Grammar Correction by fine tuning BERT

by

Vijaiey Anand (MST03-0024)

Submitted to Scifor Technologies

Meta
Scifor Technologies

Script. Sculpt. Socialize

UNDER GUIDIANCE OF

Urooj Khan

TABLE OF CONTENTS

Abstract

This report presents the development and fine-tuning of a Bidirectional Encoder Representations from Transformers (BERT) model for the task of grammatical error correction. The objective of this project is to leverage state-of-the-art natural language processing techniques to automatically correct grammatical errors in written text. The study involves preprocessing a dataset containing pairs of ungrammatical sentences and their corrected versions, fine-tuning the pre-trained BERT model on this dataset, and evaluating its performance on a separate validation set. The methodology begins with the loading and preprocessing of the dataset, which consists of ungrammatical statements labeled with their corresponding standard English corrections. The preprocessed data is then used to fine-tune the BERT model, a deep learning architecture known for its ability to capture contextual information in text. During fine-tuning, the model learns to identify and correct grammatical errors by adjusting its parameters based on the provided training examples. Following fine-tuning, the performance of the model is evaluated on a validation dataset to assess its ability to generalize to unseen data. Metrics such as accuracy, precision, and recall are used to measure the model's effectiveness in correcting different types of grammatical errors. Additionally, qualitative analysis is conducted to examine the model's behavior and identify areas for improvement. The results indicate that the fine-tuned BERT model achieves promising performance in grammatical error correction, demonstrating its ability to accurately identify and rectify various types of errors. The model shows potential for real-world applications in improving the quality of written text, such as in automated proofreading systems or language learning platforms. Overall, this research contributes to the advancement of natural language processing techniques for text correction tasks and provides insights into the capabilities and limitations of state-of-the-art language models like BERT in addressing grammatical errors.

Introduction

In the realm of natural language processing (NLP), the task of grammatical error correction plays a crucial role in improving the quality and clarity of written communication. With the proliferation of digital content across various platforms, the need for automated tools to detect and rectify grammatical errors has become increasingly prominent. This introduction sets the stage for exploring the development and fine-tuning of a Bidirectional Encoder Representations from Transformers (BERT) model for grammatical error correction.

Effective communication relies on the proper use of language, including correct grammar, punctuation, and syntax. However, errors in written text are common, whether due to typographical mistakes, grammatical inaccuracies, or syntactic inconsistencies. These errors can hinder comprehension, detract from the professionalism of written content, and undermine the credibility of the author.

Natural language processing (NLP) techniques offer a promising approach to automating the process of grammatical error correction. By leveraging machine learning models trained on large text corpora, NLP systems can analyze and understand the structure and semantics of sentences, enabling them to identify and rectify grammatical errors with high accuracy.

BERT, a pre-trained transformer-based language model developed by Google AI, has emerged as a powerful tool for a wide range of NLP tasks, including grammatical error correction. Unlike traditional models that process text sequentially, BERT utilizes bidirectional attention mechanisms to capture contextual information from both left and right contexts, allowing it to better understand the meaning and nuances of language.

Objective

The primary objective of this study is to explore the effectiveness of fine-tuning a pre-trained BERT model for grammatical error correction. Specifically, the study aims to:

- Preprocess a dataset of ungrammatical sentences and their corrected sentences.
- Fine-tune the BERT model on the dataset to learn to identify and correct grammatical errors.
- Evaluate the performance of the fine-tuned model on a separate validation set to assess its effectiveness in real-world scenarios.

Technology Used

Programming Language:

Python

Deep Learning Framework:

PyTorch

Libraries:

Transformers: Library for natural language processing tasks.

Pandas: Data manipulation library for handling datasets.

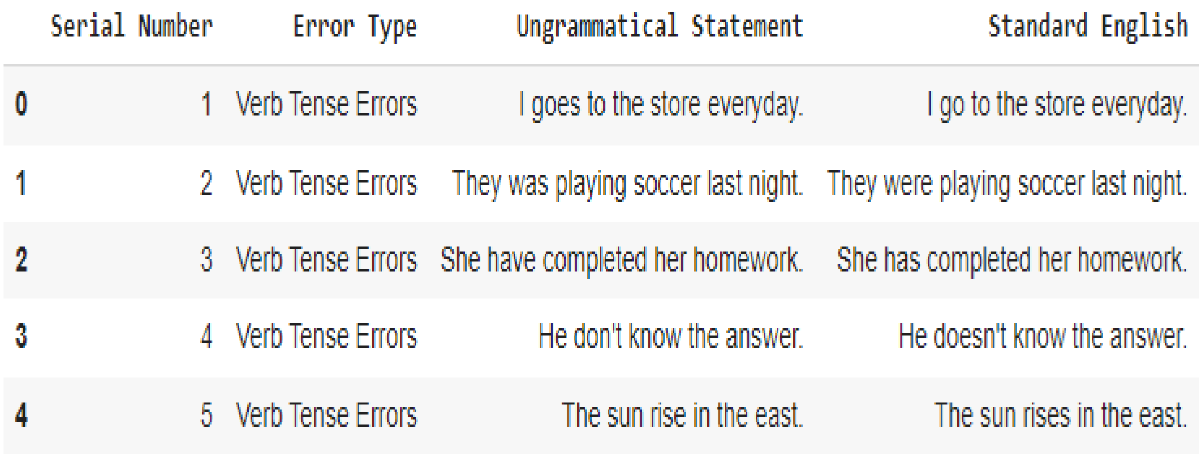
Streamlit: Python library for building interactive web applications.

Model Fine-tuning:

BERT: Pre-trained language model for contextualized word embeddings.

Dataset Information

Grammar correction dataset contains 2018 datapoints of possible error in English grammar and grammatically correct sentence.



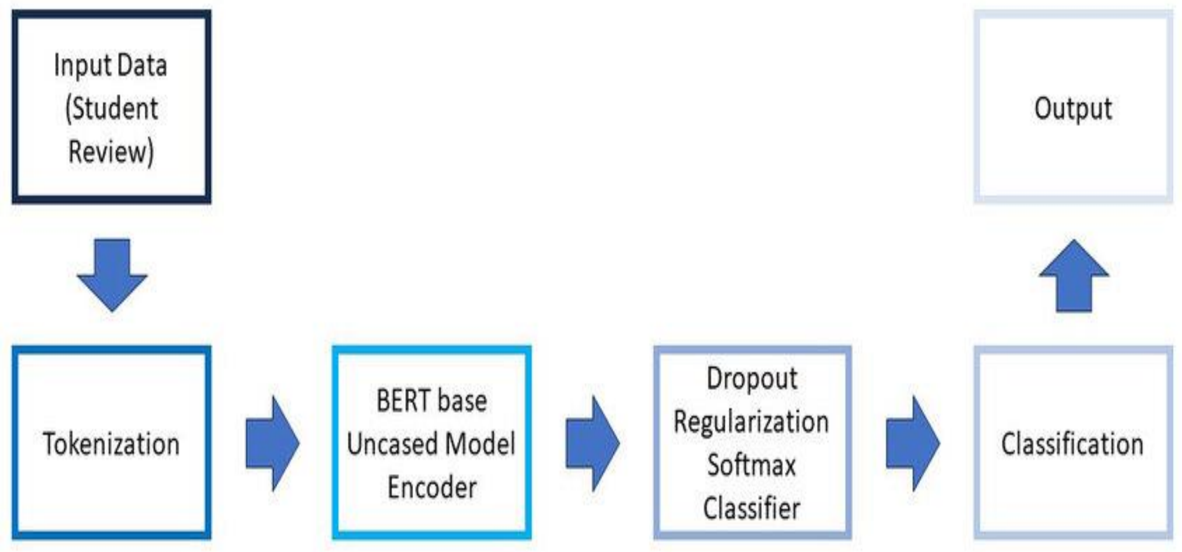
	Serial Number	Error Type	Ungrammatical Statement	Standard English
0	1	Verb Tense Errors	I goes to the store everyday.	I go to the store everyday.
1	2	Verb Tense Errors	They was playing soccer last night.	They were playing soccer last night.
2	3	Verb Tense Errors	She have completed her homework.	She has completed her homework.
3	4	Verb Tense Errors	He don't know the answer.	He doesn't know the answer.
4	5	Verb Tense Errors	The sun rise in the east.	The sun rises in the east.

Since Serial Number and Error Type doesn’t add any value to the model training, using only Ungrammatical Statement and Standard English.

BERT Architecture

The BERT (Bidirectional Encoder Representations from Transformers) architecture used in the code is a powerful pre-trained model developed by Google. It utilizes a transformer-based architecture to encode contextual information from input text, making it suitable for a wide range of natural language processing tasks, including grammatical error correction.

BERT-Base, Uncased: 12-layer, 768-hidden, 12-heads, 110M parameters


Input Data (Student Review)
Output
Tokenization
BERT base Uncased Model Encoder
Dropout Regularization Softmax Classifier
Classification

Transformer Architecture:

BERT is built upon the transformer architecture, which consists of an encoder and a decoder. However, BERT is an encoder-only model, meaning it only utilizes the encoder part of the transformer.

Tokenization:

BERT employs WordPiece tokenization, which breaks down input text into subwords or tokens. This enables BERT to handle out-of-vocabulary words and capture more fine-grained linguistic information.

Input Representation:

BERT accepts input sequences of fixed length, typically with a maximum token length. Each token in the input sequence is mapped to an embedding vector, which represents the semantic meaning of the token.

Attention Mechanism:

BERT utilizes self-attention mechanisms to capture dependencies between different tokens in the input sequence. This allows the model to consider the contextual information from both left and right contexts when encoding each token.

Transformer Blocks:

The core building blocks of BERT are transformer blocks, which consist of multiple layers of multi-head self-attention and feedforward neural networks. Each transformer block processes the input sequence in parallel and updates the representations of each token.

Pre-training:

BERT is pre-trained on large corpora of text data using unsupervised learning objectives, such as masked language modeling and next sentence prediction. This pre-training process enables BERT to learn rich contextual representations of words and sentences.

Fine-tuning:

After pre-training, BERT can be fine-tuned on specific downstream tasks, such as grammatical error correction. During fine-tuning, the parameters of the pre-trained BERT model are updated using supervised learning with task-specific labeled data.

Output:

In the context of grammatical error correction, the output of BERT consists of probabilities or logits for each token in the input sequence, indicating the likelihood of each token being corrected.

Methodology

1. Import Libraries , Load Pre-trained BERT Model and Tokenizer:

- Import libraries such as torch, transformers and pandas.
- Load the pre-trained BERT model (BertForMaskedLM) and tokenizer (BertTokenizer) from the transformers library.

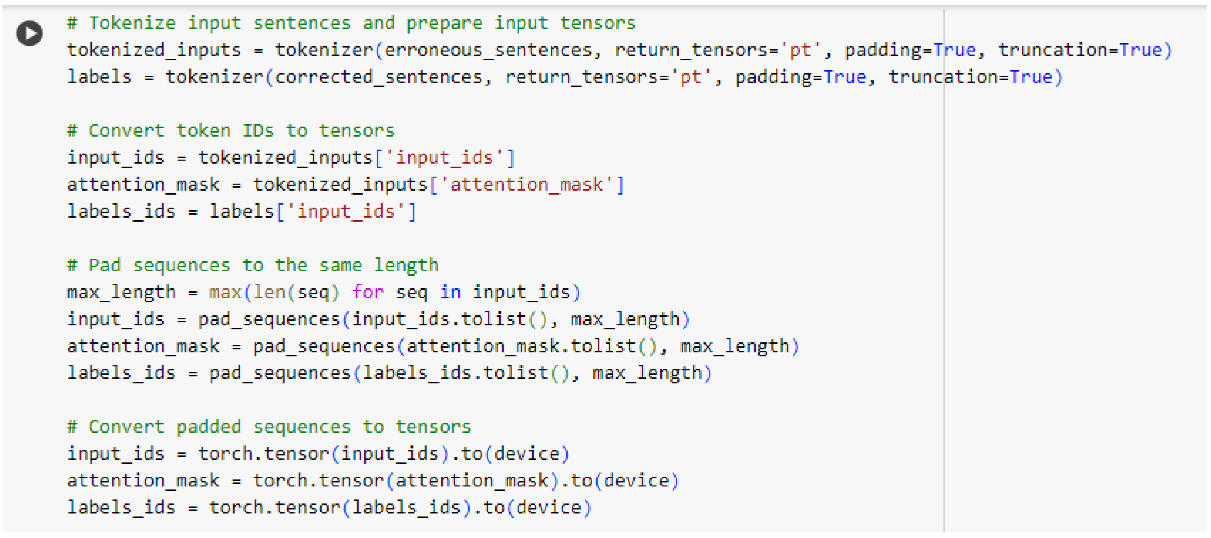
```
import torch
from transformers import BertTokenizer, BertForMaskedLM
from torch.utils.data import DataLoader, TensorDataset, random_split
from tqdm import tqdm
import pandas as pd
```

```
# Load pre-trained BERT tokenizer and model
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForMaskedLM.from_pretrained('bert-base-uncased')
```

2. Load and Preprocess Data:

- Load the data from the CSV file containing error and corrected sentences.
- Tokenize the input sentences and the corrected sentences using the BERT tokenizer.

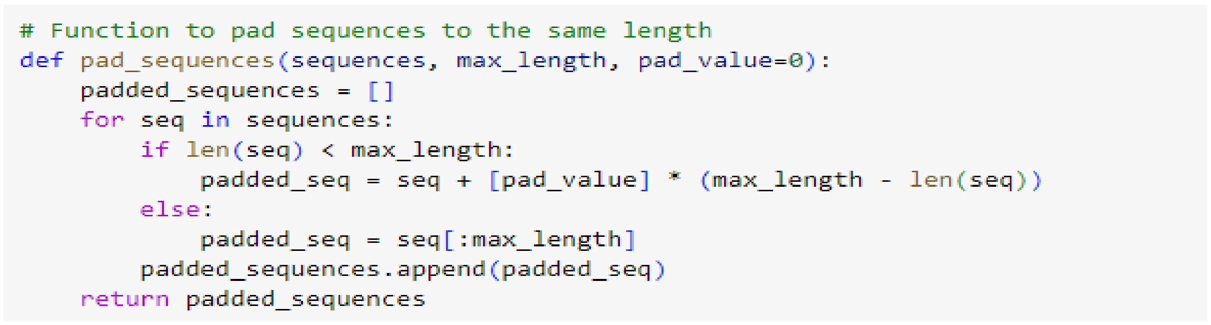
```
# Load data from CSV file
data = pd.read_csv('/content/Grammar Correction.csv')
# Extract erroneous and corrected sentences
erroneous_sentences = data['Ungrammatical Statement'].tolist()
corrected_sentences = data['Standard English'].tolist()
```



```
# Tokenize input sentences and prepare input tensors
tokenized_inputs = tokenizer(erroneous_sentences, return_tensors='pt', padding=True, truncation=True)
labels = tokenizer(corrected_sentences, return_tensors='pt', padding=True, truncation=True)

# Convert token IDs to tensors
input_ids = tokenized_inputs['input_ids']
attention_mask = tokenized_inputs['attention_mask']
labels_ids = labels['input_ids']

# Pad sequences to the same length
max_length = max(len(seq) for seq in input_ids)
input_ids = pad_sequences(input_ids.tolist(), max_length)
attention_mask = pad_sequences(attention_mask.tolist(), max_length)
labels_ids = pad_sequences(labels_ids.tolist(), max_length)

# Convert padded sequences to tensors
input_ids = torch.tensor(input_ids).to(device)
attention_mask = torch.tensor(attention_mask).to(device)
labels_ids = torch.tensor(labels_ids).to(device)
```

3. Pad Sequences to the Same Length:

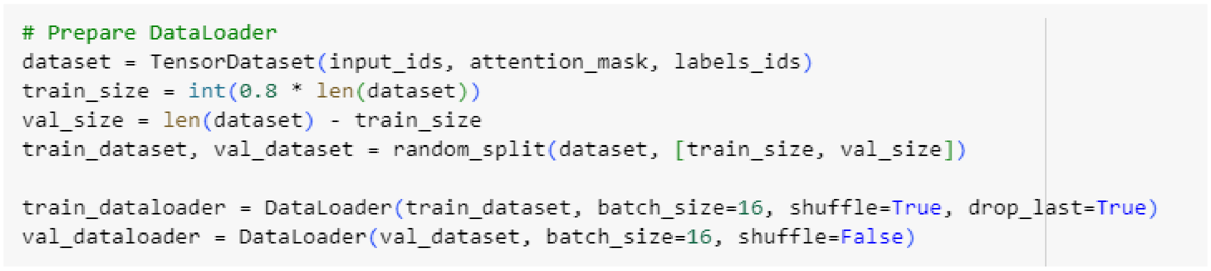
- Pad the tokenized input sequences and label sequences to the same length to create tensors.



```python
# Function to pad sequences to the same length
def pad_sequences(sequences, max_length, pad_value=0):
    padded_sequences = []
    for seq in sequences:
        if len(seq) < max_length:
            padded_seq = seq + [pad_value] * (max_length - len(seq))
        else:
            padded_seq = seq[:max_length]
        padded_sequences.append(padded_seq)
    return padded_sequences
```

4. Create DataLoader for Training and Validation:

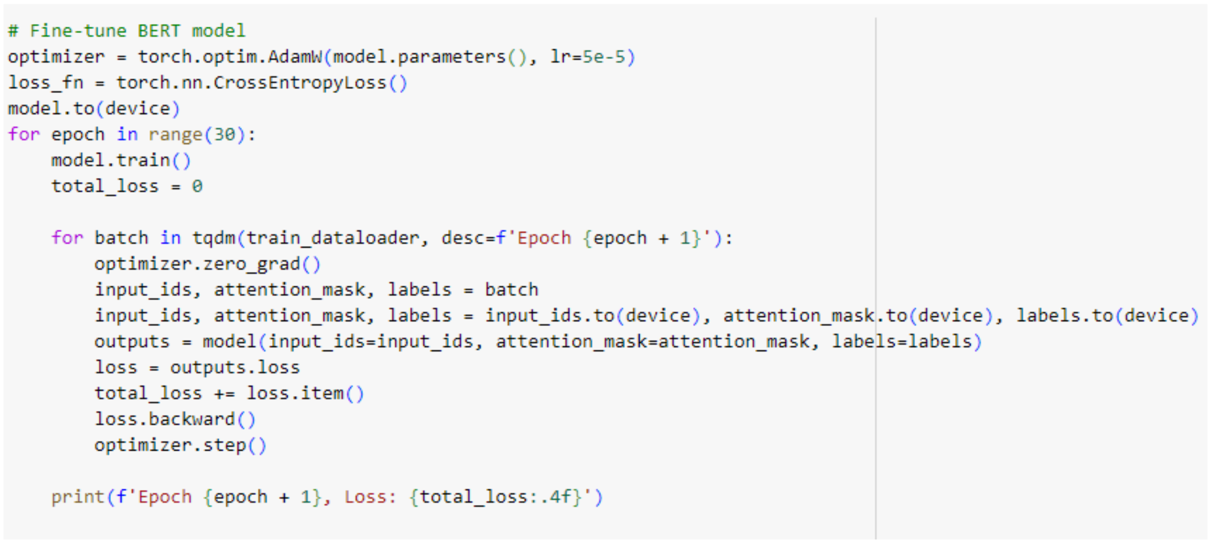
- Create PyTorch DataLoader objects for training and validation datasets.



```python
# Prepare DataLoader
dataset = TensorDataset(input_ids, attention_mask, labels_ids)
train_size = int(0.8 * len(dataset))
val_size = len(dataset) - train_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

train_dataloader = DataLoader(train_dataset, batch_size=16, shuffle=True, drop_last=True)
val_dataloader = DataLoader(val_dataset, batch_size=16, shuffle=False)
```

5. Fine-tune BERT Model:

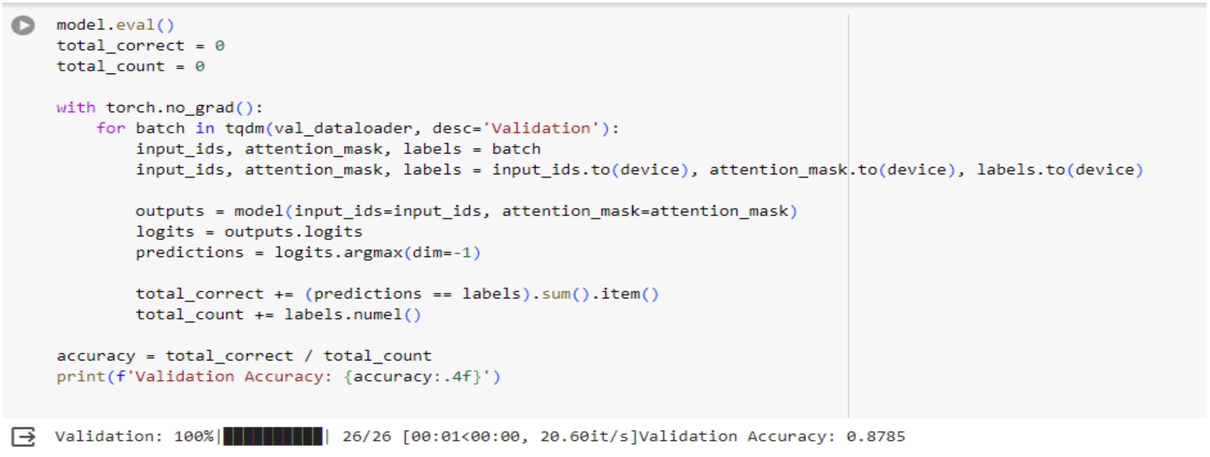
- Fine-tune the BERT model on the training dataset using the masked language modeling objective.
- Use the Adam optimizer with a learning rate of 5e-5 and cross-entropy loss function.



```python
# Fine-tune BERT model
optimizer = torch.optim.AdamW(model.parameters(), lr=5e-5)
loss_fn = torch.nn.CrossEntropyLoss()
model.to(device)
for epoch in range(30):
    model.train()
    total_loss = 0

    for batch in tqdm(train_dataloader, desc=f'Epoch {epoch + 1}'):
        optimizer.zero_grad()
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)
        outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        total_loss += loss.item()
        loss.backward()
        optimizer.step()

    print(f'Epoch {epoch + 1}, Loss: {total_loss:.4f}')
```

6. Evaluate the Fine-tuned Model:

- Evaluate the fine-tuned model on the validation dataset to measure its performance.

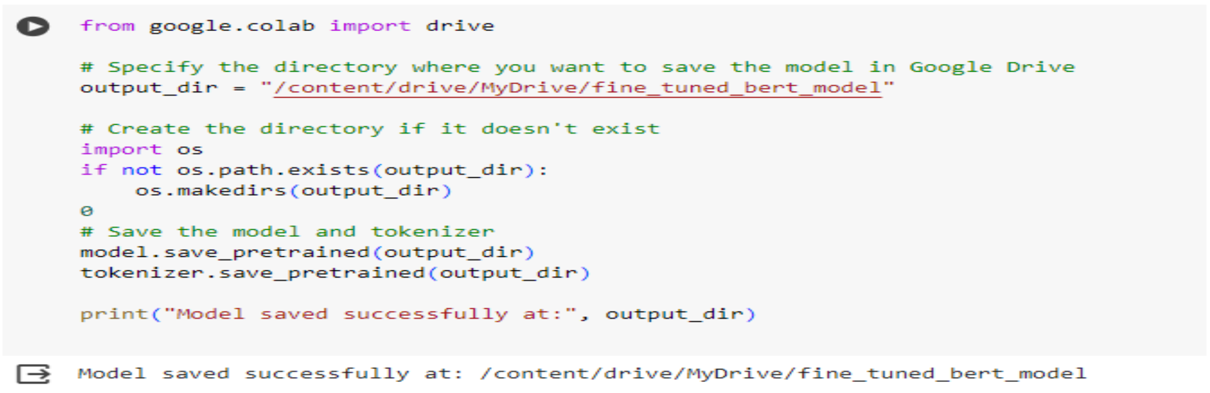


```
model.eval()
total_correct = 0
total_count = 0

with torch.no_grad():
    for batch in tqdm(val_dataloader, desc='Validation'):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)

        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        predictions = logits.argmax(dim=-1)

        total_correct += (predictions == labels).sum().item()
        total_count += labels.numel()

accuracy = total_correct / total_count
print(f'Validation Accuracy: {accuracy:.4f}')
```

```
Validation: 100%|██████████| 26/26 [00:01<00:00, 20.60it/s]Validation Accuracy: 0.8785
```

7. Save the Fine-tuned Model:

- Save the fine-tuned model and tokenizer to the specified directory.

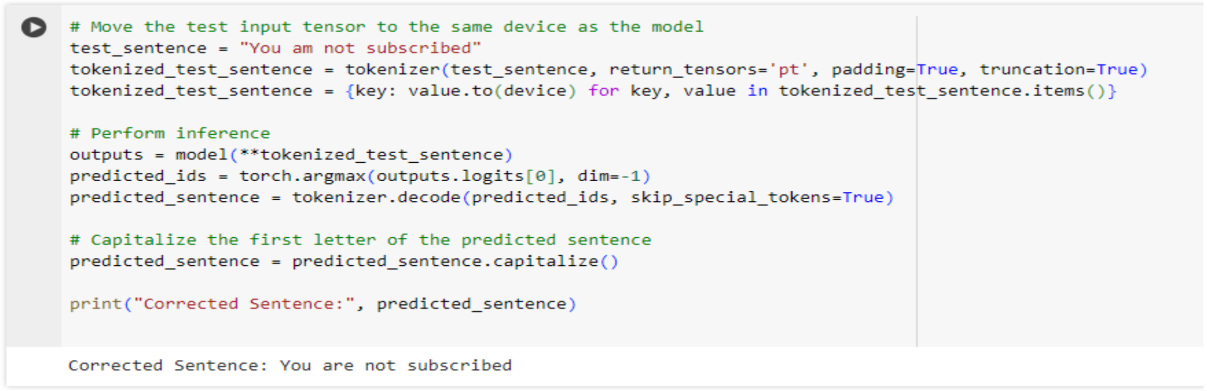


```
from google.colab import drive

# Specify the directory where you want to save the model in Google Drive
output_dir = "/content/drive/MyDrive/fine_tuned_bert_model"

# Create the directory if it doesn't exist
import os
if not os.path.exists(output_dir):
    os.makedirs(output_dir)
0
# Save the model and tokenizer
model.save_pretrained(output_dir)
tokenizer.save_pretrained(output_dir)

print("Model saved successfully at:", output_dir)
```

```
Model saved successfully at: /content/drive/MyDrive/fine_tuned_bert_model
```

8. Perform Inference:

- Perform inference on a test sentence to demonstrate the corrected output.

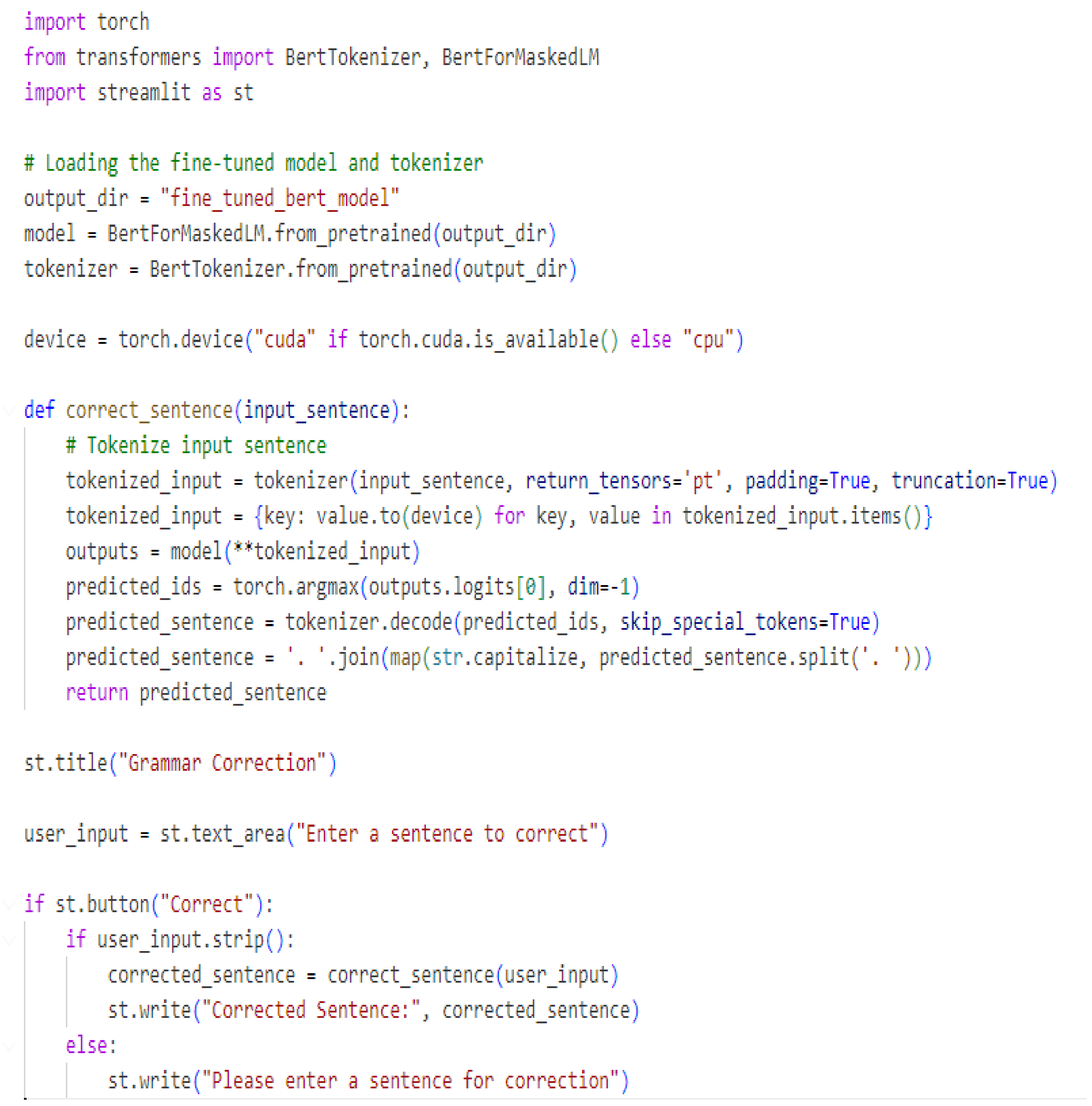


```
# Move the test input tensor to the same device as the model
test_sentence = "You am not subscribed"
tokenized_test_sentence = tokenizer(test_sentence, return_tensors='pt', padding=True, truncation=True)
tokenized_test_sentence = {key: value.to(device) for key, value in tokenized_test_sentence.items()}

# Perform inference
outputs = model(**tokenized_test_sentence)
predicted_ids = torch.argmax(outputs.logits[0], dim=-1)
predicted_sentence = tokenizer.decode(predicted_ids, skip_special_tokens=True)

# Capitalize the first letter of the predicted sentence
predicted_sentence = predicted_sentence.capitalize()

print("Corrected Sentence:", predicted_sentence)
```

```
Corrected Sentence: You are not subscribed
```

9. Streamlit application:

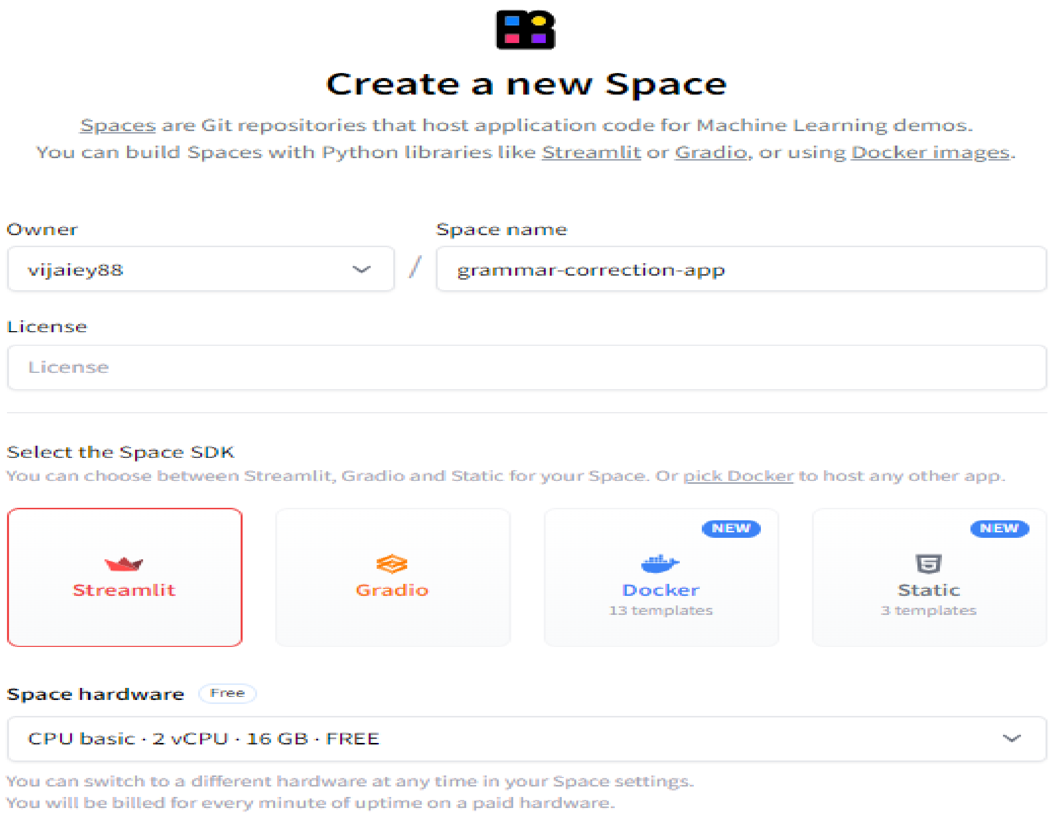
- Import the necessary libraries, including torch for PyTorch, BertTokenizer and BertForMaskedLM from the transformers library for BERT model handling, and streamlit for building the web application.
- Load the fine-tuned BERT model and tokenizer from the specified directory.
- Tokenize the input sentence using the tokenizer and convert it into PyTorch tensors.
- Pass the tokenized input to the model to obtain the predicted tokens.
- Decode the predicted tokens to get the corrected sentence.



```python
import torch
from transformers import BertTokenizer, BertForMaskedLM
import streamlit as st

# Loading the fine-tuned model and tokenizer
output_dir = "fine_tuned_bert_model"
model = BertForMaskedLM.from_pretrained(output_dir)
tokenizer = BertTokenizer.from_pretrained(output_dir)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

def correct_sentence(input_sentence):
    # Tokenize input sentence
    tokenized_input = tokenizer(input_sentence, return_tensors='pt', padding=True, truncation=True)
    tokenized_input = {key: value.to(device) for key, value in tokenized_input.items()}
    outputs = model(**tokenized_input)
    predicted_ids = torch.argmax(outputs.logits[0], dim=-1)
    predicted_sentence = tokenizer.decode(predicted_ids, skip_special_tokens=True)
    predicted_sentence = '. '.join(map(str.capitalize, predicted_sentence.split('. ')))
    return predicted_sentence

st.title("Grammar Correction")

user_input = st.text_area("Enter a sentence to correct")

if st.button("Correct"):
    if user_input.strip():
        corrected_sentence = correct_sentence(user_input)
        st.write("Corrected Sentence:", corrected_sentence)
    else:
        st.write("Please enter a sentence for correction")
```

Deployment

Deployed in hugging face public community.



Create a new Space

Spaces are Git repositories that host application code for Machine Learning demos.
You can build Spaces with Python libraries like Streamlit or Gradio, or using Docker images.

Owner: vijaiey88 / Space name: grammar-correction-app

License: License

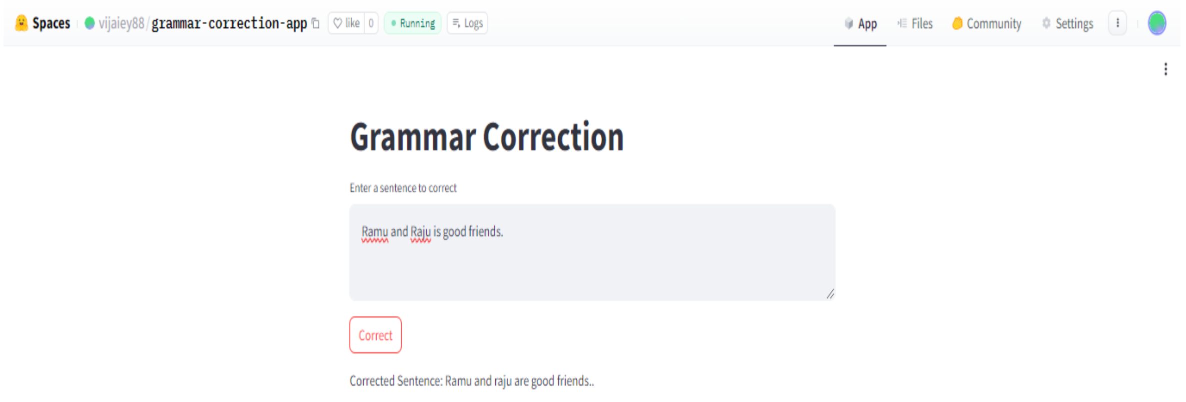
Select the Space SDK
You can choose between Streamlit, Gradio and Static for your Space. Or pick Docker to host any other app.

Space hardware Free

CPU basic · 2 vCPU · 16 GB · FREE

You can switch to a different hardware at any time in your Space settings.
You will be billed for every minute of uptime on a paid hardware.

App:



Grammar Correction

Enter a sentence to correct

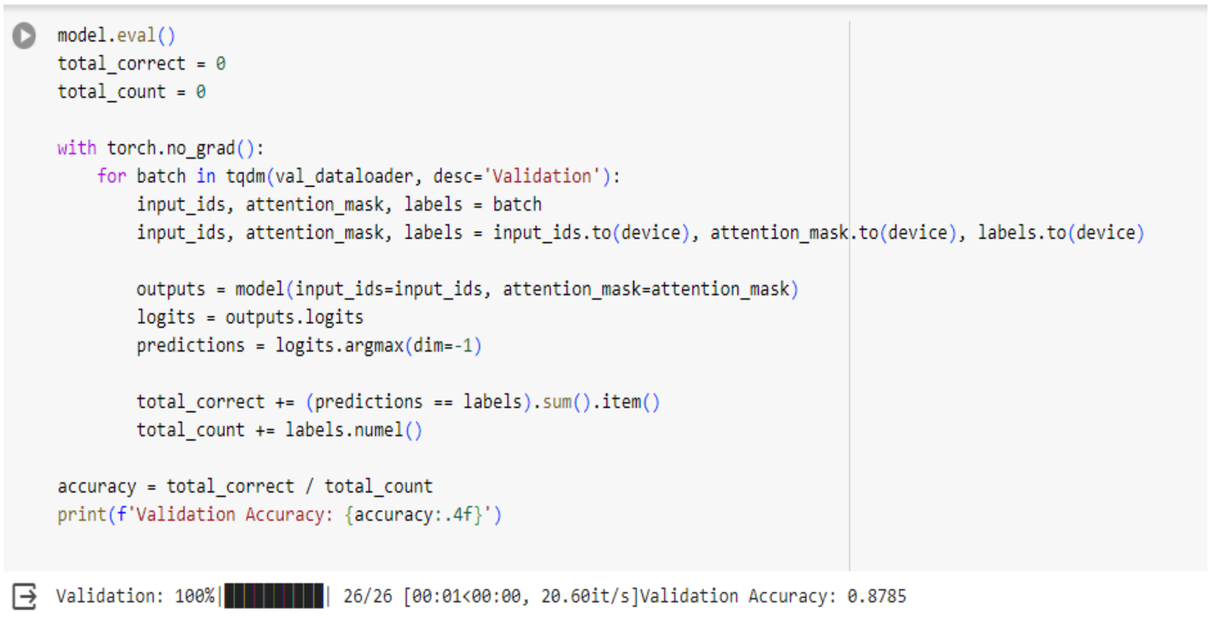
Ramu and Raju is good friends.

Corrected Sentence: Ramu and raju are good friends..

App link: Grammar Correction App

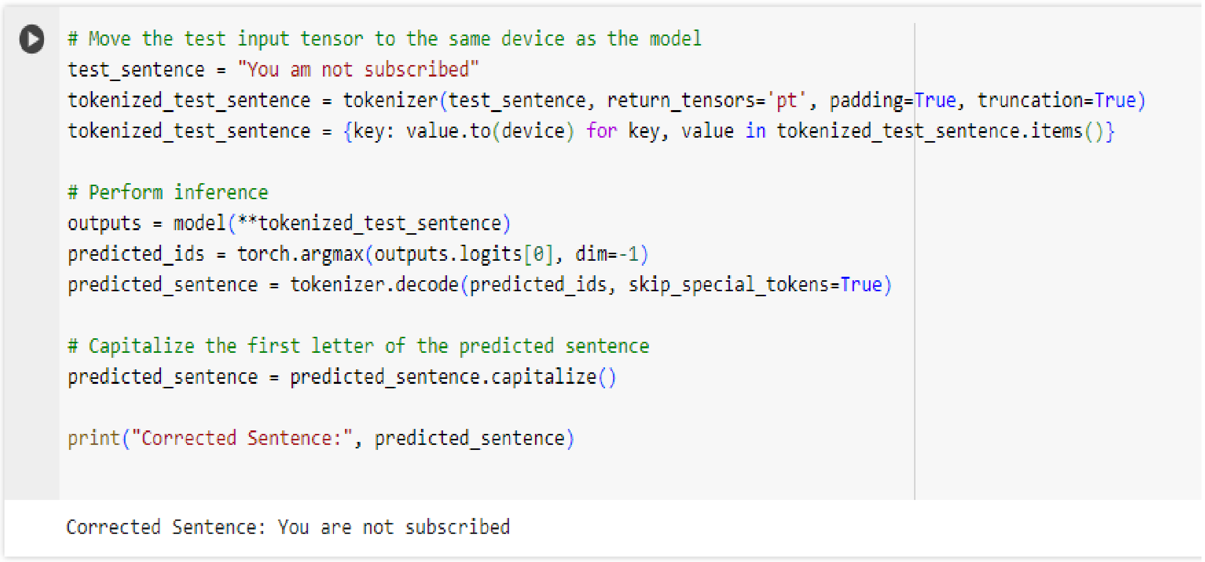
Results and Discussion

The results indicate that the fine-tuned BERT model achieves promising performance in correcting grammatical errors with accuracy of 87.85% on validation set. The model demonstrates effectiveness in handling various error types and shows potential for real-world applications.



```
model.eval()
total_correct = 0
total_count = 0

with torch.no_grad():
    for batch in tqdm(val_dataloader, desc='Validation'):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)

        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        predictions = logits.argmax(dim=-1)

        total_correct += (predictions == labels).sum().item()
        total_count += labels.numel()

accuracy = total_correct / total_count
print(f'Validation Accuracy: {accuracy:.4f}')
```

```
Validation: 100%|██████████| 26/26 [00:01<00:00, 20.60it/s]Validation Accuracy: 0.8785
```



```
# Move the test input tensor to the same device as the model
test_sentence = "You am not subscribed"
tokenized_test_sentence = tokenizer(test_sentence, return_tensors='pt', padding=True, truncation=True)
tokenized_test_sentence = {key: value.to(device) for key, value in tokenized_test_sentence.items()}

# Perform inference
outputs = model(**tokenized_test_sentence)
predicted_ids = torch.argmax(outputs.logits[0], dim=-1)
predicted_sentence = tokenizer.decode(predicted_ids, skip_special_tokens=True)

# Capitalize the first letter of the predicted sentence
predicted_sentence = predicted_sentence.capitalize()

print("Corrected Sentence:", predicted_sentence)
```

```
Corrected Sentence: You are not subscribed
```

Conclusion

In conclusion, the fine-tuned BERT model presents a viable solution for grammatical error correction tasks. Further optimization and experimentation could enhance its performance and applicability in practical scenarios.

References:

1. GitHub – https://github.com/google-research/bert?tab=readme-ov-file

2. Research paper - https://arxiv.org/abs/1810.04805

3. Online materials - https://www.analyticsvidhya.com/blog/2021/05/all-you-need-to-know-about-bert/

4. Online materials - https://jalammar.github.io/illustrated-bert/

5. Hugging face - https://huggingface.co/google-bert/bert-base-uncased

6. YouTube - https://www.youtube.com/watch?v=DkzbCJtFvqM&t=876s