Technische Hochschule Deggendorf
Faculty of Angewandte Informatik
The University of South Bohemia in České Budějovice
Faculty of Science

Degree Master Artificial Intelligence and Data Science

Grammatikfehlerkorrektur mit Deep Reinforcement Learning

Grammar Error Correction using Deep Reinforcement Learning

Master's thesis to obtain the academic degree:

Master of Science (M.Sc.)

at the Technical University of Deggendorf
and the University of South Bohemia

Presented by:
Raj Kumar Rana
Matriculation number:
12102667

On: 18. Januar 2023

Supervisor:
Prof. Dr. Andreas Fischer

Co-Supervisor:
Tom Cvjetkovic, M.Sc
Mentorium GmbH

Declaration

Name of the student: Raj Kumar Rana

Name of the supervisor: Prof. Dr. Andreas Fischer

Topic of the thesis:

Grammar Error Correction using Deep Reinforcement Learning

1. I hereby declare that I have written the final thesis independently in accordance with § 35 Para. 7 RaPO (examination regulations for the universities of applied sciences in Bavaria, BayRS 2210-4-1-4-1-WFK) and have not yet submitted it elsewhere for examination purposes, no other than have used the specified sources or aids and have marked literal and analogous quotations as such. I declare that I am the author of this qualification thesis and that in writing it I have used the sources and literature displayed in the list of used sources only.

   Deggendorf, 18.01.2023 [illegible]
   Date Signature of student

2. Release of the thesis:

   [✓ Thesis in full is released immediately]
   ◯ Release of the thesis in full is postponed
   ◯ Full version to be archived and shortened version to be released

   Deggendorf, 18.01.2023 [illegible]
   Date Signature of student

Annotation

R. K. Rana, “Grammar error correction using deep reinforcement learning,” M.S. thesis, in English, Faculty of Applied Computer Science, Deggendorf Institute of Technology, Deggendorf, Germany and Faculty of Science, University of South Bohemia, České Budějovice, Czech republic, Jan. 2023, p. 44

Annotation:
Reinforcement Learning (RL) method was used to fine-tune a sequence-to-label model for the Grammar Error Correction (GEC) task. Required components of RL like the environment, reward function and policy gradient algorithm were implemented. An action-search algorithm was implemented to mitigate the training instability due to the high dimension of the state and action spaces. The action-search algorithm used the world model to guide the policy with action selection. The results of our models trained with Supervised Learning (SL) and RL were compared to other GEC systems on three GEC benchmarks.

I declare that I am the author of this qualification thesis and that in writing it I have used the sources and literature displayed in the list of used sources only.

Deggendorf, 18.01.2023 [illegible]

Date Signature of student

Abstract

In this thesis, we investigate the potential benefit of fine-tuning a sequence-to-label model for a grammar error correction task using deep reinforcement learning instead of traditionally used supervised learning. We show that the iterative error correction method used in the evaluation stage by the recent sequence-to-label models can also be incorporated in the training stage by formulating it as a sequential task and optimizing the model using a reinforcement learning algorithm. We created a reinforcement learning environment for the grammar error correction task and implemented an action-search algorithm to utilize the world model and update the model using a policy gradient algorithm. By using reinforcement learning, we demonstrate the potential to eradicate the requirement to manually label the data for the supervised learning method and utilize the self-learning ability of reinforcement learning from simple parallel datasets of correct and incorrect sentences.

Contents

1 Introduction

Written communication is essential at almost every professional level like education, work and research. Poor grammar can hinder the readers' understanding and affect the credibility of the writer. However, like every other skill, improving one's writing skills requires a lot of time and exercise. By identifying and correcting errors, Grammar Error Correction (GEC) systems can aid language learners to improve their skills and confidence in writing in a foreign language. Users, who do not have time to study and improve their grammar skills, can also benefit from GEC systems. In general, GEC systems enhance the quality, effectiveness and ease of written communication.

1.1 Motivation

Recent GEC systems[2], [3] approached the GEC task as a sequence labelling task, where the model predicts the edit operation labels for input tokens instead of directly generating the output tokens. The edit labels consist of labels to delete, insert or replace tokens. Compared to the popular neural machine translation (NMT) approach, this approach has faster inference speed[2], [3] with comparative results. Since each token can only be updated once by its edit label, multiple iterations of corrections are applied to correct the grammar errors in the sentence. However, the training stage of these systems does not implement this iterative method to update the model weights and the model is only trained on the first iteration of correction due to the limitation of the Supervised Learning (SL) method. On the other hand, Reinforcement Learning (RL) methods are suitable for training models on sequences of events. By implementing deep RL[1] method on the GEC task, the iterative correction methods can be incorporated into the training process. Since the model will be trained using a reward function in RL, manually labelling the edit labels in the training data will not be required anymore. Even though RL has been used in other language processing tasks[4], there has been only one research on solving GEC using deep RL[5]. In this thesis, we introduce the required framework to solve GEC as an RL task and investigate its effectiveness over the traditional SL approach for GEC. The source code of this thesis is publicly available[2].

1.2 Outline

This section outlines the organization of the chapters in this thesis. Chapter 2 discusses previous researches that relate to our work in this thesis. Chapter 3 covers the background information necessary for understanding the contents of this thesis, including grammar error

[1]Deep RL refers to Reinforcement Learning using Artificial Neural Networks (ANN) from Deep Learning (DL).
[2]Source code: `https://github.com/RajK853/DRL-GEC`

correction and its approaches, pre-trained BERT models and an overview of supervised learning and reinforcement learning techniques. Chapter 4 provides details on the dataset used in this thesis, including the process of filtering it and the model architecture. Chapter 5 describes the reinforcement learning environment and its reward function, the action search algorithm and the reinforcement learning algorithm implemented in this thesis. Chapter 6 presents a brief description of the GEC benchmarks and compares the results of our experiments with other GEC systems on these benchmarks. Chapter 7 concludes the work in this thesis and suggests potential directions for future research.

2 Related Works

[6] implemented a sequence-to-sequence approach to use edit operation tags to keep or delete tokens from the input sentence. The motivation behind using the edit operations was to remove the redundant decoding of the tokens that stay the same in the input and output sentences. Therefore, their decoder used a relatively smaller auto-regressive transformer layer that generated the KEEP and DELETE tags along with some additional phrases for the output sentence. [2] and [3] implemented a sequence-to-label approach to only generate edit tags for each token in the input sentence. They introduced additional tags to insert, replace and transform tokens. [3] also implemented high-level tags to transform the case of a token, the tense of a verb and the noun number. Both [2] and [3] used an iterative correction method to correct the errors in the sentence which allows the GEC model to detect and correct errors which became more evident after correcting other errors in the sentence. All of these methods used SL on labelled datasets and to the best of our knowledge, only [5] investigated fine-tuning a GEC system using deep RL. Our approach differs from [5] in terms of the neural network model, model architecture and RL environment. Similar to [3], we use a sequence-to-label architecture with a pre-trained Bidirectional Encoder Representations from Transformers (BERT) model as the encoder instead of a sequence-to-sequence architecture with a bidirectional Gated Recurrent Unit (bi-GRU)[7] as the encoder and decoder. Finally, we optimize our model using our own RL environment for GEC which uses a different reward function from [5].

3 Background Knowledge

3.1 N-gram

An n-gram is a contiguous sequence[1] of n items from a given sequence. In Natural Language Processing (NLP), the items can be words, syllables or characters and the value of n determines the size of the n-gram. For instance, the different n-grams of tokens for the sentence *"This is a really great sentence ."* is shown in Table 3.1. N-grams are used in NLP to analyze the frequency and distribution of word combinations. They are used in a variety of NLP tasks including language modelling[8], [9], information retrieval[10], machine translation[11], text classification[12] and sentiment analysis[13].

n	Name	N-gram
1	Unigram	This, is, a, really, great, sentence, .
2	Bigram	This is, is a, a great, great sentence, sentence .
3	Trigram	This is a, is a great, a great sentence, great sentence .
4	Four-gram	This is a great, is a great sentence, a great sentence .

Table 3.1: N-grams of a sample sentence

3.2 Grammar Error Correction

Grammar Error Correction (GEC) is the process of detecting and correcting grammar errors in a sentence. Modern GEC systems are expected to not only fix grammatical and spelling errors but also improve the overall fluency of the corrected text. There have been several methods proposed for GEC systems. This section will briefly discuss each of the methods shown in Figure 3.1.

Original	She see Tom is catched by policeman in park at last night.
Corrected	She saw Tom caught by a policeman in the park last night.

Table 3.2: Example of Grammar Error Correction

[1]A contiguous sequence is a set of consecutive items in the same order as in the original set.

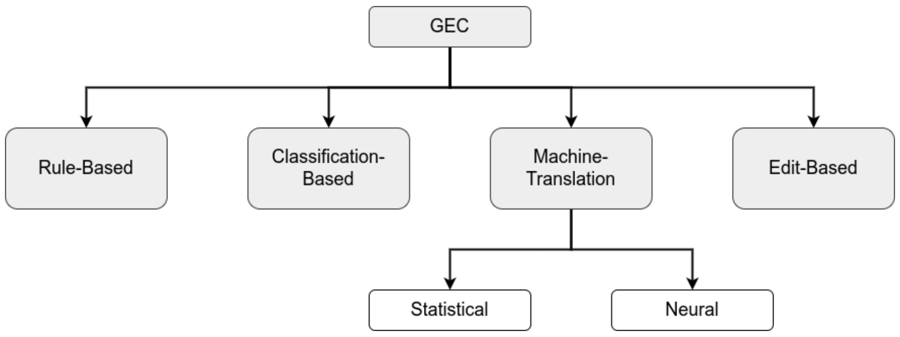


Figure 3.1: Taxonomy of GEC approaches

3.2.1 Rule-Based Approach

One of the earliest methods for GEC[14], [15] involved using manually created grammar rules and tools such as parsers and lexical resources like lookup tables and lexicons to identify and fix specific grammar mistakes in texts. While these methods worked well for specific grammar errors, developing and maintaining these grammar rules is time-consuming and labour-intensive. Not to mention that these systems were incapable of handling complex grammar errors correctly.

3.2.2 Classification-Based Approach

These approaches use Machine Learning (ML) algorithms to train a classifier on a large dataset to detect different types of errors[16]–[18]. For each grammar category, a separate model is trained to classify errors only from that particular category. For instance, a classifier to correct article errors can have three output classes to update an article into "a"/"an", "the" or "no-article". While these methods could handle more errors than the rule-based methods, they required heavy feature engineering and using a separate classifier for each category makes it difficult to handle complex errors that depend on the errors from other categories.

3.2.3 Machine-Translation Approach

In recent years, a lot of GEC methods have been proposed to handle GEC as a machine translation task. Instead of translating text from a source language to a target language, it attempts to translate from ungrammatical text into grammatically correct text. Statistical Machine Translation (SMT) and Neural Machine Translation (NMT) are two different machine translation approaches used for GEC.

- **SMT:** This is a traditional approach of using statistical models for machine translation[19]. Statistical models like the translation model and language model are trained on a large corpus of parallel texts[2] to learn the probabilistic model to generate candidate translations and to score and select the most likely translation. [20] introduced a hybrid

[2]In GEC, a parallel text refers to a pair with correct and incorrect sentences.

system of utilizing the rule-based and SMT-based approaches along with a big language model.

- **NMT:** In this approach, a sequence-to-sequence model is trained on a large corpus of parallel texts to automatically translate texts. The model uses an encoder-decoder neural network architecture as shown in Figure 3.2 where the encoder converts the input sentence into an encoded form which is decoded into the output sentence by the decoder. Despite the high computational cost, NMT is the most popular approach because of its effectiveness in GEC[21]–[23].

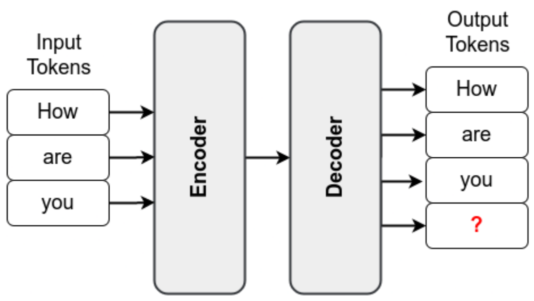


Figure 3.2: Sequence-to-Sequence Model Architecture

3.2.4 Edit-Based Approach

In this approach, the model predicts the edit operation labels per each token that are expected to correct the current sentence. While the set of edit operations varies among the papers[2], [3], [6], it usually includes edits to keep, delete, insert and replace the tokens. In this approach, a post-processing step is required to apply the edit operations on the input sentence to generate the output sentence. The bottleneck of this approach is that the edit labels are hard-coded and limited to certain most frequent tokens. Therefore, the selection of the edit operations determines the type of errors it can correct.

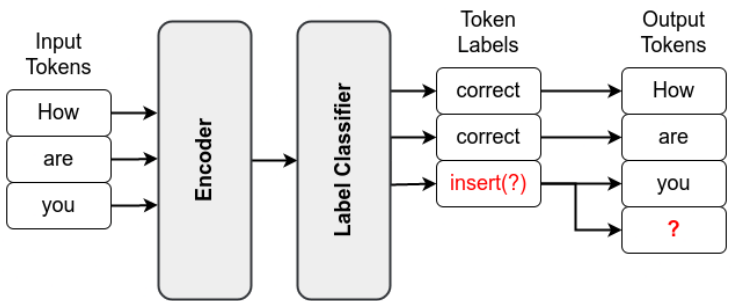


Figure 3.3: Sequence-to-Label Model Architecture

3.3 GEC Evaluation Metrics

This section will discuss the different evaluation metrics used by the GEC benchmarks in Section 6.1.

3.3.1 GLEU

Google BLEU (GLEU)[24] is a variant of the BLEU (BiLingual Evaluation Understudy)[25] developed by Google to automatically evaluate machine translation systems using a set of reference sentences. The BLEU score is calculated as a harmonic mean of the precision at different n-gram levels between the output and the reference sequences. The GLEU score was introduced with a modified precision function because the BLEU score was designed for evaluation at the corpus level and, therefore, its sentence level score did not correlate well with human evaluation. [26] and [27] introduced the variation of the GLEU score as in Equation 3.1 from [27, Equation 1] that can be used as an evaluation metric for the GEC task. Its advantage over the other evaluation metrics is that it only requires the source and reference sentences in contrast to other metrics that require gold annotations to correct the input sentences.

$$\mathrm{GLEU}(C,S,R) = \mathrm{BP}\cdot exp\left(\sum_{n=1}^{N} w_n \log p_n(C,S,R)\right) \tag{3.1}$$

$$\mathrm{BP} = \min\left(1, e^{1-\frac{\mathrm{length}(R)}{\mathrm{length}(C)}}\right)$$

$$p_n(C,S,R) = \frac{\sum\limits_{ngram\in(C\cap R)} \mathrm{count}_{C,R}(ngram) - \sum\limits_{ngram\in(C\cap S)} \max(0, \mathrm{count}_{C,S}(ngram) - \mathrm{count}_{C,R}(ngram))}{\sum\limits_{ngram\in\{C\}} \mathrm{count}_C(ngram)} \tag{3.2}$$

$$\mathrm{count}_{A,B}(ngram) = \min(\text{\# occurrences of ngram in A, \# occurrences of ngram in B})$$

where
C, S and R are the set of candidate output, source and reference sentences respectively,
BP is the brevity penalty from the BLEU score,
N is the maximum size of n-gram, defaults to 4,
w_n is the weight of the given n-gram, defaults to $1/N$ and
$p_n(C,S,R)$ is the precision of the n-gram.

3.3.2 M2

MaxMatch (M2)[28] is an algorithm that automatically extracts phrase-level edits between source and reference sentences by achieving the highest overlap with the gold-standard annotation. Levenshtein Distance[3] is used to construct the edit lattice (Table 3.3) and compute

[3]Levenshtein Distance is the minimum number of insertion, deletion and substitution operations required to transform one sequence into another sequence.

the similarity between the spans of the source and reference sentences. The alignment of tokens in the source and reference sentences can be achieved by identifying the shortest path on this lattice, ranging from the top-left corner to the bottom-right corner. The values along the edges of this path are used to determine the token edit operations (keep, delete, insert or replace) required to transform the source sentence into the reference sentence.

		Our	**baseline**	**system**	**feeds**	**a**	**word**	**into**	**PB-SMT**	**pipeline**	**.**
	0	1	2	3	4	5	6	7	8	9	10
Our	1	**0**	1	2	3	4	5	6	7	8	9
baseline	2	1	**0**	1	2	3	4	5	6	7	8
system	3	2	1	**0**	1	2	3	4	5	6	7
feeds	4	3	2	1	**0**	**1**	2	3	4	5	6
word	5	4	3	2	1	1	**1**	2	3	4	5
into	6	5	4	3	2	2	2	**1**	2	3	4
PB-SMT	7	6	5	4	3	3	3	2	**1**	2	3
pipeline	8	7	6	5	4	4	4	3	2	**1**	2
.	9	8	7	6	5	5	5	4	3	2	**1**

Table 3.3: The Levenshtein matrix from [28, Figure 1] between the source sentence (in the row) and the reference sentence (in the column). The highlighted cells indicate the shortest path obtained from breadth-first search.

Let an edit be a triple of $\{a, b, C\}$ with start-end edit index and correction value or set of corrections, $G = \{g_1, \ldots, g_n\}$ be a set of gold edits and $E = \{e_1, \ldots, e_n\}$ be a set of system edits with maximum overlap with G. Then the system edits are evaluated with the gold edits by calculating $\mathrm{F}_{0.5}$ score[29], which is a harmonic mean between the precision P and the recall R with more emphasis on the precision P as shown in Equation 3.3.

$$\mathrm{F}_{0.5} = \frac{1.25 \cdot \mathrm{P} \cdot \mathrm{R}}{0.25 \cdot \mathrm{P} + \mathrm{R}} \tag{3.3}$$

$$\mathrm{P} = \frac{\text{True Positives}}{\text{True Positives + False Positives}} = \frac{\sum\limits_{i=1}^{n} |e_i \cap g_i|}{\sum\limits_{i=1}^{n} |e_i|} \tag{3.4}$$

$$\mathrm{R} = \frac{\text{True Positives}}{\text{True Positives + False Negatives}} = \frac{\sum\limits_{i=1}^{n} |e_i \cap g_i|}{\sum\limits_{i=1}^{n} |g_i|} \tag{3.5}$$

where

$$e_i \cap g_i = \{e \in e_i | \exists g \in g_i, \mathrm{match}(e, g)\}$$
$$\mathrm{match}(e, g) \Leftrightarrow e.a = g.a \wedge e.b = g.b \wedge e.C \in g.C$$

3.3.3 ERRANT

By using a rule-based approach for error annotation, ERRor ANnotation Toolkit (ERRANT)[30] mitigates the M2 scorer's limitation of manually labelling the error types of extracted edits. ERRANT uses the linguistically-enhanced alignment algorithm[31] to extract more realistic edits from the parallel source and reference sentences as shown in Figure 3.4. In contrast to M2's use of Levenshtein Distance, [31] uses Damerau-Levenshtein, which is an extension of Levenshtein that can also handle the transposition of sequences i.e. AB $\rightarrow$ BA. Similar to M2, ERRANT also uses the $F_{0.5}$ score between the system and gold edits to evaluate GEC systems. [30] highlighted the reliability of the $F_{0.5}$ score of the ERRANT over M2 by showing that M2's $F_{0.5}$ score overestimates the performance of the GEC system by exploiting the edit boundary to maximize true positives and minimize false negatives.

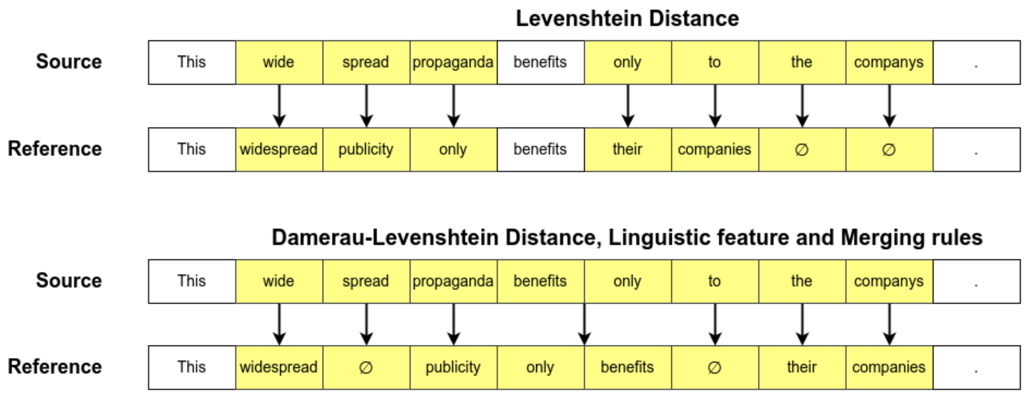


Figure 3.4: Comparison of token mappings for the edit extraction with Levenshtein Distance and ERRANT's edit extraction method[31, Table 2]

3.4 BERT

Bidirectional Encoder Representations from Transformers (BERT) models[32] are transformer-based[33] Language Models (LM) that are pre-trained on large unlabeled text corpus to learn about the hidden representations of the language itself. These pre-trained BERT models can be further fine-tuned on specific downstream tasks like Named Entity Recognition (NER) and Stanford Question Answering Dataset (SQuAD)[34], [35] via Transfer Learning (TL)[4].

As inputs, the BERT model takes in a concatenation of two sequences of tokens $x_1, x_2, \ldots, x_N$ and $y_1, y_2, \ldots, y_M$ as [CLS], $x_1, x_2, \ldots, x_N$, [SEP], $y_1, y_2, \ldots, y_M$, [EOS] where [CLS], [SEP] and [EOS] are special tokens for the classification task, sequence separator and end of the sequence respectively. The BERT model is then pre-trained on the following objectives:

[4]Transfer Learning is the process of using an acquired knowledge from a task to solve another different but related task.

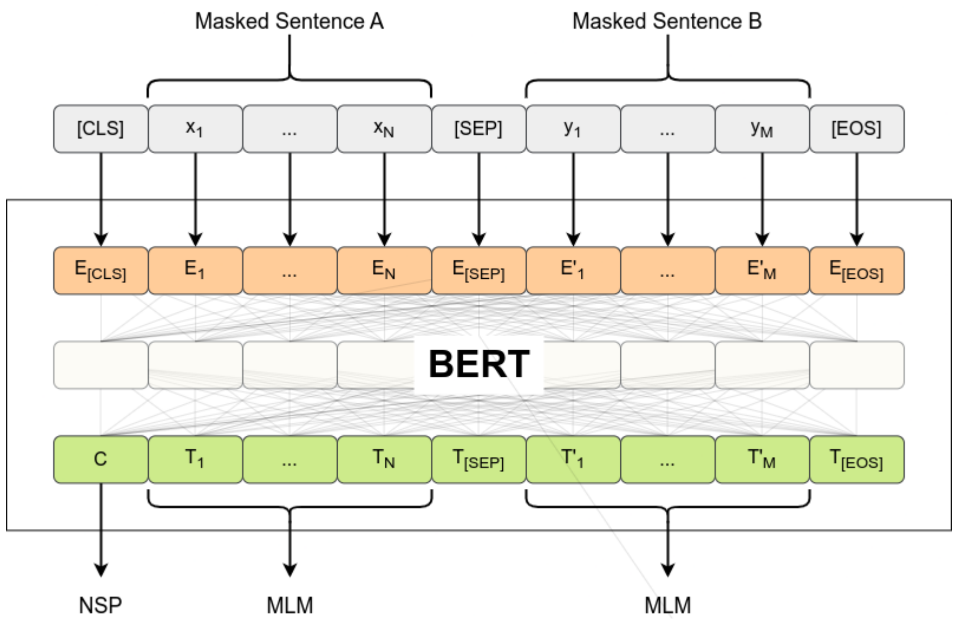


Figure 3.5: Overview of pre-training of the BERT model

1. **Masked Language Model (MLM):** For this task, approximately 15% of the input tokens are randomly replaced with a special mask token [MASK] and the model is optimized to predict the original tokens that were replaced with the mask token [MASK].

2. **Next Sentence Prediction (NSP):** NSP is a binary classification task of determining whether the two segments in the input sequences follow each other in the original text. The segments are sampled randomly such that they are either two consecutive sentences from a text or two sentences from different texts.

3.5 RoBERTa

[36] identified that the BERT model was heavily under-trained and they proposed several modifications to optimize the pre-training of the BERT model, which they called Robustly Optimized BERT Approach (RoBERTa). The RoBERTa model differs from the BERT models in the following ways:

1. **Static vs Dynamic Masking:** RoBERTa generates masking patterns dynamically for the MLM task for every pre-training step compared to BERT's approach of generating 10 different masking patterns for each sequence during the data pre-processing stage.

2. **No NSP objective:** [36] conducted pre-training and fine-tuning experiments on the RoBERTa model with and without the NSP objective and realized that it may not be necessary [37]–[39].

3. **Larger batch size and learning rate:** According to [40] and [41], BERT models can achieve good results in the benchmarks quickly using larger batch sizes and learning rates. Therefore, RoBERTa was pre-trained on a batch size of 8k sequences with a learning rate of 1e-3 compared to BERT's batch size of 256 sequences with a learning rate of 1e-4.

4. **Larger text encoding:** RoBERTa uses a byte-level Byte-Pair Encoding (BPE) [42] vocabulary of 50k sub-word units compared to BERT's character-level BPE encoding with 30k vocabulary size.

3.6 Supervised Learning

Supervised Learning (SL) is a type of Machine Learning (ML)[5] where the model is trained on labelled data. Let $X = \{x_1, \dots, x_N\}$ be a set of input features and $Y = \{y_1, \dots, y_N\}$ be the set of output labels, then the model π with a set of parameters θ is updated to minimize the given objective:

$$\frac{1}{N} \sum_{n=1}^{N} \mathcal{L}(\pi_\theta(x_n), y_n) \tag{3.6}$$

The objective function $\mathcal{L}$ depends on the category of SL. For the regression task where the output label y_n is a continuous value, the model is updated with objective functions like Mean Square Error (MSE). Likewise, for the classification task where the output label y_n is a class among N classes, the model outputs a probability distribution over the N classes and it is updated with objective functions like Cross-Entropy (CE). In SL, the goal is to learn a model by updating it based on the difference between the predicted labels and the true labels. In the end, the trained model is expected to make accurate predictions for input features that it has not seen during training, based on the patterns it has learned from the labelled data.

3.7 Reinforcement Learning

Reinforcement Learning (RL) is a branch of ML where the model learns by using its own experience collected by interacting with an environment. The experience reflects the reward or punishment the model received for performing certain actions in states of the environment. RL is suitable for solving sequential tasks where a series of events need to occur to solve a task like opening a door or parking a car. Algorithm 1 shows very generalized steps of RL and its different components.

[5]Machine learning (ML) is a field of artificial intelligence (AI) that enables computers to learn and make decisions based on data, without being explicitly programmed to do so.

Algorithm 1 Basic RL

```
1: Initialize the agent, $\pi_\theta$
2: Initialize the environment, $\mathcal{E}$
3: Initialize the experience buffer, $\mathcal{D} \leftarrow \emptyset$
4: for each episode do
5:     $s \leftarrow$ Reset($\mathcal{E}$)    ▷ Reset the environment to get the initial state
6:     while not done do    ▷ Current episode has not terminated
7:         $a \leftarrow \pi_\theta(s)$    ▷ Select an action $a$ based on the current state $s$
8:         $(r, s', done) \leftarrow \mathcal{E}(a)$    ▷ Interact with environment $\mathcal{E}$ to get reward and next state
9:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s, a, r, s', done)\}$    ▷ Store current experience to the buffer $\mathcal{D}$
10:        $s \leftarrow s'$    ▷ Update current state with next state
11:    $\theta \leftarrow$ Update($\theta, \mathcal{D}$)    ▷ Update agent parameters using RL algorithm
12:    $\mathcal{D} \leftarrow \emptyset$    ▷ Reset experience buffer
```

RL can be formulated as the Markov Decision Process (MDP) defined as a tuple $(S, A, \mathcal{P}, R, \gamma)$ where:

- S is the state space which determines the states of the agent in the environment.
- A is the action space which indicates the set of actions the agent can take in the particular state of the environment.
- $\mathcal{P}$ is the state transition function, $\mathcal{P}(s_{t+1}|s_t, a_t)$, which determines the next state, s_{t+1}, of the agent in the environment given its action, a_t, at the state, s_t. It can be either deterministic or stochastic depending on the environment. If an action, a_t, is applied to a state, s_t, several times, a deterministic transition function will always get the same next state, s_{t+1}. Whereas, a stochastic transition function may reach a different next state, s_{t+1}, every time because of some hidden factors of the environment.
- R is the reward function which assigns a reward or punishment value, $r_t \leftarrow R(s_t, a_t)$, indicating the quality of the action, a_t, at the state, s_t.
- γ is the discount factor which indicates the relative importance of the immediate reward, r_t, versus future rewards, $r_{t+1} + \cdots + r_\infty$, as in Equation 3.7.

In RL, the agent is commonly interchanged with the policy, π_θ, which is a function that decides the action, a_t, for the state, s_t. The goal of the agent is to learn an optimal policy, π_*, that maximizes the expected return, G_t, which is the sum of the discounted rewards obtained by following the policy:

$$\begin{aligned} G_t &= \sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \\ &= R(s_t, a_t) + \gamma R(s_{t+1}, a_{t+1}) + \gamma^2 R(s_{t+2}, a_{t+2}) + \dots \\ &= R(s_t, a_t) + \gamma \left(R(s_{t+1}, a_{t+1}) + \gamma R(s_{t+2}, a_{t+2}) + \dots \right) \\ &= R(s_t, a_t) + \gamma G_{t+1} \end{aligned} \tag{3.7}$$

Generally, the RL interactions are divided into episodes which end when the environment reaches a terminal state or when a predetermined number of steps have been taken. In such a case, the goal of the agent is to maximize the expected return it receives over the course of an episode.

$$\begin{aligned} G_t &= \sum_{t=0}^{T} \gamma^t R(s_t, a_t) \\ &= R(s_t, a_t) + \gamma G_{t+1} \end{aligned} \tag{3.8}$$

where T is the terminal step of an episode and $G_T = R(s_T, a_T)$ is the return of the terminal state.

The environment in RL provides the agent with the interface to interact with. It determines the agent's possible states, actions and rewards and how the agent's actions affect the state of the world and how observations are generated. However, the training of the agent depends on the type of RL algorithm used to learn from the collected experiences. Figure 3.6 shows the taxonomy of RL algorithms based on different categories. In this section, we will discuss very briefly each of these categories.

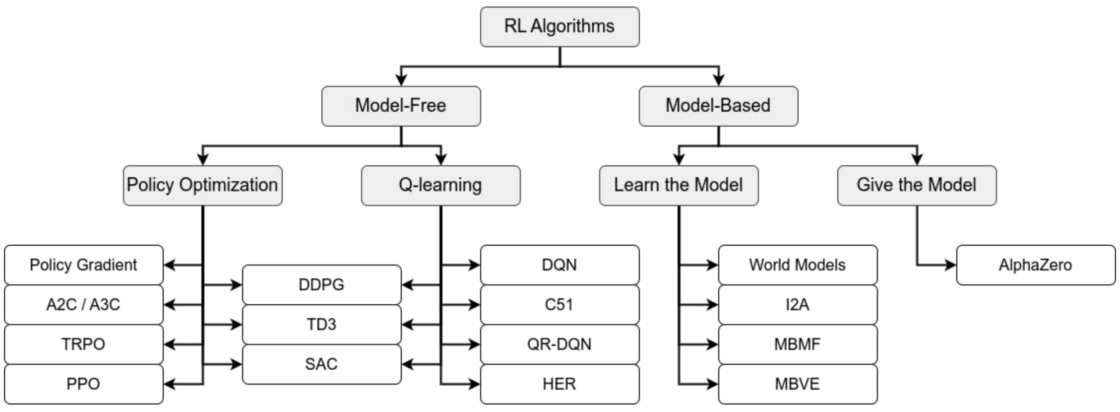


Figure 3.6: Taxonomy of RL algorithms derived from [43]; Policy Gradient[44], A2C/A3C[45], TRPO[46], PPO[47], DDPG[48], TD3[49], SAC[50], DQN[51], C51[52], QR-DQN[53], HER[54], World Models[55], I2A[56], MBMF[57], MBVE[58] and AlphaZero[59]

- **Model-Free vs Model-Based:** The model refers to the world model that consists of the state transition function $\mathcal{P}(s_{t+1}|s_t, a_t)$ and the reward function $R(s_t, a_t)$ of the environment. Having access to the world model would allow the agent to plan ahead by simulating the future outcomes and filtering them to reach the destination with the highest cumulative reward. Usually, the agent does not have access to the world model and it has to explore the environment to optimize the policy without explicitly learning the world model. An RL algorithm is a model-free algorithm if it does not explicitly learn or use the world model. Otherwise, it is a model-based algorithm.

- **Policy Optimization vs Q-learning:** Policy optimization refers to the family of methods where a policy function π is explicitly defined and its parameters θ are optimized by maximizing the objective function $\mathcal{J}_\theta(\pi)$ or its local approximation. Similarly, Q-learning refers to methods that learn the state-action value function, $Q_\theta(s_t, a_t)$, which indicates the quality of taking action, a_t, in the state, s_t. State-action pairs that are expected to give high rewards in the future will have higher Q-values. Here the policy π is implicitly defined by taking action with the maximum Q-value for the given state as follows:
$$a_t \leftarrow \text{argmax}_{a \in A} Q_\theta(s_t, a)$$
However, there are also some algorithms like Soft Actor-Critic (SAC)[50] and Deep Deterministic Policy Gradient (DDPG)[48] that fall under the grey region between the policy optimization and Q-learning as they learn both an explicit policy and a Q-function.

- **Learning vs Using the model:** In some environments like classical board games like Chess, the world model is well-defined and deterministic i.e. the outcomes of the environment are only dependent on the agent's actions and not on some hidden factors of the environment. In such situations, search algorithms can be used on the world models to efficiently explore the environment and plan the actions[59]. However, for environments with complex or unknown world models, an approximation of the world model can be learned and used for planning[57] or generating augmented experiences[55], [58] to train the policy.

4 Methodology

In this chapter, we present the training datasets and the data pre-processing techniques used to prepare data for the SL and RL methods. We also outline the details of our model and its training procedure.

4.1 Datasets

The sparsity of public parallel GEC datasets has motivated many to pre-train their GEC models on a large corpus of artificially generated grammar errors[2], [23], [60], [61] followed by fine-tuning on a specific target corpus. [23] showed that pre-training on synthetic data followed by fine-tuning on a target corpus yields better results than fine-tuning on the joint dataset with synthetic and the target corpus data because the synthetic data dominates over the actual GEC data in the joint dataset, affecting the model performance. As we result, we also implement the two-stage training method using synthetic and actual GEC datasets.

4.1.1 PIE Synthetic

[2] introduced a synthetic dataset by introducing grammatical errors into the One Billion Word benchmark [62]. The incorrect sentences were generated by randomly adding up to five errors by appending, deleting, replacing or changing the verb tokens in the sentence. For further details about the process used to generate these synthetic data, please refer to the paper [2]. This public synthetic GEC dataset contains around 44 million pairs of correct and incorrect sentences. Since these synthetic data do not reflect realistic grammar errors, they are mostly useful for pre-training the models.

4.1.2 W&I+LOCNESS

The W&I+LOCNESS dataset was created by combining the W&I[63][1] and LOCNESS[64] datasets for the BEA-2019 Shared Task[65]. The W&I dataset includes writing samples in various formats, such as letters, stories, articles, and essays, produced by non-native English speakers of varying Common European Framework of Reference for Languages (CEFR) language proficiency levels. The LOCNESS dataset, on the other hand, contains essays written by native English speakers. The combined dataset consists of a total of 43,169 sentences from 3,700 texts, which are divided into train, validation, and test datasets as shown in Table 4.1.

[1]Write & Improve is an online platform offering writing assistance to non-native English students.

		A	B	C	N	Total
Train	Text	1,300	1,000	700	-	3,000
	Sentences	10,493	13,032	10,783	-	34,308
Validation	Text	130	100	70	50	350
	Sentences	1,037	1,290	1,069	998	4,384
Test	Text	130	100	70	50	350
	Sentences	1,107	1,330	1,010	1,030	4,477
Total	Text	1,560	1,200	840	100	3,700
	Sentences	12,637	15,652	12,862	2,018	43,169

Table 4.1: Data distribution of the W&I+LOCNESS dataset. W&I (**A**, **B**, **C**) and LOCNESS (**N**) adapted from [65, Table 2]

4.2 Data Processing

4.2.1 Data Formats

The public GEC datasets like W&I+LOCNESS from the BEA-2019 Shared Task[65] are available in the M2 format as shown below.

```
S So , I think if we have to go somewhere on foot , we must put our hat .
A 16 16|||M:PREP|||on|||REQUIRED|||-NONE-|||0
A 4 5|||R:OTHER|||when|||REQUIRED|||-NONE-|||1
A 16 16|||M:PREP|||on|||REQUIRED|||-NONE-|||1
A 17 18|||R:NOUN:NUM|||hats|||REQUIRED|||-NONE-|||1
A 16 16|||M:PREP|||on|||REQUIRED|||-NONE-|||2
```

In M2 format, lines with the test sentences start with S and lines with annotator corrections start with A. Each annotation line contains the start-end edit token offsets, the ERRANT error type, the correct edit text, 2 redundant fields and the annotator id. The redundant fields, `-REQUIRED-` and NONE, are kept due to some historical reasons from the old CoNLL-2013 Shared Task[66]. In some datasets, there can be more than 1 annotator id per sentence and applying all the edits from each annotator id can generate different correct sentences. On the other hand, the PIE synthetic dataset is available as a parallel dataset with separate files for the correct and incorrect sentences as shown in Table 4.2.

Incorrect	**Correct**
Housing and labor market have not been as strong .	Housing and the labor market have not been as strong .
Pole-position qualifying its that saturday	Pole-position qualifying is Saturday .

Table 4.2: Example of parallel data from PIE synthetic dataset

To fine-tune our model using SL, we use the data format used by [3]. In this format, the tokens and their labels are arranged next to each other in the format given in Figure 4.1.

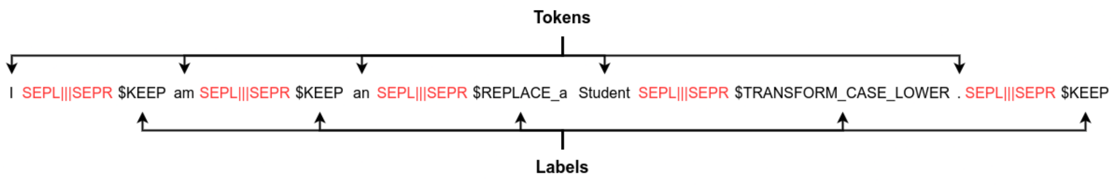

Tokens
I SEPL|||SEPR $KEEP am SEPL|||SEPR $KEEP an SEPL|||SEPR $REPLACE_a Student SEPL|||SEPR $TRANSFORM_CASE_LOWER . SEPL|||SEPR $KEEP
Labels

Figure 4.1: GECToR data format

For the fine-tuning stage using RL, we use data in the JSON format with test and reference sentences as in Figure 4.2. This removes the token labelling requirement from the data and allows the RL environment to operate only on pairs of test and reference sentences. Not to mention, this format also supports examples with multiple reference sentences, where the RL agent will be rewarded if it obtains any of the reference sentences.

```
{
   test: "So , I think if we have to go somewhere on foot , we must put our hat .",
   references: [
      "So , I think if we have to go somewhere on foot , we must put on our hat .",
      "So , I think when we have to go somewhere on foot , we must put on our hats ."
   ]
}
```

Figure 4.2: JSON data format

Figure 4.3 depicts the data conversion pipeline we use to convert the parallel data format from PIE Synthetic dataset and the M2 data format from the W&I+LOCNESS dataset into the JSON format discussed earlier. During this conversion stage, several data filtering methods are applied to remove noisy data from the datasets. Details about the data filtering are discussed in the next section. We adapted the data processing functions from [3] to further convert the dataset into the GECToR format for the SL training stages.

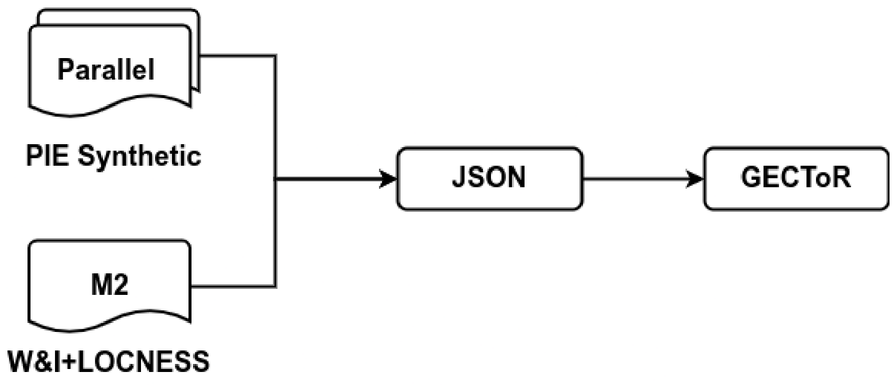


Figure 4.3: Data Format Pipeline

4.2.2 Data Filtering

[67] showed that sequence-to-label GEC models can achieve similar to slightly better results on the filtered GEC dataset, which has better quality with less quantity. As a result, we employ the following data filtering methods:

- **Number of tokens:** Filter sentences based on the minimum and the maximum number of tokens in the test sentences. In this way, we remove examples that are either too short to be informative or too long to possibly cause an issue with the GPU during training.

- **Improper terminal token:** Remove examples whose reference sentences are either not starting with a capitalized token or the ending tokens are not one of these punctuations, .!?". In this way, we remove sentence fragments of a single sentence spanning over multiple examples.

- **Test-reference similarity:** In order to eliminate test sentences that are flawed or semantically different from the reference sentences, we calculated the token-based similarity between the test and reference sentences using a sequence matcher[2].

$$f_{sim}(a,b) = \frac{2 \times \textit{Number of matching tokens between a and b}}{\textit{Number of tokens in a} + \textit{Number of tokens in b}}$$

Test	**Reference**	**Similarity**
You are only relying on it .	It relies on you alone .	0.308
other friends coll poles .	The other friends called the police .	0.333
I hope yours news .	I look forward to your reply .	0.333

Table 4.3: Examples from W&I+LOCNESS dataset with low similarity

- **Ellipsis**: Ellipsis indicates the omission of words from a sentence or it adds some pauses for dramatic effect. Since some of these examples can be incomplete sentences, we remove them from the dataset.

Big shot of the week : Just as he thought BP was back . . .
For example , racing games , action games , puzzle games and more . . .
He threw the door open to reveal . . . a lost puppy .

Table 4.4: Examples with an ellipsis from PIE synthetic dataset

[2]We used `SequenceMatcher` from the standard `difflib` Python library to calculate the token-level similarity ratio.

In the earlier phase of the thesis, we found some examples in the Lang-8[68] dataset that have parenthetical expressions inside brackets only in the reference sentences as shown in Table 4.5. Since they are used to provide additional context information, we remove all parenthetical expressions from the test and reference sentences.

Test	For example , today I ordered some clothes on the internet shop !
Reference	For example , today I ordered some clothes online (you do n't say " internet shop ") .
Cleaned Reference	For example , today I ordered some clothes online .

Table 4.5: Removing parenthetical expressions from reference sentence

Additionally, we normalise some tokens like a double apostrophe, ' ', into a quotation, ", and many more that are included in the data processing script of the BEA-2019 Shared Task datasets like the W&I+LOCNESS dataset. We also use a spell checker to correct any spelling errors from the test sentences. The data filtering parameters used to clean our datasets are listed in Appendix 8.1.

Filter Category	**PIE Synthetic**	**W&I+LOCNESS**	
		Train	Validation
Number of tokens	101,391	2,762	290
Improper terminal token	301,465	1,397	161
Test-reference similarity	170,732	3,333	343
Ellipsis	6,349	1	0
Total Filtered	579,937	7,493	794

Table 4.6: Number of sentences filtered out from each dataset

While filtering the PIE synthetic dataset, we process the sentences until we get 2.0M sentences after filtering. Since it does not have validation data, we split it into train-validation datasets with a 98:2 ratio; the first 1.96M sentences are the training data and the last 400k sentences are the validation data. Similarly, we further filter the W&I+LOCNESS training dataset to remove unsolvable examples, which have at least 1 $UNKNOWN label. The motivation for training on only solvable examples is discussed in Section 5.1.

# of Sentences	**PIE Synthetic**	**W&I+LOCNESS**	
	Train + Validation	Train	Validation
Original	2.58M	34,308	4,384
Filtered	2.0M	26,815	3,590
Only Solvable	-	24,734	-

Table 4.7: Dataset sizes after different stages of filtering

4.3 Models

In this section, we will provide details about our model architecture and its output labels. We will also discuss the multi-stage fine-tuning process using SL and RL to investigate the effect of different fine-tuning methods.

4.3.1 Model Architecture

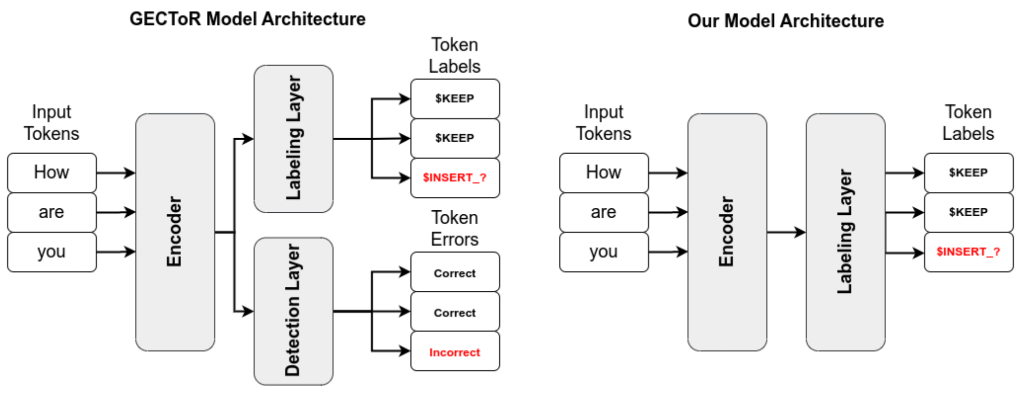


Figure 4.4: Comparison between GECToR's and our model architectures

Figure 4.4 shows the comparison between the sequence-to-label model architecture used by [3] and us. [3] experimented with pretrained BERT, RoBERTa and XLNet as the encoders for their GEC models, among which RoBERTa had a good trade-off between performance and inference speed. Therefore, we use the RoBERTa-base model as the encoder of our GEC model. To generate the output tokens in the post-processing stage, [3] used the token labels from the labelling layer and the token correctness confidences from the error detection layer. In contrast to that, our post-processing stage only uses token labels since we removed the error detection layer to ensure that the model can be easily fine-tuned using the RL algorithm. Besides this difference, we use the same 5k output labels used by [3].

Label Category	# of Labels
$KEEP	1
$DELETE	1
$APPEND	1,167
$REPLACE	3,802
$TRANSFORM	27
$MERGE	2
$UNKNOWN	1

Table 4.8: Token labels categories with the number of labels in each

Table 4.8 shows the label categories and the number of labels in each category. Please note that we have 5001 labels instead of 5k because we included the ”$UNKNOWN” label for any out-of-vocabulary (OOV) labels i.e. labels that are not present among the 5k labels. The source code of [3][3] implements AllenNLP[4] which handled the OOV labels automatically. The labels consist of basic transformations like $DELETE, $REPLACE_x and $APPEND_x to delete, replace or insert tokens in the sentence. Similarly, it consists of high-level transformations, that [3] called g-transformations, which include transformations like changing the case of the token, the tense of the verb or a noun from singular to plural and vice versa. The description of each label category is as follows:

- **$KEEP:** This label indicates that the token is correct and it does not need to be changed.
- **$DELETE:** This label removes the token from the sentence.
- **$APPEND:** These are very specific labels where $APPEND_x will insert the token ”x” after the current token.
- **$REPLACE:** Like the $APPEND labels, these labels are very specific labels where $REPLACE_x will replace the current token with ”x”.
- **$TRANSFORM_VERB:** These labels change the verb form of the token using a verb conjugation dictionary[5]. Each verb consists a total of 20 mapping pairs among the 5 different verb forms; base form (VB), past tense (VBD), gerund or present participle (VBG), past participle (VBN) and 3rd person singular present (VBZ). For instance, the $TRANSFORM_VERB_VB_VBD label will change the verb ”abandon” in the base form (VB) into its past tense (VBD) ”abandoned”.
- **$TRANSFORM_CASE:** These labels include operations to change the case of the token. Table 4.9 shows the verb transformation labels with example transformations.

Label	**Input Token**	**Output Token**
$TRANSFORM_CASE_LOWER	Tiger	tiger
$TRANSFORM_CASE_UPPER	Tiger	TIGER
$TRANSFORM_CASE_CAPITAL	tiger	Tiger
$TRANSFORM_CASE_CAPITAL_1	iphone	iPhone
$TRANSFORM_CASE_UPPER_-1	cds	CDs

Table 4.9: Verb Labels examples

- **$TRANSFORM_AGREEMENT:** These labels change the noun agreement from singular to plural and vice versa by simply adding and removing the suffix ”-s” from the current token. While this transformation does not work for irregular nouns like ”knife” whose

[3]GECToR source code: `https://github.com/grammarly/gector`
[4]AllenNLP is a library for NLP using Pytorch developed by Allen Institute of AI.
[5]Source: `https://github.com/gutfeeling/wordforms/blob/master/wordforms/en-verbs.txt`

plural is ”knives”, it is out of the scope of this thesis to investigate better noun agreement transformations. Therefore, we use the same methods from [3] to change the agreement of the token using the $TRANSFORM_AGREEMENT_SINGULAR and $TRANSFORM_AGREEMENT_PLURAL labels.

- **$TRANSFORM_SPLIT_HYPHEN:** This is a specific label that splits the current token into multiple tokens at all positions with a hyphen. An example of this transformation is changing the token ”out-of-the-box” into the tokens ”out”, ”of”, ”the” and ”box”.

- **$MERGE:** This category consists of $MERGE_SPACE and $MERGE_HYPHEN labels that combine the current and the next token into a single token as shown in Table 4.10.

Label	Input Tokens	Output Token
$MERGE_SPACE	every day	everyday
$MERGE_HYPHEN	cold blooded	cold-blooded

Table 4.10: Merge Labels examples

- **$UNKNOWN:** This is the OOV label used for any label that is not present in the set of labels.

4.3.2 Training Procedure

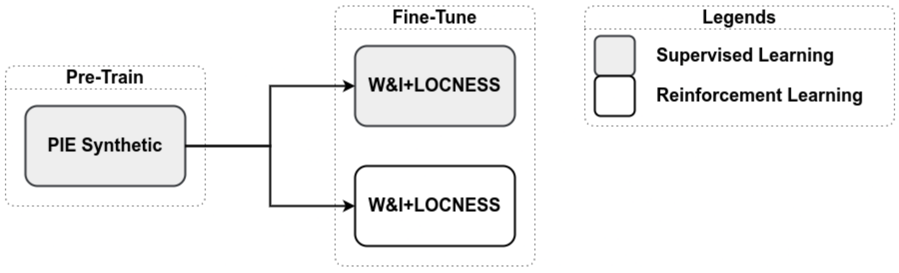


Figure 4.5: Pre-training and fine-tuning stages

As illustrated in Figure 4.5, we pre-train our GEC model on the PIE synthetic dataset using SL and then fine-tune the pre-trained model on the W&I+LOCNESS dataset using SL and RL methods. The pre-training stage adjusts the model weights to the token-labelling approach for the GEC task. Despite the huge amount of data, the grammar errors in the PIE synthetic dataset do not represent the actual kind of grammar errors. Therefore, we fine-tune the pre-trained model on the W&I+LOCNESS dataset which contains grammar errors from the writers of different proficiency levels in the English language. The fine-tuning stage adapts the pre-trained model to more authentic grammar errors. To investigate the difference in the performance of using SL and RL methods, we fine-tune the pre-trained model on the same dataset using different approaches. During the evaluation of models on benchmark datasets, the model outputs are generated using a maximum of 10 correction iterations.

For the SL training, we use methods from [3] to process the data into the GECToR format and their approach of training the model until convergence using the early-stopping strategy to terminate the training when the validation loss stops decreasing for a given number of epochs. The model with the lowest validation loss is evaluated on the benchmark datasets. For the RL training, our model interacts with our RL environment to gather experience for training and we evaluate the model periodically on the validation dataset using the corpus-level GLEU score. The model with the highest validation GLEU score is evaluated on the benchmark datasets. Details about the training hyperparameters are available in Appendix 8.2.

5 Implementation

The fine-tuning stage using RL requires an environment with which the policy can interact to gather experiences. A RL algorithm is used to adapt the policy on these experiences and to achieve the goal of the environment by maximizing the cumulative reward it receives through interactions with the environment. However, the reward function alone is not adequate to guide the policy in our GEC RL environment, given the large dimensionality of the state-action space. To enable safe and efficient exploration by the policy, we implemented an action search algorithm. This section will provide a detailed overview of the components used to fine-tune the policy using RL.

5.1 GEC RL Environment

To fine-tune our policy using RL, we created a custom GEC RL environment using OpenAI's gym[1] library, which provides a standard API for RL environment development. Our environment allows interaction with any GEC dataset processed in the JSON format as mentioned in Section 4.2. Please note that our environment utilizes only the training data from the GEC dataset since policy evaluation using the validation data is not part of the RL environment.

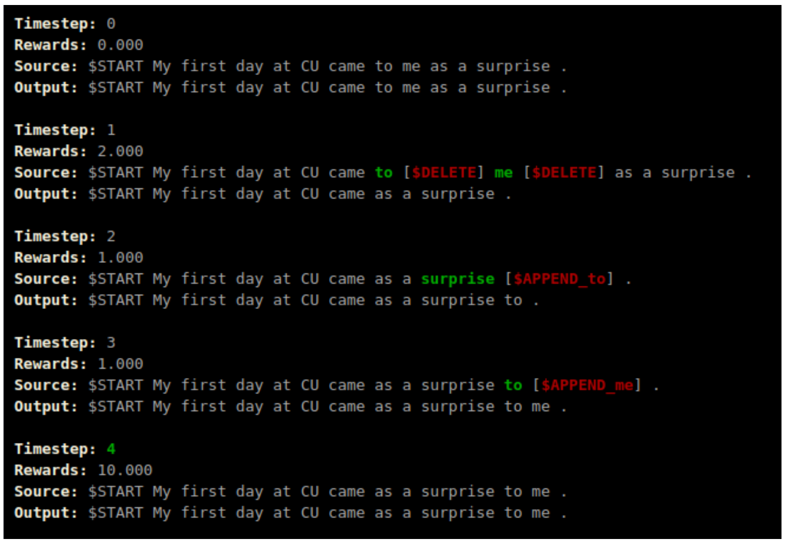
The datasets used in this thesis have only one reference sentence per test sentence. However, our environment is also able to accommodate datasets with multiple reference sentences per test sentence. For every episode, we randomly select an example that includes a test sentence and a set of reference sentences. Trying to correct any grammatical errors in the test sentence, the policy interacts with it for a maximum of 5 iterations, after which the episode ends. However, the episode can also terminate early under one of the following conditions:

1. **Unchanged tokens:** When the current state, s_t, and next state, s_{t+1}, have the same tokens, the state will not change anymore even if the policy keeps interacting with the environment for the remainder of the episode. Therefore, we end the episode and return the reward depending on whether the current state is grammatically correct or not.

2. **Low or high token count:** Under unfortunate circumstances, the policy might start modifying the sentences too much by either removing or inserting too many tokens. To discourage such behaviours, the episode will end early when the next state, s_{t+1}, has less than 3 tokens or more than 1.5 times the token count in the initial state, s_0.

To ensure that the policy can correct all sentences presented to it, the training dataset consists solely of solvable examples. This is to discourage the policy from getting trapped in a state where it is unable to continue making corrections, as a result of poor decisions made earlier in the training episode.

[1]OpenAI Gym: `https://www.gymlibrary.dev/`



```
Timestep: 0
Rewards: 0.000
Source: $START My first day at CU came to me as a surprise .
Output: $START My first day at CU came to me as a surprise .

Timestep: 1
Rewards: 2.000
Source: $START My first day at CU came to [$DELETE] me [$DELETE] as a surprise .
Output: $START My first day at CU came as a surprise .

Timestep: 2
Rewards: 1.000
Source: $START My first day at CU came as a surprise [$APPEND_to] .
Output: $START My first day at CU came as a surprise to .

Timestep: 3
Rewards: 1.000
Source: $START My first day at CU came as a surprise to [$APPEND_me] .
Output: $START My first day at CU came as a surprise to me .

Timestep: 4
Rewards: 10.000
Source: $START My first day at CU came as a surprise to me .
Output: $START My first day at CU came as a surprise to me .
```

Figure 5.1: Rendering of a sample episode of our GEC RL environment. **Timestep:** Current iteration of the episode. **Rewards:** Reward for the current interaction calculated using Algorithm 2. **Source:** Input tokens with action labels (in red) next to their respective token (in green). The keep action labels are not rendered. **Output:** Tokens after applying the actions.

5.2 Reward Function

In their paper, [5] used the GLEU score [26] as the reward function to fine-tune their GEC model using RL. However, using GLEU as the reward function has some drawbacks in GEC.

In this study, we will compare the GLEU score and Levenshtein Distance (LD) of two variants of the same sentence, one that is shorter and one that is longer. We will see how each of these metrics performs when applied to these two variants with the same kinds of errors.

Short Reference	Albert Einstein wrote his first scholarly paper at just 16 years old !
Long Reference	Albert Einstein , a German-born theoretical physicist , wrote his first scholarly paper at just 16 years old !

Table 5.1: Shorter and longer version of the same sentence.

For both sentence variants, we generated test sentences with the same type of errors as in Table 5.2, allowing us to evaluate how well each metric can capture the similarity between the correct and incorrect sentences based on the sentence length and error position. We calculated GLEU scores using the official scorer from `https://github.com/cnap/gec-ranking`. To calculate the token-level LD between the test and reference sentences, we used the RapidFuzz[2] Python library.

[2]RapidFuzz official repo: `https://github.com/maxbachmann/RapidFuzz`

Error Type	Description	Test Sentence
Type A	Remove terminal token[3]	Albert Einstein wrote his first scholarly paper at just 16 years old ~~!~~
Type B	Remove non-terminal token	Albert Einstein wrote ~~his~~ first scholarly paper at just 16 years old !

Table 5.2: Different error types based on the removed token's position in the sentence.

From Figure 5.2, it can be seen that the GLEU score tends to give higher scores to longer sentences with the same types of errors because longer sentences are more difficult to translate accurately. It also tends to favour errors that occur in the terminal tokens of a sentence, as these tokens have fewer n-grams when the GLEU score is calculated. In contrast, the LD score is relatively consistent across sentence lengths and error locations, indicating its robustness against these factors. Therefore, we use the decrease in LD as our reward function as shown in Algorithm 2.

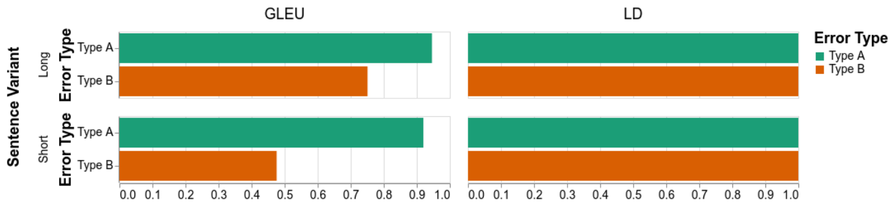


Figure 5.2: Score comparisons based on sentence length and error position

Algorithm 2 Reward Function

```
Input: $s_t$: Current Tokens, $a_t$: Action, $s_{t+1}$: Next Tokens, $s_{ref}$: Reference Tokens
Output: $reward$: Scalar Reward Value
 1: if $\neg(3 \leq len(s_{t+1}) \leq 1.5 \times len(s_0))$ then        ▷ Too low/high number of tokens
 2:     return -10
 3: if $s_t = s_{t+1}$ then                                            ▷ Current and next tokens are same
 4:     $all\_keep \leftarrow \forall_a \in a_t, a = KEEP\_INDEX$         ▷ Check if all actions are keep-action
 5:     if $s_{t+1} = s_{ref}$ and $all\_keep$ then                    ▷ Only keep-actions for correct tokens
 6:         $reward \leftarrow 10$
 7:     else                              ▷ Incorrect tokens and/or not all actions are keep-action
 8:         $reward \leftarrow 0$
 9: else
10:     $reward \leftarrow LevenshteinDistance(s_t, s_{ref}) - LevenshteinDistance(s_{t+1}, s_{ref})$
11: return $reward$
```

[3]By terminal tokens, we are referring to the first and last tokens in a sentence.

5.3 Action Search Algorithm

The high dimensionality of the state-action spaces leads to significant variance when sampling over all the actions using the policy. Since the world model of our GEC RL environment is quite simple and deterministic, we implement the action search algorithm to mitigate the variance by simulating the next tokens and filtering the actions to obtain a set of potentially good actions for each token. It not only prevents unsafe explorations but also implicitly incorporates Grammar Error Detection (GED)[4] by suggesting candidate labels only for incorrect tokens.

Algorithm 3 SearchActions

```
Input: $\pi_\theta$: Policy, $\mathcal{P}$: Transition Function, $s$: Current Tokens, $s_{ref}$: Reference Tokens
Output: $a$: Action for each token
1:  $a \leftarrow \emptyset$                    ▷ Initialize an empty array with same size as current tokens $s$
2:  $\underline{s} \leftarrow copy(s)$
3:  $d \leftarrow LevenshteinDistance(\underline{s}, s_{ref})$     ▷ Compute edit distance between $\underline{s}$ and $s_{ref}$
4:  $e \leftarrow EditOperations(\underline{s}, s_{ref})$ [5]     ▷ Compute set of edit operations to get $s_{ref}$ from $\underline{s}$
5:  for token index $i = len(\underline{s}), \ldots, 1$ do     ▷ Loop from the last to the first token
6:      $e_{tok} \leftarrow e[i]$                ▷ Edit type for the $i^{th}$ token of $\underline{s}$
7:      if $e_{tok} =$ "$equal$" then          ▷ The $i^{th}$ token of $\underline{s}$ does not need to change
8:          $a[i] = KEEP\_INDEX$            ▷ Assign the keep-action
9:      else
10:         $\hat{a} \leftarrow \textsc{GetCandidateActions}(\mathcal{P}, \underline{s}, s_{ref}, i, e_{tok}, d)$
11:         $a[i] \leftarrow \textsc{SampleTokenAction}(\pi_\theta, \underline{s}, i, \hat{a})$     ▷ Assign the sampled action
12:         $\underline{s} \leftarrow \mathcal{P}(\underline{s}, a[i], i)$          ▷ Update the $i^{th}$ token of $\underline{s}$
13:         $d \leftarrow LevenshteinDistance(\underline{s}, s_{ref})$   ▷ Update edit distance between $\underline{s}$ and $s_{ref}$
14: return $a$
```

The first part of our action search algorithm is the process of finding a set of potential candidate actions for a token that can move the current tokens closer to the reference tokens by decreasing the Levenshtein distance between them. A simple version of this method involves applying all actions to a token and collecting the actions that result in tokens with a lower Levenshtein distance to the reference tokens than the Levenshtein distance between the original tokens and the reference tokens. Our method optimizes this approach by using a sequence matcher[6] to find the non-matching tokens between the current and reference tokens and get the type of edit required to change each token into the tokens in the reference sentence as shown in Figure 5.3. We cluster the actions into the four edit types: *equal*, *delete*, *insert*, and *replace* as shown in Table 5.3. This edit-to-action mapping reduces the number of actions to search for the candidate actions based on the edit type of a token.

[4]In GED, the objective is to only detect tokens with grammar errors.
[6]We used `SequenceMatcher` from the standard `difflib` Python library to identify the edit types.

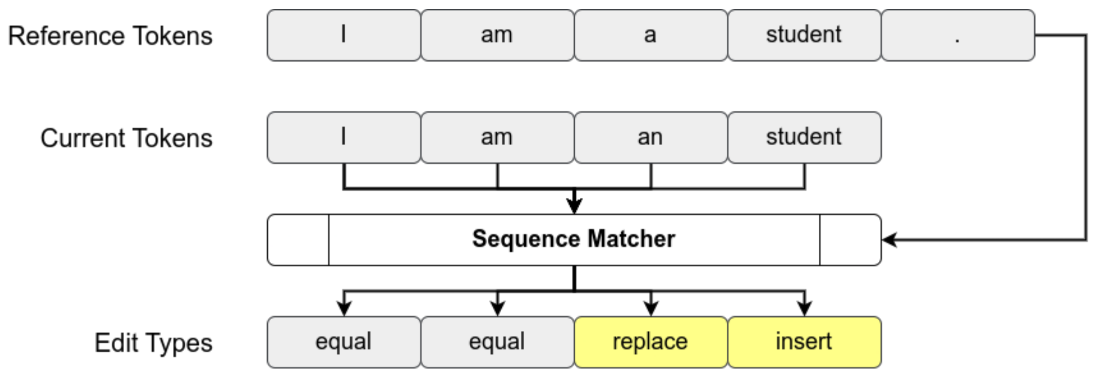


Figure 5.3: Generating edit types for each token using Sequence Matcher.

Edit Type	Action Labels
equal	$KEEP
delete	$DELETE
insert	$APPEND
replace	$REPLACE $TRANSFORM $MERGE $UNKNOWN

Table 5.3: Edit type to action labels mapping

Algorithm 4 GetCandidateActions

```
Input: $\mathcal{P}$: Transition Function, $s$: Current Tokens, $s_{ref}$: Reference Tokens, $i_{token}$: Token
    Index, $e_{token}$: Token Edit Type, $d_{ref}$: Current Edit Distance
Output: $\hat{a}_t$: Candidate Actions
1: $\hat{a} \leftarrow \emptyset$                                    ▷ Initialize empty set of candidate actions
2: $a_{edit} \leftarrow \mathit{Edit2Actions}(e_{token})$            ▷ Edit type to action mapping as in Table 5.3
3: for $a \in a_{edit}$ do
4:     $\bar{s} \leftarrow \mathcal{P}(s, a, i_{token})$              ▷ Get $\bar{s}$ by applying action $a$ at $i^{th}$ token of $s$
5:     $\bar{d} \leftarrow \mathit{LevenshteinDistance}(\bar{s}, s_{ref})$
6:     if $\bar{d} < d_{ref}$ then                                     ▷ Between $s$ and $\bar{s}$, $\bar{s}$ is closer to $s_{ref}$
7:         $\hat{a} \leftarrow \hat{a} \cup \{a\}$
8: return $\hat{a}$
```

As outlined in Algorithm 5, we choose the action for each token by sampling from a set of candidate actions rather than sampling from all actions. If a token has no candidate actions, it is assigned the $UNKNOWN action, indicating that the policy has modified that token and it could not be recovered anymore using any other actions. Conversely, if there are candidate actions, we add the $KEEP action to the candidate actions and select one of them randomly using the policy's output values as the probability weights for the sampling. This allows the policy to retain errors until it is confident in correcting them, rather than forcing it to fix all present errors at once.

Algorithm 5 SampleTokenAction

```
Input: $\pi_\theta$: Policy, $s_t$: Current Tokens, $i_{token}$: Token Index, $\hat{a}_t$: Candidate Actions
Output: $a$: Token Action
1: if $len(\hat{a}_t) = 0$ then
2:     return $UNKNOWN\_INDEX$                          ▷ Index of the UNKNOWN label
3: $\hat{a}_t \leftarrow \hat{a}_t \cup \{KEEP\_INDEX\}$      ▷ Include keep-action in candidate actions
4: $O \leftarrow \pi(s_t)$                                   ▷ Predict using the policy
5: $O_{candidate} \leftarrow O[i_{token}, \hat{a}_t]$        ▷ Candidate labels' raw outputs of the $i^{th}$ token
6: $p_{candidate} \leftarrow softmax(O_{candidate})$         ▷ Probability distribution over candidate actions
7: $a \leftarrow random(\hat{a}_t, p_{candidate})$           ▷ Weighted sampling over candidate actions
8: return $a$
```

As illustrated in Figure 5.4, our action search algorithm assists the policy by filtering the action options from all 5k actions to just a few candidate actions. In the example in Figure 5.4, all the non-keep candidate actions are equally good because they reduce the LD by 2. However, the action "$REPLACE_a" will transform the current sentence into "I am a .", which is difficult to correct in the next iteration. Since only solvable sentences are provided in this stage, only such a faulty correction by the policy will make sentences unsolvable. Therefore, the policy must learn to avoid such situations through experience by decreasing the confidence of such labels in the action sampling stage.

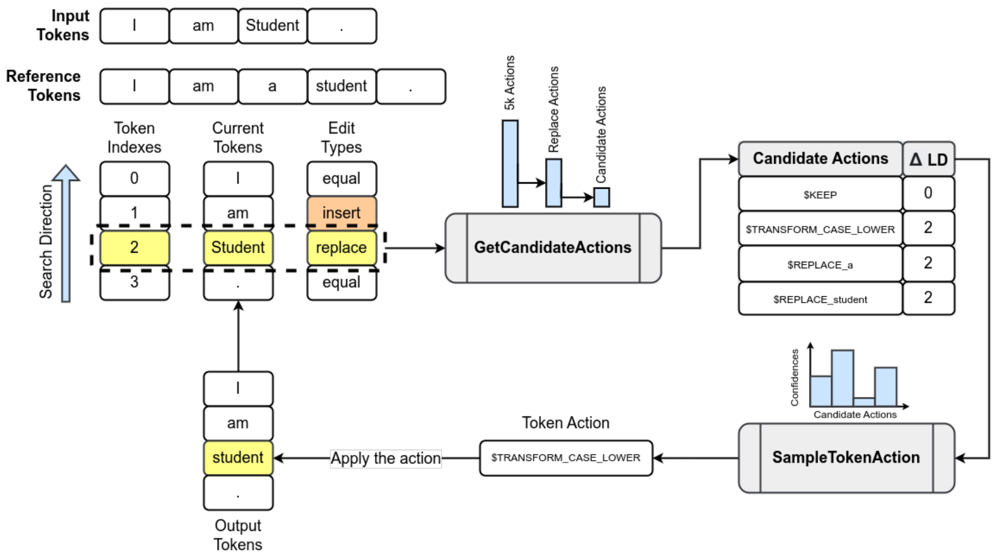


Figure 5.4: Action Search Algorithm

5.4 REINFORCE Algorithm

To fine-tune our policy using RL, we implemented the REINFORCE (REward Increment = Non-negative Factor times Offset REinforcement times Characteristic Eligibility) algorithm[69]. It is a policy-gradient algorithm that directly optimizes the policy in the direction that increases the expected reward. It achieves this by performing gradient-descent on the estimated gradient of the expected reward with respect to the policy parameters as depicted in Equation 5.1.

$$\nabla_\theta \mathcal{J} = -\frac{1}{N}\sum_{t=1}^{N} G_t \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{5.1}$$

where
$\nabla_\theta \mathcal{J}$ is the estimated gradient of the expected reward w.r.t the policy parameters θ,
G_i is the discounted return at the i^{th} timestep from Equation 3.8

In our task, the state, s_t, is an array of tokens whose dimension is in the range $[1, M]$ and its action, a_t, is also an array of the same dimension. Therefore, Equation 5.1 is adapted as Equation 5.2 which computes the gradient over the sum of the log probability of token action.

$$\nabla_\theta \mathcal{J} = -\frac{1}{N}\sum_{t=1}^{N} G_t \nabla_\theta \sum_{i=1}^{M} \log \pi_\theta(a_{t,i}|s_{t,i}) \tag{5.2}$$

We implemented the batched version of the REINFORCE algorithm that updates the policy on minibatches sampled from the experiences collected from multiple episodes. We accumulate the gradients from the minibatches and update the policy parameters after all experiences have been sampled. Therefore, the number of gradient accumulations in our implementation depends on the total number of experiences collected.

Algorithm 6 Batched REINFORCE Algorithm

```
1:  Initialize policy with random parameters $\theta$
2:  Initialize gradient buffer $\mathbf{g} \leftarrow 0$
3:  Initialize experience buffer $\mathcal{D} \leftarrow \emptyset$
4:  for episode $k = 1, 2, \dots$ do
5:      $\tau \leftarrow \{(s_t, a_t, r_t, s'_t)\}_{t=1}^{T}$        ▷ Sample episodic trajectory using Algorithm 3
6:      $\tau_G \leftarrow \{G_t\}_{t=1}^{T}$        ▷ Compute discounted returns using Equation 3.8
7:      $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, G_t) \mid (s_t, a_t, r_t, s'_t) \in \tau, G_t \in \tau_G\}_{t=1}^{T}$        ▷ Update experience buffer
8:      if $k \bmod K = 0$ then        ▷ Update every $K^{th}$ episode
9:          $n \leftarrow \lceil len(\mathcal{D})/N \rceil$        ▷ Calculate number of mini-batches from the buffer
10:         for mini-batch $\{(s_i, a_i, G_i)\}_{i=1}^{N} \in \mathcal{D}$ do        ▷ Sample mini-batches of N items
11:             $\mathbf{g} \leftarrow \mathbf{g} + \frac{1}{n}\nabla_\theta \mathcal{J}$        ▷ Accumulate gradients computed using Equation 5.2
12:         $\theta \leftarrow \theta + \alpha \mathbf{g}$        ▷ Update the policy parameters
13:         $\mathcal{D} \leftarrow \emptyset$        ▷ Reset experience buffer
14:         $\mathbf{g} \leftarrow 0$        ▷ Reset gradient buffer
```

6 Evaluation

In this section, we compare the performance of our models against other GEC systems that have achieved the best results on different GEC benchmarks at the time this thesis was written. We do not include the results of ensemble models[1] to make a reasonable comparison between our and others' GEC models. Determining the current state-of-the-art (SOTA) GEC system in each benchmark is difficult because the platforms[2][3] keeping track of the GEC systems are public and the papers are not consistently tracked across all of these platforms. As a result, we emphasize that the GEC systems chosen for evaluation may not necessarily reflect the current SOTA GEC systems. We evaluate the *best model* from our baseline and RL models using the validation metrics mentioned in Section 4.3.2.

6.1 Benchmarks

To highlight the strengths and limitations of different GEC systems and promote their further development, several GEC benchmarks have been introduced. Each of them evaluates the GEC system using different datasets and metrics. Therefore, it is important to evaluate any GEC system on multiple benchmarks to compare it with different existing approaches and to evaluate the effectiveness of the implemented approach. In this section, we will discuss the GEC benchmarks used to evaluate our approaches in this thesis.

6.1.1 CoNLL-2014

Computational Natural Language Learning in 2014 (CoNLL-2014) benchmark[29] consists of 50 essays written by 25 non-native English speakers on 2 different topics. Errors in each essay were annotated by 2 independent annotators. Over the previous CoNLL-2013[66] benchmark, CoNLL-2014 introduced further changes such as the detection and correction of all 28 grammar errors instead of just 5 grammar errors and the use of $F_{0.5}$ score by the M2 scorer instead of F_1 score. $F_{0.5}$ is used since it emphasizes precision twice as much as recall. GEC system with high precision is preferred because inaccurate error detection is undesired over missing some errors.

6.1.2 JFLEG

Johns Hopkins University FLuency-Extended GUG corpus (JFLEG)[70] benchmark was introduced extending the "Grammatical" versus "UnGrammatical" (GUG) corpus[71] to evaluate

[1]Ensemble models combine the outputs of multiple diverse models to make predictions
[2]NLP Progress: `https://nlpprogress.com/english/grammaticalerrorcorrection.html`
[3]Papers With Code: `https://paperswithcode.com/task/grammatical-error-correction`

GEC systems with a focus on fluency-oriented corrections that make the corrected sentences sound more native. It is a parallel corpus consisting of 754 and 747 sentences in the validation and test datasets respectively. Each sentence in the dataset has been corrected by 4 different annotators and the corpus-level GLEU score is used as the evaluation metric.

Original	they just creat impression such well that people are drag to buy it .
Minimal edit	They just create an impression so well that people are dragged to buy it .
Fluency edit	They just create such a good impression that people are compelled to buy it .

Table 6.1: Difference between minimal and fluency edits

6.1.3 BEA-2019

The Building Educational Applications (BEA) 2019 Shared Task introduced the W&I+LOCNESS dataset whose test dataset (see Table 4.1) is used as the benchmark dataset. It consists of 350 essays on approximately 50 different topics written by 334 authors consisting of both native and non-native English speakers. It is the largest GEC benchmark at the moment with essays from different proficiency levels. It uses the ERRANT $F_{0.5}$ for the evaluation metrics in order to provide detailed feedback about the GEC system's performance on different error categories.

Benchmark	**# of Sentences**	**Metrics**
CoNLL-2014	1,381	M2 $F_{0.5}$
JFLEG (test)	747	GLEU
BEA-2019	4,477	ERRANT $F_{0.5}$

Table 6.2: Comparison of GEC benchmarks

6.2 Results

On the JFLEG benchmark, [5] showed that their RL model outperformed their SL model. [22] demonstrated that incorporating the BERT's representation and the output of a BERT fine-tuned as a GED model as the additional input features can benefit sequence-to-sequence GEC systems. [72] introduced a Neural Verification Network (VERNet) to effectively estimate the token level quality from multiple hypothesises generated from sequence-to-sequence GEC systems. [21] introduced gT5, the GEC version of the mT5[73][4], by performing multilingual GEC in English, Czech, German and Russian. Using the large gT5-xxl model, they generated a new variant of Lang-8[68] GEC dataset called cLang-8, whose target sentences are the output sentences of the gT5-xxl model. [21] showed that just fine-tuning on the cLang-8 dataset can substitute the typical multi-stage training of GEC systems.

[4]mT5 is the multilingual version of T5 (Text-To-Text Transfer Transformer)[74].

Model	**BEA-2019 ($F_{0.5}$)**	**CONLL-2014 ($F_{0.5}$)**	**JFLEG test (GLEU)**
Neural RL Model [5]	-	-	53.98
BERT-fuse GED [22]	65.6	62.6	61.3
ELECTRA-VERNet [72]	68.77	63.43	**62.07**
GECToR (RoBERTa-base) [3]	71.5	64.0	-
T5-base [21]	69.38	65.05	-
T5-xxl [21]	**75.88**	**68.75**	-
Baseline Model (our)	60.38 ± 0.41	61.65 ± 0.24	60.14 ± 0.20
RL Model (our)	64.00 ± 0.89	63.62 ± 0.39	57.88 ± 0.43

Table 6.3: Comparison of GEC systems on different benchmarks. Our model scores are the mean and standard deviation from five experiments.

According to the results of our experiments in Table 6.3, our RL model outperforms our baseline model on the BEA-2019 and CONLL-2014 benchmarks. The RL model has the advantage of interacting with sentences during training, which allows it to see new sentences that are not present in the dataset and adapt to them. In contrast, the SL model only sees the sentences present in the training dataset. However, the baseline model shows better performance on the JFLEG test benchmark, which also evaluates the fluency of the generated texts. This indicates that the outputs from the RL model are not as fluent as those from the SL model and the RL model's ability to adapt to new sentences comes at the expense of fluency. Despite having a smaller fine-tuning dataset, our RL model's performance on some benchmarks is comparable to other GEC methods. These results highlight the strengths and weaknesses of both models on different benchmarks.

The ERRANT scorer of the BEA-2019 benchmark generates a very detailed report over its 24 error categories. From Figure 6.1, we can see that on average the RL model has higher precision and $F_{0.5}$ scores than the baseline model while the baseline model has higher recall than the RL model. From these scores, we can also identify the error categories where the baseline and RL models struggled the most.

For instance, the $F_{0.5}$ scores of both models are very low in the contraction category, CONTR, which includes transforming contractions like ”n't” and ”'m” into their full forms ”not” and ”am” respectively. Investigating the input and output sentences for contractions revealed that out of 613 input sentences with contractions, there were still 484 output sentences which contained contractions. Table 6.4 shows the number of different contractions present in one of the RL model's outputs in the BEA 2019 benchmark. Please note that number for ”'s” is high because ”'s” can also be a part of a possessive noun as in Table 6.5. This highlighted the limitation of our data processing pipeline which did not process the contractions properly.

Similarly, the RL models' $F_{0.5}$ scores have a huge fluctuation in the verb inflection category, VERB : INFL, indicating an unusual difference in their ability to correct verb errors. We do not know the cause because verb inflection errors are harder to detect in the model outputs than contraction errors. Since the gold annotations of the BEA-2019 test dataset are not publicly available, we cannot know the difference between the model outputs and the correct sentences.

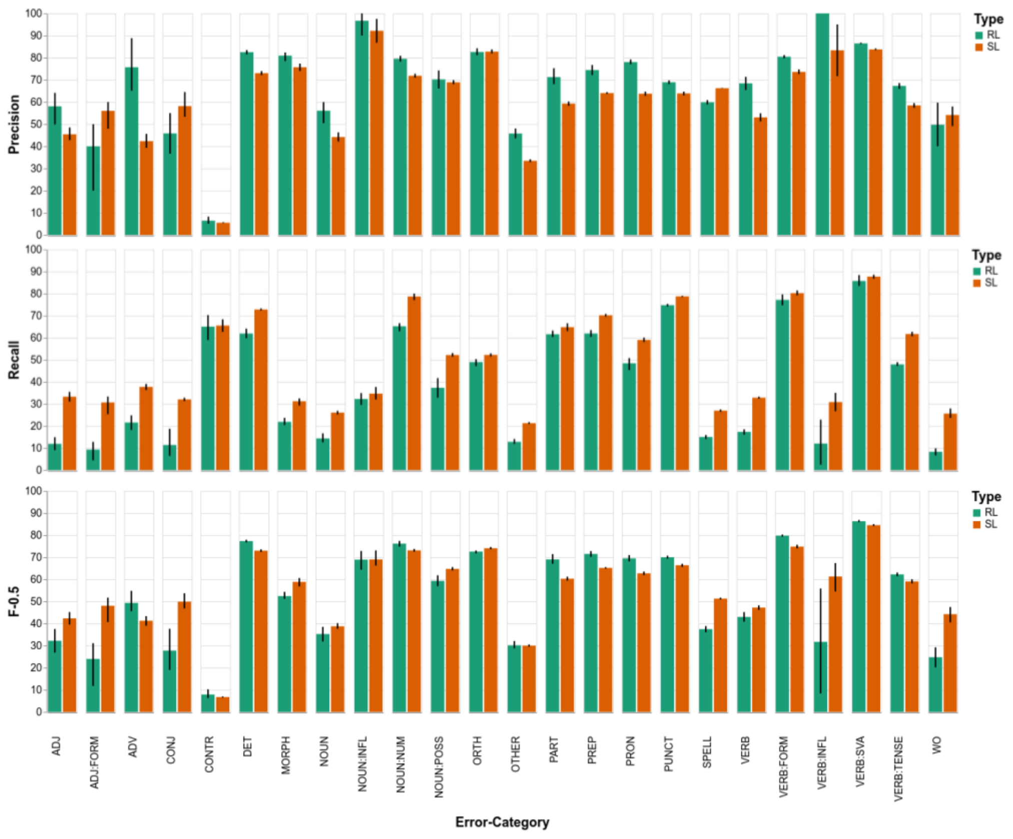


Figure 6.1: Mean Precision, Recall and $F_{0.5}$ with the 95% confidence interval of the baseline and RL models in the BEA-2019 benchmark

Contraction	**# of Tokens**
’s	298
n’t	138
’m	55
’ll	27
’re	17
’ve	14
’d	11

Table 6.4: Number of contractions present in the model output of the BEA-2019 benchmark

’s as contraction	Not only that , he ’s a responsible and reliable guy .
’s in possessive noun	The philosopher ’s stone was pulverized into dust .

Table 6.5: Usage of ”’s” as contraction and in possessive noun

7 Conclusion and Future Works

In this thesis, we developed the GEC RL environment using a Levenshtein Distance based reward function. The parallel data used by our environment does not require manually labelling the error labels in the training data. Therefore, it removes the possibility to hinder the sequence-to-label GEC model's performance due to labelling issues in the training data. We also implemented the action search method to mitigate the issues of high-dimensional state-action spaces in RL. In this way, we present the potential benefit of fine-tuning a sequence-to-label GEC model using RL. In future works, the following directions would be interesting:

1. Since RL training is sensitive to hyperparameters and requires more training to converge, we used a smaller dataset and a relatively simple RL algorithm in this thesis to quickly examine our ideas. Therefore, the next promising step would be to fine-tune the model on larger GEC datasets using more advanced RL algorithms like Proximal Policy Optimization (PPO)[47].

2. In this thesis, we pre-train the model using SL. However, pre-training and fine-tuning the model using only RL can mitigate the limitations of manual data labelling in SL and take full advantage of RL's ability to learn from its own experience.

3. One of the bottleneck of using the sequence-to-label GEC model is a large number of very specific edit labels. Optimizing and reducing the number of predefined labels required by the sequence-to-label model can improve its performance.

4. Our action search algorithm is quite simple and it does not benefit much from test sentences without any errors since it will always generate keep labels for them. Therefore, a better search algorithm like Monte-Carlo Tree Search (MCTS)[75] can bring further improvements.

8 Appendix

8.1 Data Filtering Parameters

This section discusses the data filtering parameters used while converting the data into JSON format. The command to convert the W&I+LOCNESS dataset from its M2 format to our JSON format is as follows:

```
python m2_to_json.py \
--m2_path M2_PATH \         # Path to the input M2 file
--json_path JSON_PATH \     # Path to the output JSON file
--min_len MIN_LEN \         # Minimum number of tokens in a sentence
...
--remove_ellipsis           # Remove ellipsis from the sentences
```

To convert the PIE synthetic dataset into the JSON format, we created a Jupyter Notebook file ”notebooks/PIE_to_JSON.ipynb”. The notebook file is specifically created to process and filter the parallel texts from the PIE synthetic dataset using the same hyperparameters used to convert M2 files into JSON files. It also splits the data into train-validation datasets.

Hyperparameter	PIE Synthetic	W&I+LOCNESS	Description
min_len	5		Minimum number of tokens in a sentence
max_len	50		Maximum number of tokens in a sentence
min_sim	0.8		Minimum similarity between source and target sentences
only_proper_sent	True		Allow examples with only proper target sentences
spell_check	False	True	Check and correct spelling errors in source and target sentences
remove_ellipsis	True		Remove ellipsis from source and target sentences

Table 8.1: Data Filtering Hyperparameters

8.2 Training Details

The hyperparameters used in our experiments are saved in YAML format under the subdirectory `configs` which can be used to reproduce our results using the following commands:

```
python train_sl.py configs/sl_pretrain.yaml   # SL Pre-Training
python train_sl.py configs/sl_finetune.yaml   # SL Fine-Tuning
python train_rl.py configs/rl_finetune.yaml   # RL Fine-Tuning
```

For our experiments, we used a cloud system with 52 GB RAM and NVIDIA T4 GPU to pre-train and fine-tune our model, which uses RoBERTa-base as the encoder. To optimize the utilization of our hardware resources, we employed Pytorch's gradient accumulation and automatic mixed precision techniques. Using gradient accumulation, we could train our model on bigger batch sizes by adding the gradients of several mini-batches. Likewise, automatic mixed precision further reduces the GPU requirements and increases training speed by casting the floating point operations into a low-precision float-16 format. Pytorch's implementation of automatic mixed precision attempts to mitigate the potential loss of accuracy due to lower precision floating operations by only casting certain regions that can work with float-16 operations.

Table 8.2 shows approximate training time for different stages of our experiment. The result for the pre-training is only from one experiment whereas the fine-tuning results are the mean time of 5 experiments.

Stage	**Average training time**
Pre-Training	47 hours
SL Fine-Tuning	40 mins
RL Fine-Tuning	24 hours

Table 8.2: Approximate training time for different stages

[2]We use the term "cold" epochs to indicate the training epochs when the transformer encoder weights are frozen so that only the final layers are trained. So "warm" epochs are the training epochs when both the transformer encoder and the final layers are fine-tuned.

[2]A sentence is solvable if it can be converted into its reference sentence using the actions possible by the sequence-to-label model.

Hyperparameter	**Pre-Train**	**Fine-Tune**	**Description**
Dropout	0.1		Dropout Rate before the final layer
Cold epochs	2	0	Number of cold epochs[1]
Total epochs	20		Maximum number of training epochs
Optimizer	Adam[76]		Gradient-based optimizer
Learning rate (cold epochs)	1.0×10^{-3}		Optimizer learning rate in cold epochs
Learning rate (warm epochs)	1.0×10^{-5}		Learning rate in warm epochs
Patience	1 epoch	5 epochs	Number of epochs without improvement to terminate the training before total epochs
Batch size	128	320	Training batch size
Accumulation Size	2	5	Number of iterations to accumulate gradient
Datasets	PIE Synthetic	W&I+LOCNESS	Training datasets
Keep Correct Examples	False	True	Training dataset contains already correct sentences if set to ”True”. Otherwise, it only contains sentences with errors.
Only Solvable Examples	False	True	Training dataset contains only solvable sentences[2]if set to ”True”. Otherwise, it contains all sentences

Table 8.3: SL pre-training and fine-tuning hyperparameters

Hyperparameter	**Fine-Tune**	**Description**
Gamma (γ)	0.95	Discount factor in Equation 3.8
Dropout	0.1	Dropout rate before the final layer
Optimizer	Adam	Gradient-based optimizer
Learning rate	1.0×10^{-5}	Optimizer learning rate
Episodes	1×10^{6}	Total number of episodes
Batch size	64	Mini-batch size
Update interval	200 Episodes	Interval to update the model using RL algorithm
Evaluation interval	1×10^{4} Episodes	Interval to evaluate the model
Environment ID	gec_lev_dist-v1	Environment id for the GEC environment with particular reward function
Datasets	W&I+LOCNESS	Datasets to load in the GEC environment
Only Solvable Examples	True	Training dataset contains only solvable sentences if set to ‘True‘. Otherwise, it contains all sentences

Table 8.4: RL fine-tuning Hyperparameters

Bibliography

[1] R. K. Rana, "Grammar error correction using deep reinforcement learning," M.S. thesis, in English, Faculty of Applied Computer Science, Deggendorf Institute of Technology, Deggendorf, Germany and Faculty of Science, University of South Bohemia, České Budějovice, Czech republic, Jan. 2023, p. 44.

[2] A. Awasthi, S. Sarawagi, R. Goyal, S. Ghosh, and V. Piratla, *Parallel iterative edit models for local sequence transduction*, 2019. DOI: 10.48550/ARXIV.1910.02893. [Online]. Available: https://arxiv.org/abs/1910.02893.

[3] K. Omelianchuk, V. Atrasevych, A. Chernodub, and O. Skurzhanskyi, "GECToR – grammatical error correction: Tag, not rewrite," in *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, Seattle, WA, USA → Online: Association for Computational Linguistics, Jul. 2020, pp. 163–170. DOI: 10.18653/v1/2020.bea-1.16. [Online]. Available: https://aclanthology.org/2020.bea-1.16.

[4] V. Uc-Cetina, N. Navarro-Guerrero, A. Martin-Gonzalez, C. Weber, and S. Wermter, "Survey on reinforcement learning for language processing," *Artificial Intelligence Review*, Jun. 2022. DOI: 10.1007/s10462-022-10205-5.

[5] K. Sakaguchi, M. Post, and B. Van Durme, "Grammatical error correction with neural reinforcement learning," in *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, Taipei, Taiwan: Asian Federation of Natural Language Processing, Nov. 2017, pp. 366–372. [Online]. Available: https://aclanthology.org/I17-2062.

[6] E. Malmi, S. Krause, S. Rothe, D. Mirylenka, and A. Severyn, *Encode, tag, realize: High-precision text editing*, 2019. DOI: 10.48550/ARXIV.1909.01187. [Online]. Available: https://arxiv.org/abs/1909.01187.

[7] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, 2014. DOI: 10.48550/ARXIV.1412.3555. [Online]. Available: https://arxiv.org/abs/1412.3555.

[8] A. Bakhtin, A. Szlam, M. Ranzato, and E. Grave, *Lightweight adaptive mixture of neural and n-gram language models*, 2018. DOI: 10.48550/ARXIV.1804.07705. [Online]. Available: https://arxiv.org/abs/1804.07705.

[9] C. Chelba, M. Norouzi, and S. Bengio, *N-gram language modeling using recurrent neural network estimation*, 2017. DOI: 10.48550/ARXIV.1703.10724. [Online]. Available: https://arxiv.org/abs/1703.10724.

[10] E. Miller, D. Shen, J. Liu, and C. Nicholas, "Performance and scalability of a large-scale n-gram based information retrieval system," *Journal of Digital Information; Vol 1, No 5 (2000)*, vol. 1, Jan. 2000.

[11] M. Federico and M. Cettolo, "Efficient handling of n-gram language models for statistical machine translation," in *Proceedings of the Second Workshop on Statistical Machine Translation*, Prague, Czech Republic: Association for Computational Linguistics, Jun. 2007, pp. 88–95. [Online]. Available: `https://aclanthology.org/W07-0712`.

[12] W. Cavnar and J. Trenkle, "N-gram-based text categorization," *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval*, May 2001.

[13] F. Aisopos, G. Papadakis, and T. Varvarigou, "Sentiment analysis of social media content using n-gram graphs," in *Proceedings of the 3rd ACM SIGMM international workshop on Social media*, 2011, pp. 9–14.

[14] G. Sidorov, A. Gupta, M. Tozer, D. Catala, A. Catena, and S. Fuentes, "Rule-based system for automatic grammar correction using syntactic n-grams for English language learning (L2)," in *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 96–101. [Online]. Available: `https://aclanthology.org/W13-3613`.

[15] J. C. Park, M. Palmer, and C. Washburn, "An English grammar checker as a writing aid for students of English as a second language," in *Fifth Conference on Applied Natural Language Processing: Descriptions of System Demonstrations and Videos*, Washington, DC, USA: Association for Computational Linguistics, Mar. 1997, pp. 24–24. DOI: `10.3115/974281.974296`. [Online]. Available: `https://aclanthology.org/A97-2014`.

[16] N.-R. HAN, M. CHODOROW, and C. LEACOCK, "Detecting errors in english article usage by non-native speakers," *Natural Language Engineering*, vol. 12, no. 2, pp. 115–129, 2006. DOI: `10.1017/S1351324906004190`.

[17] A. Rozovskaya, K.-W. Chang, M. Sammons, D. Roth, and N. Habash, "The Illinois-Columbia system in the CoNLL-2014 shared task," in *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, Baltimore, Maryland: Association for Computational Linguistics, Jun. 2014, pp. 34–42. DOI: `10.3115/v1/W14-1704`. [Online]. Available: `https://aclanthology.org/W14-1704`.

[18] Z. Kaili, C. Wang, R. Li, Y. Liu, T. Hu, and H. Lin, *A simple but effective classification model for grammatical error correction*, 2018. DOI: `10.48550/ARXIV.1807.00488`. [Online]. Available: `https://arxiv.org/abs/1807.00488`.

[19] C. Brockett, W. B. Dolan, and M. Gamon, "Correcting ESL errors using phrasal SMT techniques," in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia: Association for Computational Linguistics, Jul. 2006, pp. 249–256. DOI: `10.3115/1220175.1220207`. [Online]. Available: `https://aclanthology.org/P06-1032`.

[20] M. Felice, Z. Yuan, Ø. E. Andersen, H. Yannakoudakis, and E. Kochmar, "Grammatical error correction using hybrid systems and type filtering," in *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, Baltimore, Maryland: Association for Computational Linguistics, Jun. 2014, pp. 15–24. DOI: 10.3115/v1/W14-1702. [Online]. Available: https://aclanthology.org/W14-1702.

[21] S. Rothe, J. Mallinson, E. Malmi, S. Krause, and A. Severyn, "A simple recipe for multilingual grammatical error correction," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, Online: Association for Computational Linguistics, Aug. 2021, pp. 702–707. DOI: 10.18653/v1/2021.acl-short.89. [Online]. Available: https://aclanthology.org/2021.acl-short.89.

[22] M. Kaneko, M. Mita, S. Kiyono, J. Suzuki, and K. Inui, "Encoder-decoder models can benefit from pre-trained masked language models in grammatical error correction," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online: Association for Computational Linguistics, Jul. 2020, pp. 4248–4254. DOI: 10.18653/v1/2020.acl-main.391. [Online]. Available: https://aclanthology.org/2020.acl-main.391.

[23] S. Kiyono, J. Suzuki, M. Mita, T. Mizumoto, and K. Inui, "An empirical study of incorporating pseudo data into grammatical error correction," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 1236–1242. DOI: 10.18653/v1/D19-1119. [Online]. Available: https://aclanthology.org/D19-1119.

[24] Y. Wu, M. Schuster, Z. Chen, *et al.*, *Google's neural machine translation system: Bridging the gap between human and machine translation*, 2016. DOI: 10.48550/ARXIV.1609.08144. [Online]. Available: https://arxiv.org/abs/1609.08144.

[25] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, Jul. 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. [Online]. Available: https://aclanthology.org/P02-1040.

[26] C. Napoles, K. Sakaguchi, M. Post, and J. Tetreault, "Ground truth for grammatical error correction metrics," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, Beijing, China: Association for Computational Linguistics, Jul. 2015, pp. 588–593. DOI: 10.3115/v1/P15-2097. [Online]. Available: https://aclanthology.org/P15-2097.

[27] C. Napoles, K. Sakaguchi, M. Post, and J. Tetreault, *Gleu without tuning*, 2016. DOI: 10.48550/ARXIV.1605.02592. [Online]. Available: https://arxiv.org/abs/1605.02592.

[28] D. Dahlmeier and H. T. Ng, "Better evaluation for grammatical error correction," in *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Montréal, Canada: Association for Computational Linguistics, Jun. 2012, pp. 568–572. [Online]. Available: `https://aclanthology.org/N12-1067`.

[29] 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 *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, Baltimore, Maryland: Association for Computational Linguistics, Jun. 2014, pp. 1–14. DOI: `10.3115/v1/W14-1701`. [Online]. Available: `https://aclanthology.org/W14-1701`.

[30] C. Bryant, M. Felice, and T. Briscoe, "Automatic annotation and evaluation of error types for grammatical error correction," in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 793–805. DOI: `10.18653/v1/P17-1074`. [Online]. Available: `https://aclanthology.org/P17-1074`.

[31] M. Felice, C. Bryant, and T. Briscoe, "Automatic extraction of learner errors in ESL sentences using linguistically enhanced alignments," in *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 825–835. [Online]. Available: `https://aclanthology.org/C16-1079`.

[32] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *ArXiv*, vol. abs/1810.04805, 2019.

[33] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2017. DOI: `10.48550/ARXIV.1706.03762`. [Online]. Available: `https://arxiv.org/abs/1706.03762`.

[34] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, *Squad: 100,000+ questions for machine comprehension of text*, 2016. DOI: `10.48550/ARXIV.1606.05250`. [Online]. Available: `https://arxiv.org/abs/1606.05250`.

[35] P. Rajpurkar, R. Jia, and P. Liang, *Know what you don't know: Unanswerable questions for squad*, 2018. DOI: `10.48550/ARXIV.1806.03822`. [Online]. Available: `https://arxiv.org/abs/1806.03822`.

[36] Y. Liu, M. Ott, N. Goyal, *et al.*, *Roberta: A robustly optimized bert pretraining approach*, 2019. DOI: `10.48550/ARXIV.1907.11692`. [Online]. Available: `https://arxiv.org/abs/1907.11692`.

[37] M. Joshi, D. Chen, Y. Liu, D. S. Weld, L. Zettlemoyer, and O. Levy, *Spanbert: Improving pre-training by representing and predicting spans*, 2019. DOI: `10.48550/ARXIV.1907.10529`. [Online]. Available: `https://arxiv.org/abs/1907.10529`.

[38] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, *Xlnet: Generalized autoregressive pretraining for language understanding*, 2019. DOI: `10.48550/ARXIV.1906.08237`. [Online]. Available: `https://arxiv.org/abs/1906.08237`.

[39] G. Lample and A. Conneau, *Cross-lingual language model pretraining*, 2019. DOI: 10.48550/ARXIV.1901.07291. [Online]. Available: https://arxiv.org/abs/1901.07291.

[40] M. Ott, S. Edunov, D. Grangier, and M. Auli, "Scaling neural machine translation," in *Proceedings of the Third Conference on Machine Translation: Research Papers*, Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 1–9. DOI: 10.18653/v1/W18-6301. [Online]. Available: https://aclanthology.org/W18-6301.

[41] Y. You, J. Li, S. Reddi, *et al.*, *Large batch optimization for deep learning: Training bert in 76 minutes*, 2019. DOI: 10.48550/ARXIV.1904.00962. [Online]. Available: https://arxiv.org/abs/1904.00962.

[42] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 1715–1725. DOI: 10.18653/v1/P16-1162. [Online]. Available: https://aclanthology.org/P16-1162.

[43] J. Achiam, Accessed: 05-01-2023, 2018. [Online]. Available: https://spinningup.openai.com/en/latest/spinningup/rlintro2.html.

[44] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12, MIT Press, 1999. [Online]. Available: https://proceedings.neurips.cc/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[45] V. Mnih, A. P. Badia, M. Mirza, *et al.*, *Asynchronous methods for deep reinforcement learning*, 2016. DOI: 10.48550/ARXIV.1602.01783. [Online]. Available: https://arxiv.org/abs/1602.01783.

[46] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, *Trust region policy optimization*, 2015. DOI: 10.48550/ARXIV.1502.05477. [Online]. Available: https://arxiv.org/abs/1502.05477.

[47] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. DOI: 10.48550/ARXIV.1707.06347. [Online]. Available: https://arxiv.org/abs/1707.06347.

[48] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, *Continuous control with deep reinforcement learning*, 2015. DOI: 10.48550/ARXIV.1509.02971. [Online]. Available: https://arxiv.org/abs/1509.02971.

[49] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, 2018. DOI: 10.48550/ARXIV.1802.09477. [Online]. Available: https://arxiv.org/abs/1802.09477.

[50] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, 2018. DOI: 10.48550/ARXIV.1801.01290. [Online]. Available: https://arxiv.org/abs/1801.01290.

[51] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, *Playing atari with deep reinforcement learning*, 2013. DOI: 10.48550/ARXIV.1312.5602. [Online]. Available: https://arxiv.org/abs/1312.5602.

[52] M. G. Bellemare, W. Dabney, and R. Munos, *A distributional perspective on reinforcement learning*, 2017. DOI: 10.48550/ARXIV.1707.06887. [Online]. Available: https://arxiv.org/abs/1707.06887.

[53] W. Dabney, M. Rowland, M. G. Bellemare, and R. Munos, *Distributional reinforcement learning with quantile regression*, 2017. DOI: 10.48550/ARXIV.1710.10044. [Online]. Available: https://arxiv.org/abs/1710.10044.

[54] M. Andrychowicz, F. Wolski, A. Ray, *et al.*, *Hindsight experience replay*, 2017. DOI: 10.48550/ARXIV.1707.01495. [Online]. Available: https://arxiv.org/abs/1707.01495.

[55] D. Ha and J. Schmidhuber, *World models*, 2018. DOI: 10.5281/ZENODO.1207631. [Online]. Available: https://zenodo.org/record/1207631.

[56] T. Weber, S. Racanière, D. P. Reichert, *et al.*, *Imagination-augmented agents for deep reinforcement learning*, 2017. DOI: 10.48550/ARXIV.1707.06203. [Online]. Available: https://arxiv.org/abs/1707.06203.

[57] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine, *Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning*, 2017. DOI: 10.48550/ARXIV.1708.02596. [Online]. Available: https://arxiv.org/abs/1708.02596.

[58] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine, *Model-based value estimation for efficient model-free reinforcement learning*, 2018. DOI: 10.48550/ARXIV.1803.00101. [Online]. Available: https://arxiv.org/abs/1803.00101.

[59] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, *Mastering chess and shogi by self-play with a general reinforcement learning algorithm*, 2017. DOI: 10.48550/ARXIV.1712.01815. [Online]. Available: https://arxiv.org/abs/1712.01815.

[60] F. Stahlberg and S. Kumar, "Synthetic data generation for grammatical error correction with tagged corruption models," in *Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications*, Online: Association for Computational Linguistics, Apr. 2021, pp. 37–47. [Online]. Available: https://aclanthology.org/2021.bea-1.4.

[61] R. Grundkiewicz, M. Junczys-Dowmunt, and K. Heafield, "Neural grammatical error correction systems with unsupervised pre-training on synthetic data," in *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 252–263. DOI: 10.18653/v1/W19-4427. [Online]. Available: https://aclanthology.org/W19-4427.

[62] C. Chelba, T. Mikolov, M. Schuster, *et al.*, *One billion word benchmark for measuring progress in statistical language modeling*, 2013. DOI: 10.48550/ARXIV.1312.3005. [Online]. Available: https://arxiv.org/abs/1312.3005.

[63] H. Yannakoudakis, Ø. Andersen, A. Geranpayeh, T. Briscoe, and D. Nicholls, "Developing an automated writing placement system for esl learners," *Applied Measurement in Education*, vol. 31, Apr. 2018. DOI: `10.1080/08957347.2018.1464447`.

[64] S. Granger, "The computer learner corpus: A versatile new source of data for sla research," 1998.

[65] C. Bryant, M. Felice, Ø. E. Andersen, and T. Briscoe, "The BEA-2019 shared task on grammatical error correction," in *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, Florence, Italy: Association for Computational Linguistics, Aug. 2019, pp. 52–75. DOI: `10.18653/v1/W19-4406`. [Online]. Available: `https://aclanthology.org/W19-4406`.

[66] H. T. Ng, S. M. Wu, Y. Wu, C. Hadiwinoto, and J. Tetreault, "The CoNLL-2013 shared task on grammatical error correction," in *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, Sofia, Bulgaria: Association for Computational Linguistics, Aug. 2013, pp. 1–12. [Online]. Available: `https://aclanthology.org/W13-3601`.

[67] M. Tarnavskyi, "Improving sequence tagging for grammatical error correction," *Ucu.edu.ua*, 2021. DOI: `https://er.ucu.edu.ua/handle/1/2707`. [Online]. Available: `https://er.ucu.edu.ua/handle/1/2707`.

[68] T. Mizumoto, M. Komachi, M. Nagata, and Y. Matsumoto, "Mining revision log of language learning SNS for automated Japanese error correction of second language learners," in *Proceedings of 5th International Joint Conference on Natural Language Processing*, Chiang Mai, Thailand: Asian Federation of Natural Language Processing, Nov. 2011, pp. 147–155. [Online]. Available: `https://aclanthology.org/I11-1017`.

[69] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 2004.

[70] C. Napoles, K. Sakaguchi, and J. Tetreault, "JFLEG: A fluency corpus and benchmark for grammatical error correction," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 229–234. [Online]. Available: `https://aclanthology.org/E17-2037`.

[71] M. Heilman, A. Cahill, N. Madnani, M. Lopez, M. Mulholland, and J. Tetreault, "Predicting grammaticality on an ordinal scale," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, Baltimore, Maryland: Association for Computational Linguistics, Jun. 2014, pp. 174–180. DOI: `10.3115/v1/P14-2029`. [Online]. Available: `https://aclanthology.org/P14-2029`.

[72] Z. Liu, X. Yi, M. Sun, L. Yang, and T.-S. Chua, "Neural quality estimation with multiple hypotheses for grammatical error correction," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Online: Association for Computational Linguistics, Jun. 2021, pp. 5441–5452. DOI: `10.18653/v1/2021.naacl-main.429`. [Online]. Available: `https://aclanthology.org/2021.naacl-main.429`.

[73] L. Xue, N. Constant, A. Roberts, *et al.*, *Mt5: A massively multilingual pre-trained text-to-text transformer*, 2020. DOI: 10.48550/ARXIV.2010.11934. [Online]. Available: https://arxiv.org/abs/2010.11934.

[74] C. Raffel, N. Shazeer, A. Roberts, *et al.*, *Exploring the limits of transfer learning with a unified text-to-text transformer*, 2019. DOI: 10.48550/ARXIV.1910.10683. [Online]. Available: https://arxiv.org/abs/1910.10683.

[75] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-carlo tree search: A new framework for game ai.," Jan. 2008.

[76] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2014. DOI: 10.48550/ARXIV.1412.6980. [Online]. Available: https://arxiv.org/abs/1412.6980.