

NLP (grammar Correction-Model)

Project submitted in the context of the course

CSCI380 Software Engineering

Prepared By

Frederic Bicandy

Leen Al Jabeen

Heba Fakh



Submitted To

Dr.Ali Sabra

LEBANESE INTERNATIONAL UNIVERSITY

SCHOOL OF ARTS & SCIENCES

SPRING 2024 – 2025

Prepared By

Submitted To

Grammar Correction Model

Chapter I - Introduction

2. Data Collection and Pre-processing

2.1 Web Scraping

2.2 Dataset Generation

Example of the dataset format

3. Model Training

Model info

4.1 Model Choice

4.2 Tokenizer

4.3. Training Configuration

4.4 DataCollatorForSeq2Seq

4.5 Training and testing

5. Results

Example Results

8. Bug Fixes and Performance Improvements

9. Conclusion

Grammar Correction Model

Chapter I - Introduction

our project is a grammar correction model which in role allows to

This project aims to build a Python-based grammar checker. It involves **web scraping**, **dataset generating**, and **training** a model. The goal is to enhance writing quality by

detecting and correcting grammar errors effectively.

2. Data Collection and Pre-processing

2.1 Web Scraping

We implemented a web scraping script to collect text data from various online sources. The purpose was to have a huge clear and well written texts so we can use it to train our model to do so we have implemented this tool using the following libraries :

Requests: which in the job is to access a public based URL found on the internet,

Beautiful Soup: is a Python library used for **web scraping** — that means extracting data from websites.

Common Use Cases:

Task	Example	
Extract headlines	From a news website	
Grab links or images	From a webpage	
Scrape tables	From Wikipedia pages	
Pull product data	From e-commerce sites (e.g., Amazon, eBay)	
Parse structured content	Like blogs, articles, or forum posts	

for our demonstration we used Wikipedia website as a base URL

Filtering the data”

for this we made sure that our text meets the validation requirements which are:

- only ASCII characters

- no urls in the scraped lines,texts

- no heavy use of symbols which reduce the sentence meaning

- reject one word scenario

- striping words with trailing white spaces

- rejecting empty or non printable type of characters

Results

our results was very good based on what we saw in the content file 90% of our scraped content is indeed readable/american standard text level with alot of paragraphs that possess real meaning



very small portion of our scraped content

‘Meanwhile - let's not forget that the security measures that they put in place worked! Kudos for the security guard that stopped the first guy before he got into the stadium. Lucky the guy didn't just detonate right there. Grim reminder that these people (that we sometimes deride as "rent-a-cops") are none the less putting their lives in danger doing their job protecting us. This could have been so much worse.

Most systems available today use a single database. This is accessed by many users by using a distributed file system. There are few products which take Ted Nelson's idea of a wide "docuverse" literally by allowing links between nodes in different databases. In order to do this, some standardisation would be necessary. However, at the standardisation workshop, the emphasis was on standardisation of the format for exchangeable media, nor for networking. This is prompted by the strong push toward publishing of hypermedia information, for example on optical disk. There seems to be a general consensus about the abstract data model which a hypertext system should use.’

2.2 Dataset Generation

next up in this chapter we will talk about the creation of our dataset, so for this we need to understand what kind of dataset we need to solve our problem:

let us imagine our model as a small kid which we in our role as grown up we need to teach him about a new language for this we might doe the following

1. either teach him English step by step or
2. just talk to him in English and he will figure it out over after tons of speech and commands and verbs given to him, our boy will eventually be able to speak and understand the language

that's what we need for our problem to understand grammar errors and fix them, but our dataset is a web Scraped content which is 98% clear and meaningful English words with 2% chance for human errors or any grammar mistakes, hence we introduce something called noise generation

our dataset tool will read the scraped content line by line and generate a copy of each sentence by creating a error which the copied sentence for us to feed later on our model 2 things the 2 sentences identical one which have errors marked as `input` and the other is the original sentence marked as `target`

For this solution we use the following libraries:

NLTK: To enable the synonym replacement process in data augmentation by providing accurate word relationships and context.

A powerful Python library for working with human language data (text).
Used for **tokenization**, **stemming**, **POS tagging**, and many other basic NLP tasks.

nlpaug.augmenter.word: To generate grammatically incorrect or altered versions of correct sentences, forming synthetic pairs for grammar correction training.

Example of the dataset format

```
{  
  "input": "she go store yesterday",  
  "target": "She went to the store yesterday."  
}
```

3.Model Training









for our model train we initialised the necessary by reading our dataset and splitting the data into 90%,10% between training and testing (validation)

Model info

4.1 Model Choice

We used the `T5ForConditionalGeneration` model from Hugging Face Transformers.

T5Tokenizer vs. BertTokenizer

Feature	T5Tokenizer	BertTokenizer
 Model type	T5 (Text-to-Text Transfer Transformer)	BERT (Bidirectional Encoder Representations from Transformers)
 Architecture	Encoder–Decoder (Seq2Seq)	Encoder-only
 Input Format	Text-to-text (e.g. "grammar: She go store")	Sentence classification, token classification
 Tokenization type	SentencePiece (uses subword units)	WordPiece (also subword-based)
 Trained tasks	Multi-task (summarization, translation, GEC, QA, etc.)	Masked Language Modeling + Next Sentence Prediction
 Special tokens	Few special tokens (<code><pad></code> , <code><unk></code> , <code><eos></code> , etc.)	Uses <code>[CLS]</code> , <code>[SEP]</code> , <code>[MASK]</code> , etc.
 Case sensitivity	Lowercased by default (T5 is uncased)	Comes in cased and uncased versions
 Output	Encodes inputs and labels for generation tasks	Encodes inputs for classification or MLM tasks

4.2 Tokenizer

The `T5Tokenizer` was loaded from a pre-trained checkpoint or a previously trained custom model which.

4.3. Training Configuration

Training was done using the Hugging Face `Trainer` class. We used the following configuration:

- `per_device_train_batch_size=8`
- `num_train_epochs=1`
- `eval_strategy="steps"`
- `eval_steps=50`
- `save_steps=50`
- `max_steps=500` (customized for performance and runtime constraints)

Training was executed on Google Colab using GPU runtime. TPU was tested but not used due to complications.

4.4 DataCollatorForSeq2Seq

When you're training a model, your input sentences often have **different lengths**. This is a problem, because deep learning models usually expect input tensors to be the **same shape** in a batch.

hence we used Seq2Seq to not complicate our training code````````````````

1. Dynamic Padding:

- Automatically pads inputs and labels in each batch **to the longest sequence in that batch**, instead of padding everything globally to the max length across the whole dataset.
- This saves memory and speeds up training.

2. Shifts Decoder Input IDs:

- For models like T5 or BART that use an encoder-decoder structure, this collator takes care of **shifting the decoder inputs** appropriately (for teacher forcing during training).

3. Handles Attention Masks:

- Automatically creates the right attention masks for padded tokens so the model doesn't "pay attention" to padding.

4.5 Training and testing

the following arguments are responsible to start training our model and testing

```
trainer = Trainer(  
    model=model,  
    args=training_args,  
    train_dataset=tokenized_dataset["train"],  
    eval_dataset=tokenized_dataset["test"],  
    tokenizer=tokenizer,  
    data_collator=data_collator,  
)
```

5.Results

We evaluated the trained model by feeding it several incorrect input sentences. Outputs were compared against those from the pre-trained model `vennify/t5-base-grammar-correction`.

Example Results

Custom Model

Before: She go store yesterday
After: Grammatical: She went to the store yesterday

Before: In moder day pepols...
After: In moderday pepols often in the past

Pre-trained Model

Before: She go store yesterday
After: She went to the store yesterday

Before: In moder day pepols...
After: In modern times, politicians of the past

The pre-trained model produced more refined corrections, while the custom model captured the core grammatical issues but occasionally repeated outputs or showed lexical artifacts.

8. Bug Fixes and Performance Improvements

- Output duplication was fixed by avoiding redundant print statements.
- GPU runtime ensured training finished within 1 hour instead of 3+.
- Batch size and `max_steps` were tuned for efficiency.
- web scraper was scraping all kind of text including non American Standard and how we managed to filter them
- dataset noise creation had been much simpler we used `nlpaugment.word` before to swap some adj which was not so efficient for the model generalization

9. Conclusion

The project demonstrated how to fine-tune a transformer model for grammar correction using custom and synthetic datasets. While the fine-tuned model performed reasonably well, the pre-trained model still had superior output quality. Future improvements may include:

- Enlarging the training dataset
- Improving synthetic data generation
- Using larger pre-trained checkpoints

End of Report