seq Word level Models

We have trained the Seq2Seq Encoder Decoder model, as discussed in chapter 4, both with and without attention mechanism. The dataset generated by Etoori et al.[1], as discussed in section 2.1.2, has been used in the project to carry out the baseline experimentation and train the model. Also the different word embeddings, as discussed in the section 4.1.1, 4.1.2 and 4.1.3 are used while training the model such as Word2Vec, Glove, and Fasttext. It has been observed that the model trained using attention mechanism outperformed the model trained without attention (Results shown in Table 6.6). Also, the best performance was achieved using Fasttext word embedding over others (Results shown in Table 6.7).

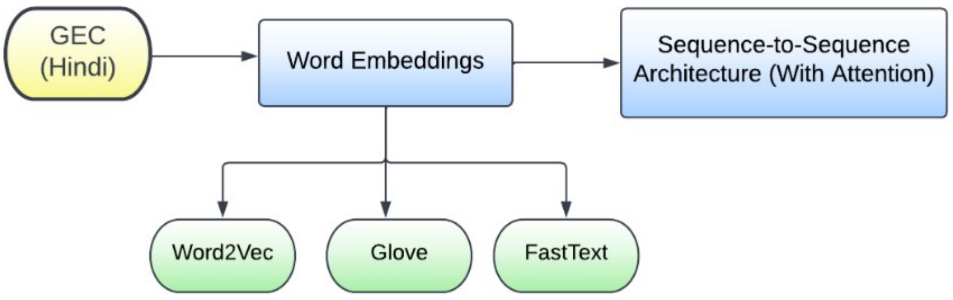
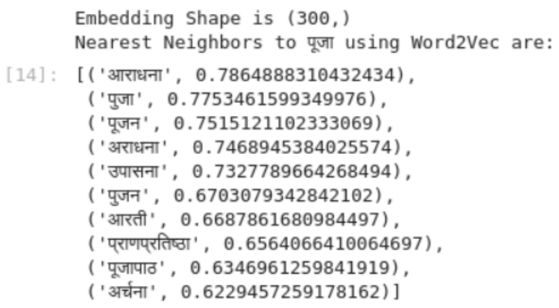


Figure 6.4: Different Word Embeddings used to train a Seq2Seq attention based model

6.2.2.1 Different Word Embedding Analysis

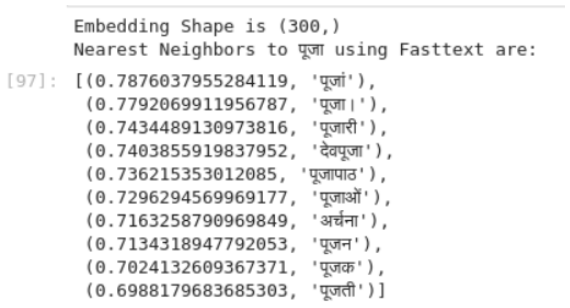
1. Common Word Predictions

The nearest common word prediction using Word2Vec and Fasttext has been shown in the Figure 6.5. As we can see, the nearest words predicted using Word2Vec are almost synonyms of each other, whereas the nearest words predicted using Fasttext are more diverse in terms of the words which can be related to the given word.



```
Embedding Shape is (300,)
Nearest Neighbors to पूजा using Word2Vec are:
[14]: [('आराधना', 0.7864888310432434),
       ('पुजा', 0.7753461599349976),
       ('पूजन', 0.7515121102333069),
       ('अराधना', 0.7468945384025574),
       ('उपासना', 0.7327789664268494),
       ('पुजन', 0.6703079342842102),
       ('आरती', 0.6687861680984497),
       ('प्राणप्रतिष्ठा', 0.6564066410064697),
       ('पूजापाठ', 0.6346961259841919),
       ('अर्चना', 0.6229457259178162)]
```

[a]



```
Embedding Shape is (300,)
Nearest Neighbors to पूजा using Fasttext are:
[97]: [(0.7876037955284119, 'पूजां'),
       (0.7792069911956787, 'पूजा।'),
       (0.7434489130973816, 'पूजारी'),
       (0.7403855919837952, 'देवपूजा'),
       (0.736215353012085, 'पूजापाठ'),
       (0.7296294569969177, 'पूजाओं'),
       (0.7163258790969849, 'अर्चना'),
       (0.7134318947792053, 'पूजन'),
       (0.7024132609367371, 'पूजक'),
       (0.6988179683685303, 'पूजती')]
```

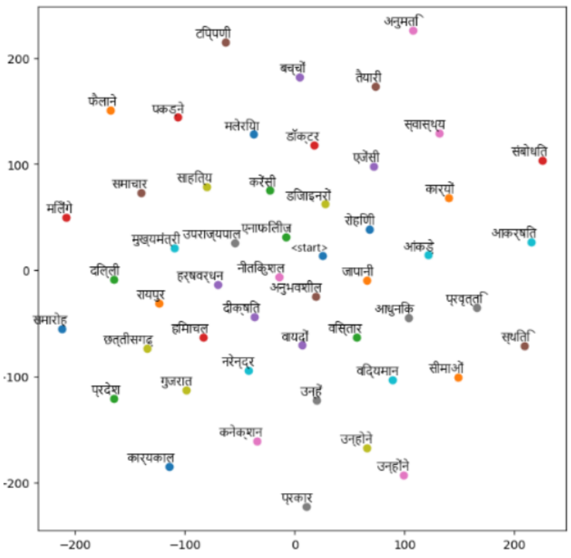
[b]

Figure 6.5: Nearest neighbor of the same Hindi word using [a] Word2Vec and [b] Fasttext

2. Visualization of Word Embeddings using tSNE

We can plot the word embeddings in a 2D space, which is helpful for checking how words are related to each other. Similar words are plotted together, and words unrelated to each other are plotted at a further distance. TSNE (t-Distributed Stochastic Neighbor Embedding) is the method to reduce the dimension of the word embeddings to 2 or 3, so they can be visualized in a 2D space. tSNE plot of different word embeddings with 50 words from the dataset vocabulary has been shown in the figure 6.6 and 6.7

[a]



[b]

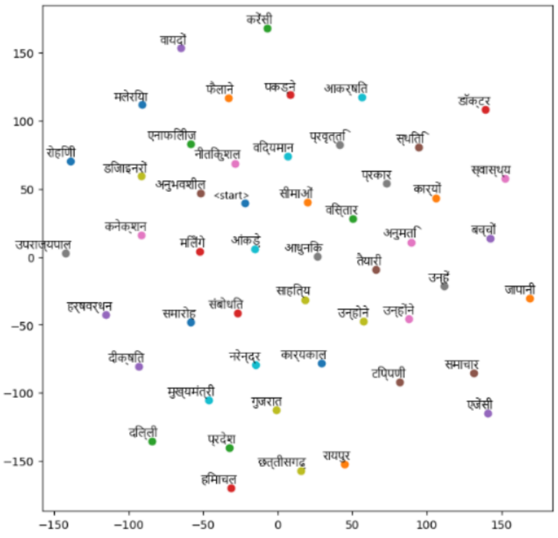


Figure 6.6: shows tSNE plot for [a]Word2Vec [b]Glove

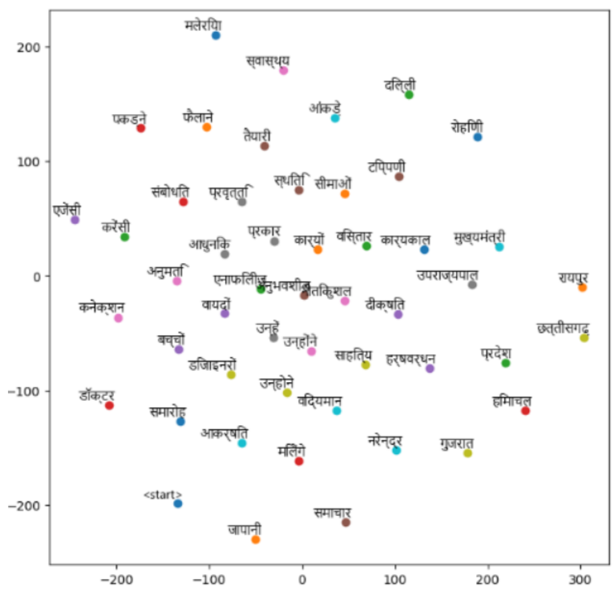


Figure 6.7: tSNE plot for Fasttext

6.2.3 Transformer Based Models

I have trained 3 different transformer-based models, as discussed in Chapter 5, Section 5.2 :

- Model 1: Encoder-Decoder Model (initialized with pre-trained bert-base-multilingual-cased encoder and decoder) with bert-base-multilingual-cased tokenizer.
- Model 2: T5ForConditionalGeneration (initialized with pre-trained t5-base) with MuRIL and RoBERTa tokenizers.
- Model 3: the multilingual variant of T5, mT5ForConditionalGeneration with MuRIL tokenizers.

The different tokenizers used are explained in Chapter 3. The difference in the configurations of these models is explained in Chapter 5, Section 5.4. All three models listed above (Models 1,2 and 3) are trained on the Etoori dataset (112K Training and 28K Validation sentences). The complete pipeline of transformer based model training is shown in the Figure 6.8. The performance was evaluated using GLEU Score for 10k sentences from the Etoori Test dataset. As per the results shown in Tables 6.9 and 6.10, the best performance is achieved using the T5ForConditionalGeneration model using the MuRIL tokenizer.

In order to compare the best performing model (T5ForConditionalGeneration) with present state-of-the-art results obtained by Ankur et al. [2], Model 2: T5ForConditionalGeneration with MuRIL Tokenizer has additionally been trained on 2.6M Artificially generated dataset by Ankur et al.[2] for GEC task in Hindi. The performance of different models based on the GLEU Score for different types of Errors for the HiWikEd Test Dataset is shown in Table 6.12.

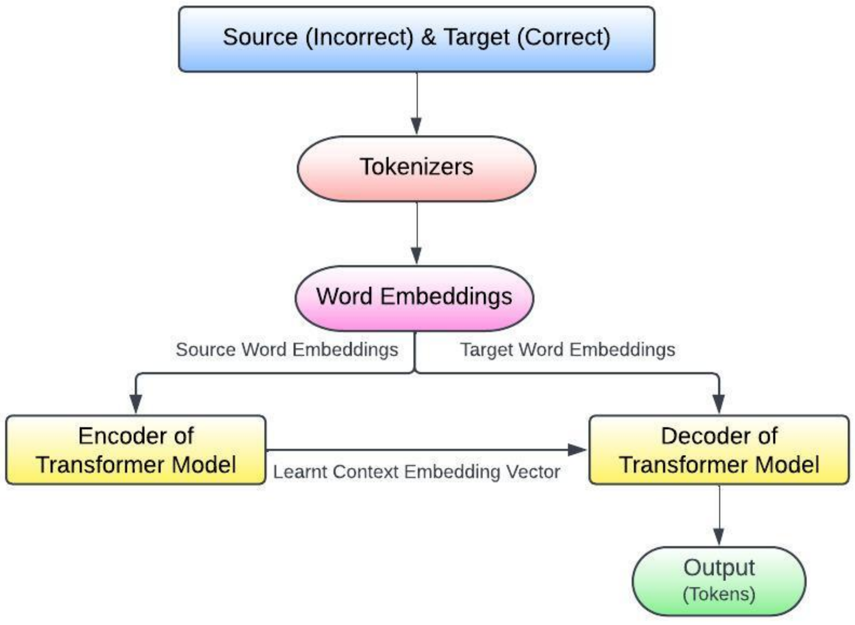

Source (Incorrect) & Target (Correct)
Tokenizers
Word Embeddings
Source Word Embeddings
Target Word Embeddings
Encoder of Transformer Model
Learnt Context Embedding Vector
Decoder of Transformer Model
Output (Tokens)

Figure 6.8: Pipeline to train a Transformer based Model

6.2.4 Copy Augmented Model

As discussed in Chapter 5, Section 5.3, in order to reproduce the results obtained by Ankur et al. [2] I was able to train the model for the Etoori Training dataset instead (112K Train and 28K Valid sentence pair), as training the model on 2.6M Artificially generated dataset was taking a lot of time. The word embedding dimension used is 512 with Vocabulary of size 73829. The model has been trained from scratch. Cuda version 10.0 (its not compatible for Cuda Version 11.1 used to train the other models) and cuDNN version 7.6.5. are used to train the model for 5 epochs.

6.2.5 Model Training

All the models are trained on the Etoori dataset (112K Train and 28K Validation Sentence pair). The train-valid-test split for each of the models is listed in Tables 6.3 and 6.4. Only the best-performing model, T5ForConditionalGeneration with MuRIL Tokenizer, is trained on an additional 2.6M Artificially generated dataset by Ankur et al. [2] in order to compare it with the present state-of-the-art results obtained by Ankur et al. [2]. Hyperparameters used to train the different models are listed in Table 6.5.

T5ForConditionalGeneration with MuRIL Tokenizer is further trained on 2.6M Artificially generated train sentenced by Ankur et.al and tested on 13K test sentences (HiWikEd).

Figures 6.9, 6.10 and 6.11 shows the plot of validation loss Vs epoch and validation accuracy Vs epoch for the Sequence-to-Sequence Encoder Decoder architecture with attention for Word2Vec, Glove and Fasttext respectively, showing a decrease in loss and increase in accuracy with epoch ensuring proper training. An early stopping criterion on validation loss has been used

Model	Embedding	Training Sentences	Valid Sentences	Test Sentences
Encoder Decoder	Trainable	112k	28k	10k
Encoder Decoder(Attention)	Trainable	112k	28k	10k
Encoder Decoder(Attention)	Word2Vec	112k	28k	10k
Encoder Decoder(Attention)	Glove	112k	28k	10k
Encoder Decoder(Attention)	Fasttext	112k	28k	10k

Table 6.3: Number of sentences (Etoori Dataset) used for training, validating and testing different Sequence-to-Sequence models for GEC in Hindi

Model	Tokenizer	Train Sentences	Valid Sentences	Test Sentences
Transformer Encoder Decoder	bert-base-multilingual-cased	112k	28k	10k
T5ForConditionalGeneration	RoBERTa-hindi-guj-san	112k	28k	10k
T5ForConditionalGeneration	MuRIL-base-cased	112k	28k	10k
mT5ForConditionalGeneration	MuRIL-base-cased	112k	28k	10k
Copy Augmented	Tokenize based on white spaces	112k	28k	10k

Table 6.4: Number of sentences (Etoori Dataset) used for training, validating and testing different Transformer based models for GEC in Hindi

Model	Seq-to-Seq (attention)	EncoderDecoder Model	T5ForConditionalGeneration	mT5ForConditionalGeneration	Copy Augmentation
Epochs	50 with early stopping	2	10	5	10
Loss	Sparse Categorical Cross Entropy	Cross Entropy	Cross Entropy	Cross Entropy	Cross Entropy
Optimizer	Adam	AdamW	AdamW	AdamW	Adam
Learning rate	1e-3	2e-5	2e-5	2e-5	1e-3
Batch Size	32	8	8	1	32
Weight decay	-	0.1	0.1	0.1	-
Dropout	-	0.1	0.1	0.1	0.2
fp16	False	True	True	False	-
Evaluation metric	Accuracy, BLEU Score, GLEU Score	Train/Valid Loss, GLEU Score	Train/Valid Loss, GLEU Score	Train/Valid Loss, GLEU Score	Train/Valid Loss, GLEU Score
cuDNN Version	8.1.1	8.1.1	8.1.1	7.6.5	7.6.5
Cuda Version	11.1	11.1	11.1	10.0	10.0

Table 6.5: Hyperparameters used to train different models

to avoid overfitting of the model. Also, plot of training loss Vs Epoch and Validation loss Vs Epoch is shown in Figures 6.12, 6.13, 6.14 and 6.15 for different transformer-based models.

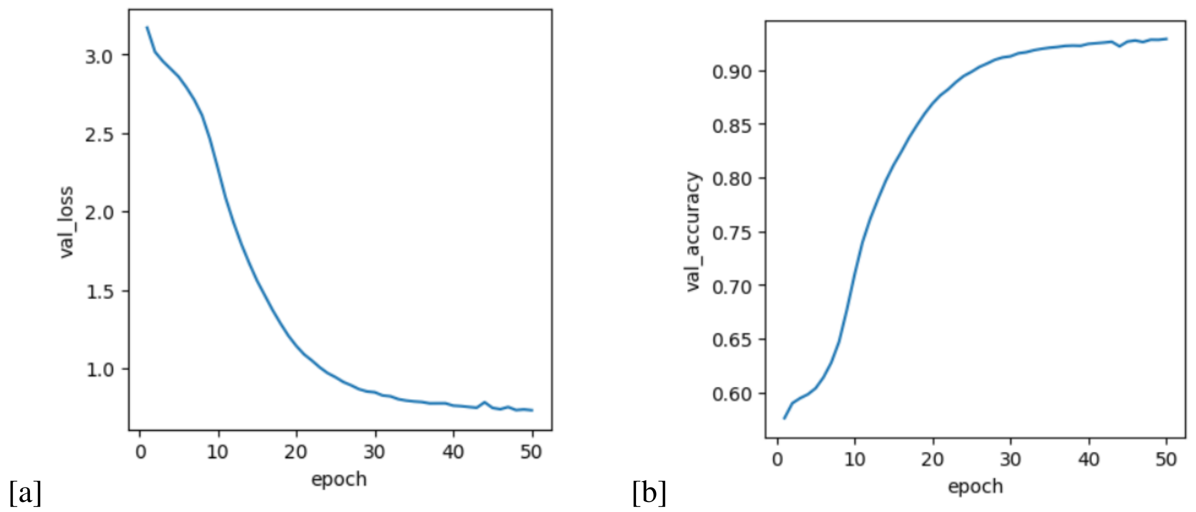
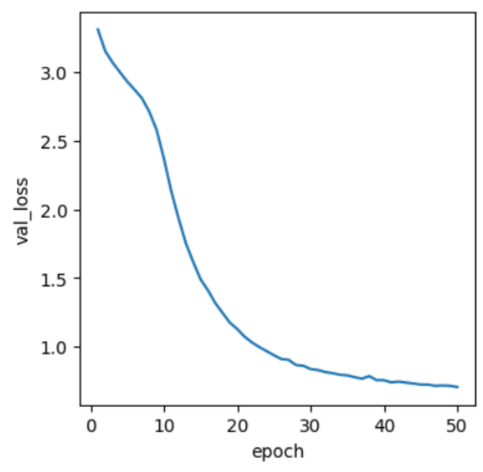
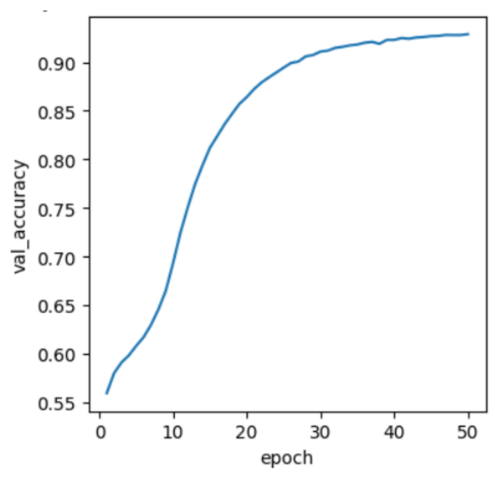


Figure 6.9: [a] Validation Loss Vs Epoch and [b] Validation Accuracy Vs Epoch for Seq2Seq Encoder Decoder Architecture with Attention using Word2Vec Word Embedding

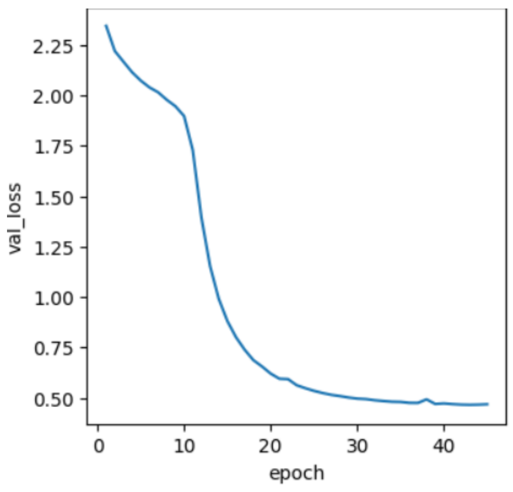


[a]

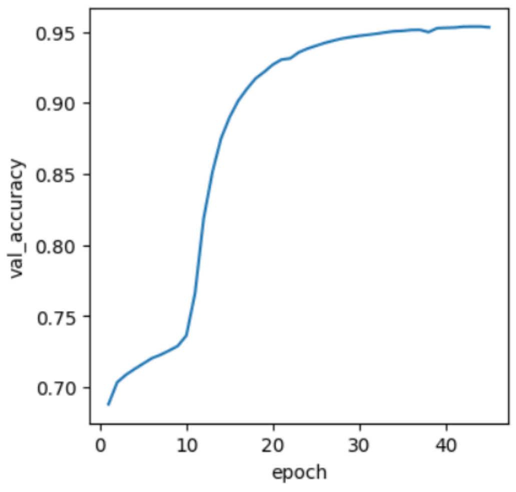


[b]

Figure 6.10: [a] Validation Loss Vs Epoch and [b] Validation Accuracy Vs Epoch for Seq2Seq Encoder Decoder Architecture with Attention using Glove Word Embedding



[a]



[b]

Figure 6.11: [a] Validation Loss Vs Epoch and [b] Validation Accuracy Vs Epoch for Seq2Seq Encoder Decoder Architecture with Attention using Fasttext Word Embedding

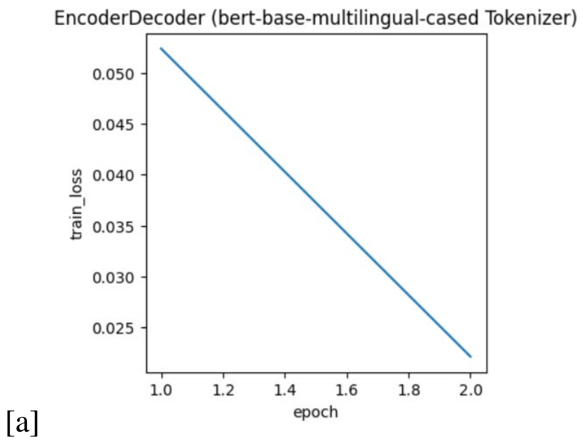
6.2.6 Model Comparison

Table 6.6 indicates that the performance of Sequence2Sequence Encoder Decoder model with attention mechanism (BLEU Score 0.76631) outperformed the Seq2Seq Encoder Decoder model without Attention (BLEU Score 0.30497), as attention mechanism allows the model to put focus on the more significant part of the input and also it helps to learn the long sentences properly.

Model	**Embedding**	**Validation Accuracy (%)**	**BLEU Score**	**Model Size (Bytes)**	**Model Parameters**	**Inference Time(ms)**
Encoder Decoder	Trainable	69.55	0.30497	255761972	63934283	177
Encoder Decoder (Attention)	Trainable	99.02	**0.76631**	331019284	82746700	186

Table 6.6: Performance of Seq2Seq Encoder Decoder model with and without attention mechanism

Table 6.7 indicates that the best performance has been achieved by the Encoder Decoder model with attention trained using Fasttext word embedding with a validation accuracy of 96.55% and GLEU Score of 0.8639 over others. Also, the results obtained by me for the Sequence-to-Sequence



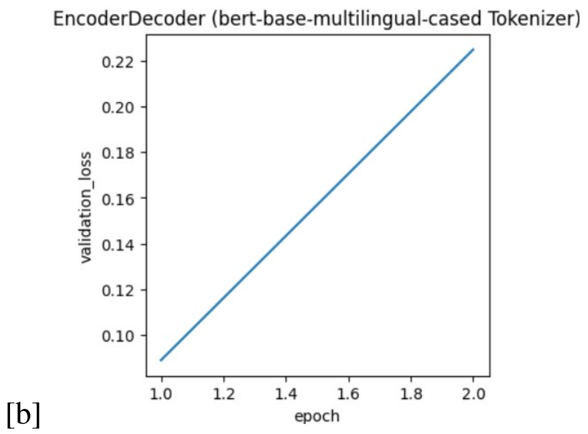
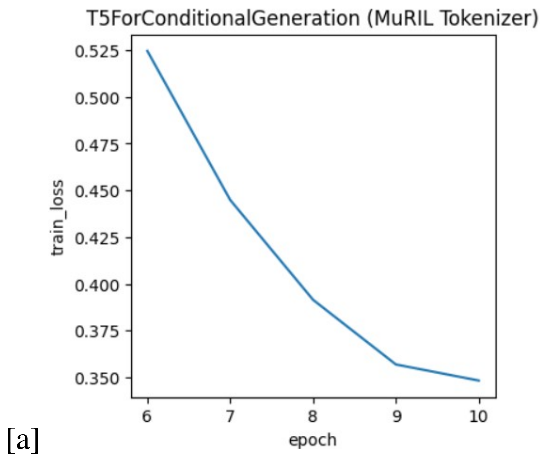


Figure 6.12: [a] Train Loss Vs Epoch and [b] Validation Loss Vs Epoch for Encoder-Decoder (pre-trained: bert-base-multilingual-cased each) transformer based Architecture with bert-base-multilingual-cased Tokenizer



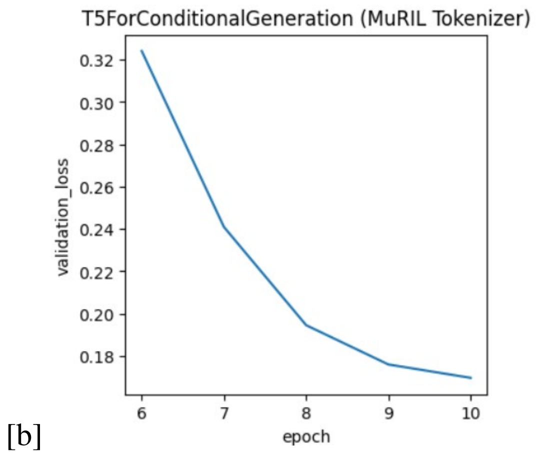


Figure 6.13: [a] Train Loss Vs Epoch and [b] Validation Loss Vs Epoch for T5ForConditionalGeneration (pre-trained: t5-base) transformer based Architecture with MuRIL Tokenizer

model with attention (Table 6.7) are comparable (exact) to the one obtained by Garg [6] in her work. Among the Transformer-based models, the result shown in Table 6.9, it was observed that all variations of the models gave better results than the Seq2Seq model, with T5ForConditionalGeneration (using MuRIL Tokenizer) giving the best result with GLEU Score of 0.92086 for Etoori (10K) Test dataset.

Model (Seq2Seq)	Embedding	Training Accuracy (%)	Validation Accuracy (%)	BLEU Score (Greedy Search)	GLEU Score
Encoder Decoder(Attention)	Trainable	99.95	99.02	0.76631	0.7856
Encoder Decoder(Attention)	Word2Vec	98.08	95.44	0.7796	0.8050
Encoder Decoder(Attention)	Glove	98.86	96.15	0.82002	0.7821
Encoder Decoder(Attention)	Fasttext	99.30	96.55	**0.8458**	**0.8639**

Table 6.7: Results obtained by me for different Seq2Seq models trained using different embeddings

Table 6.10 shows the model size, trainable parameters, and Inference Time for the different trained models used in this project. It can be observed that the size and number of trainable parameters of transformer-based models are almost 6-7 times more than the Sequence-to-Sequence

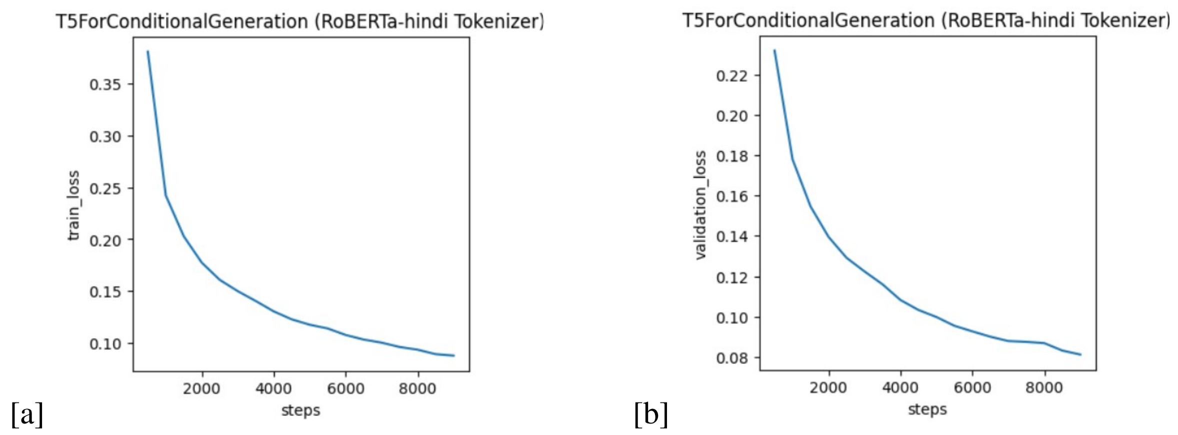


Figure 6.14: [a] Train Loss Vs Steps and [b] Validation Loss Vs Steps for T5ForConditionalGeneration (pre-trained: t5-base) transformer based Architecture with RoBERTa-hindi Tokenizer

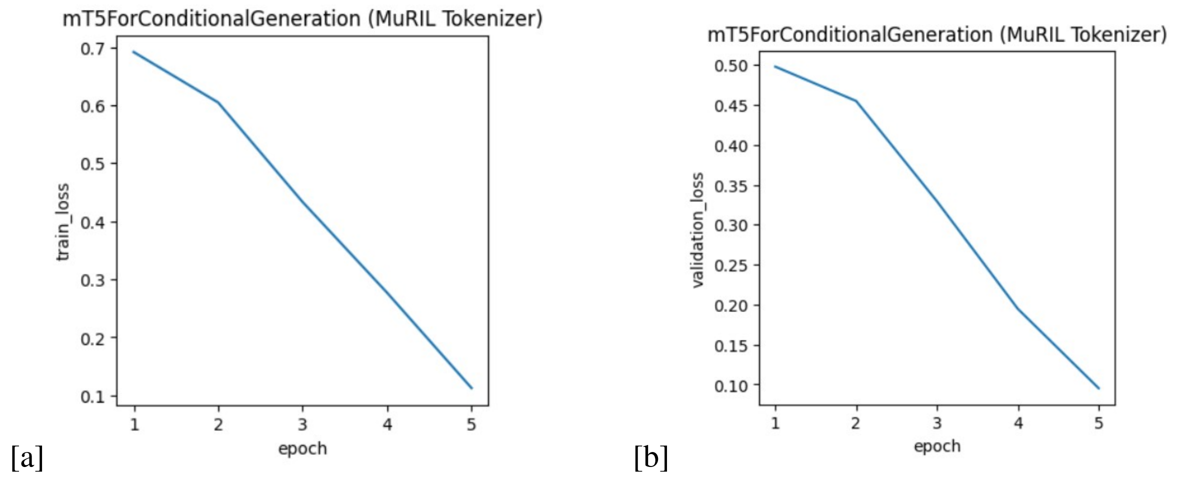


Figure 6.15: [a] Train Loss Vs Epoch and [b] Validation Loss Vs Epoch for mT5ForConditionalGeneration (pre-trained: mt5-small) transformer based Architecture with MuRIL Tokenizer

Model (Seq2Seq)	Embedding	Training Accuracy (%)	Validation Accuracy (%)	BLEU Score
Encoder Decoder(Attention)	Word2Vec	92.85	90.9	0.5790
Encoder Decoder(Attention)	Glove	98.91	96.28	0.8268
Encoder Decoder(Attention)	Fasttext	99.34	90.69	**0.8461**

Table 6.8: Results obtained by Garg[6] using Vyakaranly Seq2Seq Word level Hindi Models

Transformer Model	Pre trained	Tokenizer	Training Loss	Validation Loss	Vocab Size	GLEU Score
Encoder Decoder	bert-base-multilingual-cased	bert-base-multilingual-cased	0.32566	0.2248	119547	0.91757
T5ForConditionalGeneration	t5-base	RoBERTa-hindi-guj-san	0.08328	0.06815	30522	0.82015
T5ForConditionalGeneration	t5-base	MuRIL	0.43034	0.16969	197285	**0.92086**
mT5ForConditionalGeneration	mt5-small	MuRIL	0.11250	0.09533	250112	0.90232
Copy Augmented	-	Tokenizes based on white spaces	0.37821	0.24620	73829	0.8574

Table 6.9: Results obtained by me for different Transformer based models

based models. However, Sequence-to-Sequence-based architecture's Inference time is much better than transformer-based architecture's.

Model	Architecture	Embedding	Tokenizer	Model Size(MB)	Model Parameters	Inference Time(s)
Encoder Decoder(Attention)	Seq2seq	Trainable	indic-tokenizer	331.0192	82746700	0.200
Encoder Decoder(Attention)	Seq2seq	Word2Vec	indic-tokenizer	195.5136	48870508	0.273
Encoder Decoder(Attention)	Seq2seq	Glove	indic-tokenizer	195.5136	48870508	0.253
Encoder Decoder(Attention)	Seq2seq	Fasttext	indic-tokenizer	195.5136	48870508	0.345
Encoder Decoder	Transformer	Contextual	bert-base-multilingual-cased	1465.603	384194811	1.16
T5ForConditionalGeneration	Transformer	Contextual	RoBERTa-hindi-guj-san	845.604	221670144	1.38
mT5ForConditionalGeneration	Transformer	Contextual	MuRIL	1145.083	300176768	1.02
T5ForConditionalGeneration	Transformer	Contextual	MuRIL	1334.168	349744128	1.47
Copy Augmented	Copy Augmentation	Contextual	Tokenizes based on white spaces	1116.105	293570124	1.23

Table 6.10: Trained model size, Model Parameters and the Inference Time(ms) required for grammatical correcting one incorrect sentence.

Table 6.11 shows the performance analysis of different Models based on GLEU Scores (for evaluation) both on Etoori and HiWikEd Test Dataset. * indicates the models trained by me on 140K Etoori dataset (112K train and 28K Validation). Only T5ForConditionalGeneration is trained on an additional 2.6M Artificially generated dataset for Hindi for 1 Epoch (computation takes a very long time, nearly 35 hours for 1 epoch). Moreover, @ indicates the models implemented by Ankur et al. [2] where all models are trained on 2.6 M artificially generated dataset for Hindi.

Model	Etoori test dataset	HiWikEd test dataset
Transf$^{@}$	-	0.69
MLConv$^{@}$	-	0.73
Copy Augmented$^{* @}$	0.8574	0.80
Seq2Seq + Attention (Glove)*	0.78218	-
Seq2Seq + Attention (Word2Vec)*	0.80502	-
Seq2Seq + Attention (fastText)*	0.86394	-
Bert based Encoder Decoder Transformer*	0.91757	-
mT5ForConditionalGeneration (Transformer)*	0.90232	-
T5ForConditionalGeneration (Transformer)*	**0.92086**	**0.8107**

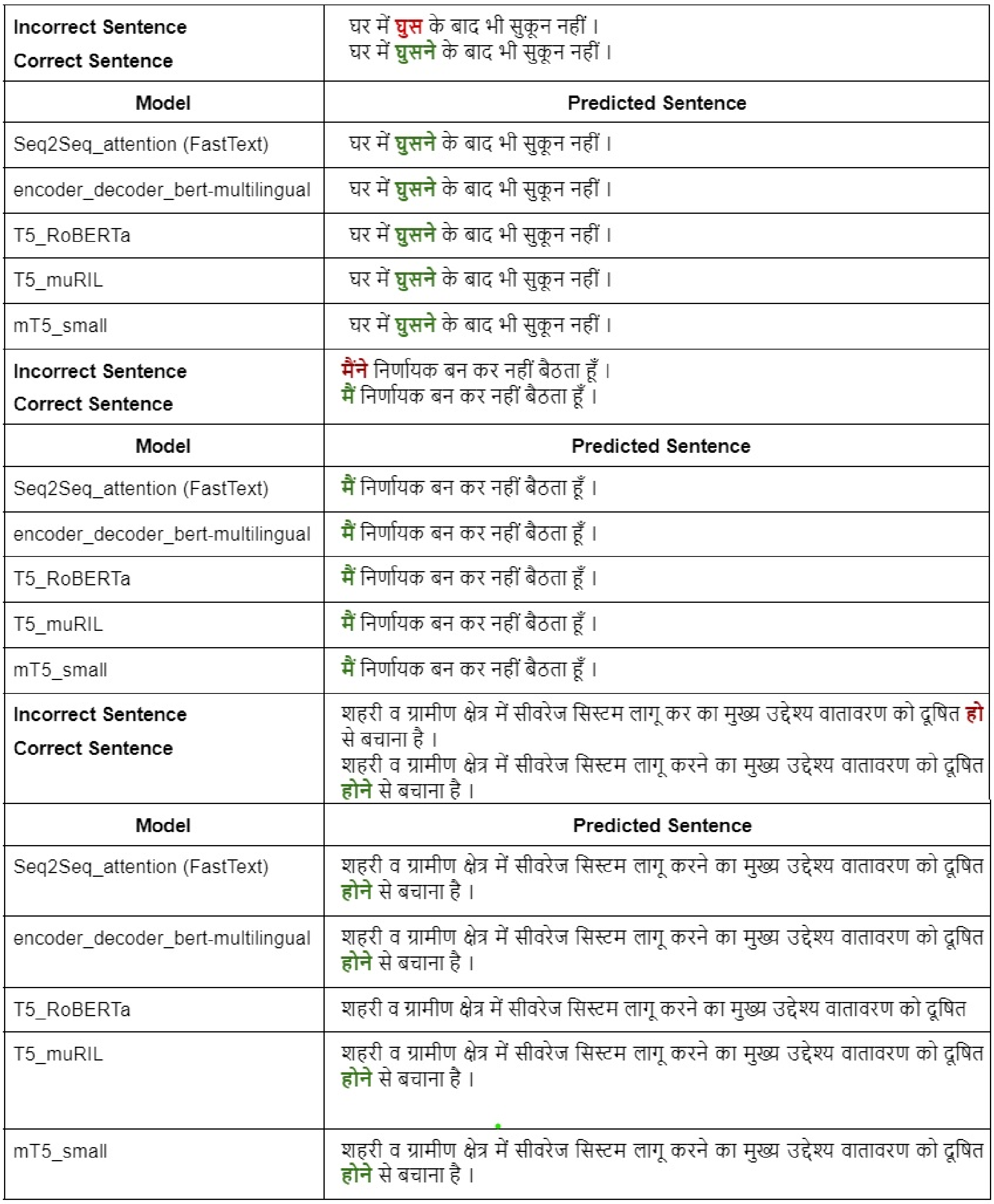
Table 6.11: Performance Analysis of different models based on GLEU Score. * : Models implemented by me ;@ : Models implemented by Ankur et al.[2])

Model	Epoch	ADP:INFL	PRON:INFL	ADJ:INFL	VERB:INFL	Full Dataset
Tranf @	1	0.62	0.67	0.57	0.79	0.69
CopyAug @	9	0.84	0.71	0.69	0.87	0.80
T5ForConditionalGeneration *	1	0.8082	**0.8095**	**0.7434**	0.8298	**0.8107**

Table 6.12: Performance Analysis of different models based on GLEU Score for different types of Errors for the HiWikEd Test Dataset. * : Models implemented by me ;@ : Models implemented by Ankur et al.[2]

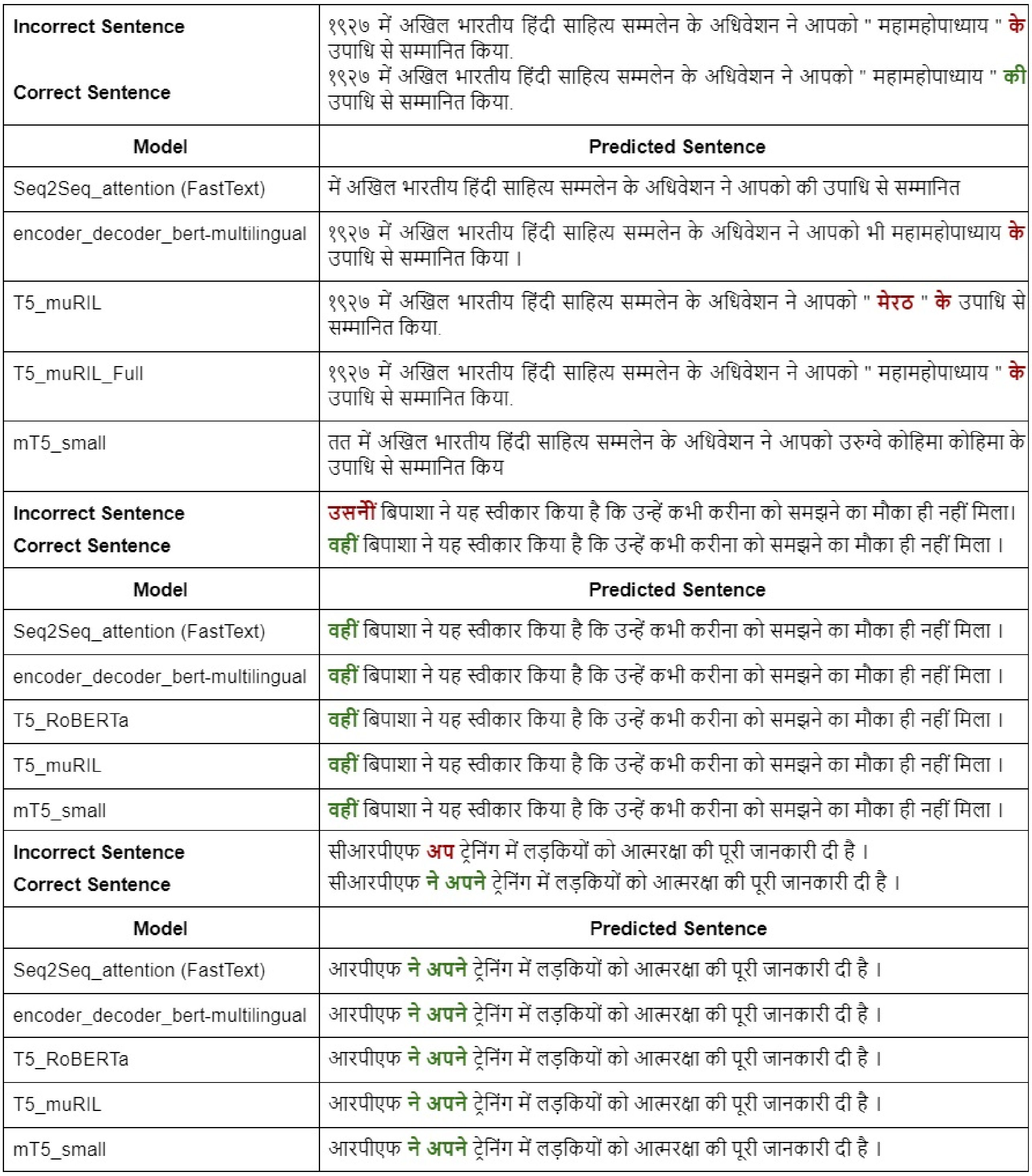
Results obtained in Table 6.12 indicates that my best performing model T5ForConditionalGeneration shows better results than the state-of-the-art results obtained by Ankur et al. [2] for GEC task in Hindi. Also, T5ForConditionalGeneration showed significant improvement for PRON:INFL and ADJ:INFL type of errors.

List of grammatical error correction for Hindi using different trained models has been shown in Figures 6.16 and 6.17.



Incorrect Sentence **Correct Sentence**	घर में **घुस** के बाद भी सुकून नहीं । घर में **घुसने** के बाद भी सुकून नहीं ।
Model	**Predicted Sentence**
Seq2Seq_attention (FastText)	घर में **घुसने** के बाद भी सुकून नहीं ।
encoder_decoder_bert-multilingual	घर में **घुसने** के बाद भी सुकून नहीं ।
T5_RoBERTa	घर में **घुसने** के बाद भी सुकून नहीं ।
T5_muRIL	घर में **घुसने** के बाद भी सुकून नहीं ।
mT5_small	घर में **घुसने** के बाद भी सुकून नहीं ।
Incorrect Sentence **Correct Sentence**	**मैंने** निर्णायक बन कर नहीं बैठता हूँ । **मैं** निर्णायक बन कर नहीं बैठता हूँ ।
Model	**Predicted Sentence**
Seq2Seq_attention (FastText)	**मैं** निर्णायक बन कर नहीं बैठता हूँ ।
encoder_decoder_bert-multilingual	**मैं** निर्णायक बन कर नहीं बैठता हूँ ।
T5_RoBERTa	**मैं** निर्णायक बन कर नहीं बैठता हूँ ।
T5_muRIL	**मैं** निर्णायक बन कर नहीं बैठता हूँ ।
mT5_small	**मैं** निर्णायक बन कर नहीं बैठता हूँ ।
Incorrect Sentence **Correct Sentence**	शहरी व ग्रामीण क्षेत्र में सीवरेज सिस्टम लागू कर का मुख्य उद्देश्य वातावरण को दूषित **हो** से बचाना है । शहरी व ग्रामीण क्षेत्र में सीवरेज सिस्टम लागू करने का मुख्य उद्देश्य वातावरण को दूषित **होने** से बचाना है ।
Model	**Predicted Sentence**
Seq2Seq_attention (FastText)	शहरी व ग्रामीण क्षेत्र में सीवरेज सिस्टम लागू करने का मुख्य उद्देश्य वातावरण को दूषित **होने** से बचाना है ।
encoder_decoder_bert-multilingual	शहरी व ग्रामीण क्षेत्र में सीवरेज सिस्टम लागू करने का मुख्य उद्देश्य वातावरण को दूषित **होने** से बचाना है ।
T5_RoBERTa	शहरी व ग्रामीण क्षेत्र में सीवरेज सिस्टम लागू करने का मुख्य उद्देश्य वातावरण को दूषित
T5_muRIL	शहरी व ग्रामीण क्षेत्र में सीवरेज सिस्टम लागू करने का मुख्य उद्देश्य वातावरण को दूषित **होने** से बचाना है ।
mT5_small	शहरी व ग्रामीण क्षेत्र में सीवरेज सिस्टम लागू करने का मुख्य उद्देश्य वातावरण को दूषित **होने** से बचाना है ।

Figure 6.16: Grammatical Error Correction for Etoori Test Data using trained models.



Incorrect Sentence **Correct Sentence**	१९२७ में अखिल भारतीय हिंदी साहित्य सम्मलेन के अधिवेशन ने आपको " महामहोपाध्याय " **के** उपाधि से सम्मानित किया. १९२७ में अखिल भारतीय हिंदी साहित्य सम्मलेन के अधिवेशन ने आपको " महामहोपाध्याय " **की** उपाधि से सम्मानित किया.
Model	**Predicted Sentence**
Seq2Seq_attention (FastText)	में अखिल भारतीय हिंदी साहित्य सम्मलेन के अधिवेशन ने आपको की उपाधि से सम्मानित
encoder_decoder_bert-multilingual	१९२७ में अखिल भारतीय हिंदी साहित्य सम्मलेन के अधिवेशन ने आपको भी महामहोपाध्याय **के** उपाधि से सम्मानित किया ।
T5_muRIL	१९२७ में अखिल भारतीय हिंदी साहित्य सम्मलेन के अधिवेशन ने आपको " **मेरठ** " **के** उपाधि से सम्मानित किया.
T5_muRIL_Full	१९२७ में अखिल भारतीय हिंदी साहित्य सम्मलेन के अधिवेशन ने आपको " महामहोपाध्याय " **के** उपाधि से सम्मानित किया.
mT5_small	तत में अखिल भारतीय हिंदी साहित्य सम्मलेन के अधिवेशन ने आपको उरुग्वे कोहिमा कोहिमा के उपाधि से सम्मानित किय
Incorrect Sentence **Correct Sentence**	**उसनीं** बिपाशा ने यह स्वीकार किया है कि उन्हें कभी करीना को समझने का मौका ही नहीं मिला। **वहीं** बिपाशा ने यह स्वीकार किया है कि उन्हें कभी करीना को समझने का मौका ही नहीं मिला ।
Model	**Predicted Sentence**
Seq2Seq_attention (FastText)	**वहीं** बिपाशा ने यह स्वीकार किया है कि उन्हें कभी करीना को समझने का मौका ही नहीं मिला ।
encoder_decoder_bert-multilingual	**वहीं** बिपाशा ने यह स्वीकार किया है कि उन्हें कभी करीना को समझने का मौका ही नहीं मिला ।
T5_RoBERTa	**वहीं** बिपाशा ने यह स्वीकार किया है कि उन्हें कभी करीना को समझने का मौका ही नहीं मिला ।
T5_muRIL	**वहीं** बिपाशा ने यह स्वीकार किया है कि उन्हें कभी करीना को समझने का मौका ही नहीं मिला ।
mT5_small	**वहीं** बिपाशा ने यह स्वीकार किया है कि उन्हें कभी करीना को समझने का मौका ही नहीं मिला ।
Incorrect Sentence **Correct Sentence**	सीआरपीएफ **अप** ट्रेनिंग में लड़कियों को आत्मरक्षा की पूरी जानकारी दी है । सीआरपीएफ **ने अपने** ट्रेनिंग में लड़कियों को आत्मरक्षा की पूरी जानकारी दी है ।
Model	**Predicted Sentence**
Seq2Seq_attention (FastText)	आरपीएफ **ने अपने** ट्रेनिंग में लड़कियों को आत्मरक्षा की पूरी जानकारी दी है ।
encoder_decoder_bert-multilingual	आरपीएफ **ने अपने** ट्रेनिंग में लड़कियों को आत्मरक्षा की पूरी जानकारी दी है ।
T5_RoBERTa	आरपीएफ **ने अपने** ट्रेनिंग में लड़कियों को आत्मरक्षा की पूरी जानकारी दी है ।
T5_muRIL	आरपीएफ **ने अपने** ट्रेनिंग में लड़कियों को आत्मरक्षा की पूरी जानकारी दी है ।
mT5_small	आरपीएफ **ने अपने** ट्रेनिंग में लड़कियों को आत्मरक्षा की पूरी जानकारी दी है ।

Figure 6.17: Grammatical Error Correction for Hindi using trained model.

7 Conclusion

This project was initially started by analyzing and carrying out several experiments using different architectures (Sequence-to-Sequence, Transformers, Copy Augmentation) to study the existing models available for GEC in English. After achieving a significant result for English (Tables: 6.1 and 6.2), the same methods were applied to the Hindi language. As discussed in Chapter 5, the architectures used were the Sequence-to-Sequence model with and without attention (using different word embeddings such as Word2Vec, Glove, and Fasttext), the Transformer-based architecture, and Copy Augmentation model.

For training the Sequence-to-Sequence model with and without attention mechanism), pre-trained models were used to generate the embeddings for Hindi, which were given to the encoder and decoder module of the architecture. All the models were trained from scratch, and I was able to reproduce the benchmark results (Table: 6.8) obtained by Garg[6] in her work for GEC in Hindi. As expected, the model trained using the attention mechanism outperformed the one without attention (Table: 6.6) as it allows to put focus on the more significant part of the input over others. Also, among different word embeddings, the best results were obtained from the model trained using the Fasttext word embedding (Table: 6.7) over the ones trained with Word2Vec and Glove.

Apart from Sequence-to-Sequence, the other architectures used for the GEC task were Transformers and Copy Augmentation model. Three different types of transformers being used in the project were Encoder Decoder model, initialized with pre-trained ”bert-base-multilingual-cased”, T5ForConditionalGeneration, initialized with pre-trained ”t5-base” and mT5ForConditionalGeneration, initialized with pre-trained ”mt5-small”. The different tokenizers (bert-base-multilingual-cased, MuRIL, and RoBERTa) were used to tokenize the source (incorrect) and target (correct) data. It was observed that all variations of the transformer model gave better results than the Sequence-to-Sequence model, with T5ForConditionalGeneration (using MuRIL) giving the best results among all (Table: 6.9). Also, results were reproduced for the Copy Augmentation model used by Ankur et al. [2] in his work for comparison.

Based on the experiments conducted, it can be concluded that the transformer-based architecture with pre-trained models outperformed the other two architectures, sequence-to-sequence, and Copy Augmentation. and perform better for grammatical error correction tasks. Also, the choice of tokenizer and word embedding can significantly impact the model’s performance. A detailed comparison of the model performance using different architectures, tokenizers, and word embeddings, along with comparing it with the present state-of-the-art model for GEC task in Hindi has been done in this project.

8
Future Work

- The Etoori [1] and HiWikEd [2] dataset have only one error per sentence. A more error inclusive dataset (dataset with multiple forms of errors and more than one error per sentence) can be prepared to train the model for better learning.
- A simple and efficient sequence tagging approach as proposed by Omelianchuk et. al. in GECToR [17] can be used for Hindi as well.
- The process can be generalized and made open source so the same architecture can be used by other low-resource Indian languages for GEC tasks.

References

[1] P. Etoori, M. Chinnakotla, and R. Mamidi, *“Automatic spelling correction for resource-scarce languages using deep learning,” in Proceedings of ACL 2018, Student Research Workshop,* 2018*, p. 146–152.*

[2] A. Sonawane, S. Kumar, Vishwakarma, B. Srivastava, and A. K. Singh, *“Generating inflectional errors for grammatical error correction in hindi,” in Proceedings of the 1st Conference of the Asia Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop,* 2020 *, p. 165–171.*

[3] C. Bryant, M. Felice, E. Andersen, and T. Briscoe, *“The bea-2019 shared task on grammatical error correction,” in In Proceedings of the 14th Workshop on Innovative Use of NLP for Building Educational Applications (BEA-2019), Association for Computational Linguistics,* 2019*, p. 52–75.*

[4] H. T. Ng, S. M. Wu, T. Briscoe, C. Hadiwinoto, R. H. Susanto, and C. Bryant, *“The conll-2014 shared task on grammatical error correction,” in In Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task (CoNLL-2014 Shared Task),* 2014.

[5] K. Saurav, K. Saunack, D. Kanojia, and P. Bhattacharyya, *“A passage to india”: Pre-trained word embeddings for indian languages,” in Proceedings of the 1st Joint SLTU and CCURL Workshop (SLTU-CCURL 2020),* 2020*, pp. 352–357.*

[6] V. Garg *”Vyakaranly”: Grammatical Error Correction for Indian Languages, Master’s Thesis, IIT Jodhpur,* 2022

[7] B. Kaur, H. Singh, *“Design and Implementation of HINSPELL -Hindi Spell Checker using Hybrid approach”,* 2015.

[8] A. Pal and A. Mustafi, *“Vartani Spellcheck – Automatic Context-Sensitive Spelling Correction of OCR-generated Hindi Text Using BERT and Levenshtein Distance”,* 2020.

[9] M. Mittal, S. Sharma and A. Sethi *” Detection and Correction of Grammatical Errors in Hindi Language Using Hybrid Approach” . International Journal of Computer Sciences and Engineering* 2019.

[10] J. Dodge, M. Sap, A. Marasovic, A. William, G. Gabriel, D. Groeneveld and M. Gardner, *”Documenting the English Colossal Clean Crawled Corpus”* 2021.

[11] I. Sutskever, O. Vinyals, and Q. V. Le, *“Sequence to Sequence Learning with Neural Networks”,* 2014.

[12] A. Vaswani, N.M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser and I. Polosukhin,(2017). *“Attention is All you Need”,* 2017.

[13] G. Tao, X. Zhang, F. Wei, and M. Zhou, *”Automatic Grammatical Error Correction for Sequence-to-sequence Text Generation: An Empirical Study”. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics,* 2019.

[14] A. Schmaltz, Y. Kim, A. M. Rush, and S. M. Shieber, *”Sentence-Level Grammatical Error Identification as Sequence-to-Sequence Correction”,* 2016.

[15] J. Devlin and M.W. Chang and K. Lee and K. Toutanova, *”BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”,* 2018.

[16] S. Khanuja, D. Bansal, S. Mehtani, S. Khosla, A. Dey, B. Gopalan, D.K. Margam, P. Aggarwal, R.T. Nagipogu, S. Dave, S. Gupta, S.C.B Gali, V. Subramanian and P. Talukdar, *”MuRIL: Multilingual Representations for Indian Languages”,* 2021.

[17] Omelianchuk, Kostiantyn, Atrasevych, Vitaliy, Chernodub, Artem, Skurzhanskyi and Oleksandr, *”GECToR: Grammatical Error Correction: Tag, Not Rewrite”,* 2020.

[18] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li and P.J. Liu, *”Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer”,* 2020.

[19] L. Xue, N. Constant, A. Roberts, M. Kale, R. Al-Rfou, A. Siddhant, A. Barua and C. Raffel, *”mT5: A massively multilingual pre-trained text-to-text transformer”,* 2021.

[20] S. Rothe, S. Narayan and A. Severyn, *”Leveraging Pre-trained Checkpoints for Sequence Generation Tasks”,* 2020.

[21] W. Zhao, L. Wang, K. Shen, R. Jia and J. Liu, *”Improving Grammatical Error Correction via Pre-Training a Copy-Augmented Architecture with Unlabeled Data”,* 2019.

[22] Rothe, Sascha, Mallinson, Jonathan, Malmi, Eric, Krause, Sebastian, Severyn and Aliaksei, *”A Simple Recipe for Multilingual Grammatical Error Correction”,* 2021.