German Dataset Augmentation and Performance Gains through Character-Level Modeling

Bachelor Thesis

Faculty of Science, University of Bern

submitted by

Dimoth Paththiniwasam

from Colombo (LAK), Switzerland

Supervision:

PD Dr. Kaspar Riesen

Corina Masanti

Institute of Computer Science (INF)

University of Bern, Switzerland

Abstract

Grammatical Error Correction (GEC) systems are vital for automating labor-intensive proofreading tasks, yet they often underperform in non-English language domains due to the scarcity of high-quality annotated datasets. This thesis addresses these challenges by enhancing a German-language dataset, the Rotstift dataset, through synthetic data generation. A novel two-phase approach is introduced, utilizing a fine-tuned CANINE model to predict character-level modifications and applying category-specific rules to inject errors systematically. The enhanced dataset includes four error types—Deletion, Replacement, ß-ss, and Whitespace—and is evaluated using mBERT on a binary classification task. Results indicate that synthetic augmentation improves recall across all categories, with notable gains in the underperforming ß-ss category, where accuracy surpasses 50%. However, accuracy gains remain modest due to increased false positives. This work underscores the potential of character-level modeling and synthetic augmentation for improving GEC systems in resource-scarce settings, providing a foundation for further advancements in multilingual and domain-specific applications.

Contents

Chapter 1

Introduction

This chapter introduces the challenges posed by limited annotated datasets for Grammatical Error Correction (GEC) and outlines the approaches explored in this study. It provides an overview of the Rotstift dataset, including its error categories and inherent limitations, which serve as the foundation of this thesis. Finally, the specific challenges addressed and the objectives pursued are described.

1.1 Topic

In modern proofreading agencies multiple professionals collaborate to refine and enhance client texts, ensuring that each document is accurate and polished. The proofreading process typically involves several steps, each requiring meticulous attention to detail. Proofreaders must review every sentence carefully, addressing grammar, punctuation, style and coherence errors thereby identifying even the smallest mistakes that might affect the quality or meaning of the text.

Recent advancements in Artificial Intelligence (AI) and Machine Learning (ML), particularly in Natural Language Processing (NLP), have raised expectations for automating labor-intensive tasks like grammatical error correction. Fully automated systems have the potential to significantly reduce the manual effort required in proofreading. However, as Masanti et al. [1] point out, current solutions often fail to meet the requirements of professional proofreading agencies due to the scarcity of well-annotated datasets. This limitation is especially pronounced in non-English language domains such as German, where high-quality annotated datasets for Grammar Error Correction (GEC)—a task focused on identifying and correcting grammatical errors in sentences [2, 3]—are particularly scarce [4].

While the revolutionary Transformer architecture introduced by Vaswani et al. [5]

has significantly advanced NLP capabilities, the lack of diverse and comprehensive datasets continues to hinder the development of effective GEC systems that meet professional standards.

To address these challenges, synthetic dataset generation has emerged as a viable solution. By injecting deliberate errors into correctly written sentences, it is possible to expand existing datasets. For instance rule-based methods can algorithmically inject errors such as typos, incorrect punctuation or character substitutions. However, these approaches often fail to capture the diversity and complexity of human errors, as noted by Masanti et al.[1]. This highlights the need for innovative methods that improve both dataset quality and model performance.

While advancements in AI and NLP hold great potential for automating proofreading tasks, significant obstacles remain, particularly in non-English domains. Addressing these challenges requires a concerted effort to develop enhanced datasets and robust model architectures capable of meeting professional standards. The interplay of these advancements with existing limitations underscores the necessity of ongoing research to bridge the gap between current technologies and the expectations of proofreading professionals.

1.2 Overview Dataset

The foundation for this thesis is a dataset that has been collected since 2019 by the proofreading agency Rotstift AG in Switzerland. This dataset comprises approximately 80,000 documents [6] containing text with grammatical or stylistic errors in English, German, French and Italian, alongside their manually corrected versions by the Rotstift agency. In addition, errors are already classified into different types. This study focuses exclusively on the German portion of the dataset and the error types detailed in Table 1.1. From this point forward, this dataset is referred to as the Rotstift dataset.

The Rotstift dataset presents the following challenges. Mixed content and formatting artifacts, such as line breaks (\n), special symbols (e.g., Peha®), and inconsistent annotations, complicate pre-processing and disrupt tokenization. Addressing these issues is essential to create a clean dataset for training robust models.

Category	Description	Example Incorrect → Correct	Size
Deletion	Removing a single character results in an error	Walang → Walfang	2281
Replacement	Replacing a single character causes an error	Hous → Haus	739
ß-ss	The ß is replaced with ss, which is grammatically incorrect	Strasse → Straße	95
Whitespace	Errors caused by incorrect placement of whitespace	Wal fang → Walfang	2403

Table 1.1: Error categories in the Rotstift dataset with examples and the quantity in each category. All errors are subject to a specific permutation on one character and can be found multiple times in a text

1.3 Goal

This thesis aims to address the challenges of limited annotated data in the Rotstift dataset by generating a synthetic datasets for each error category while maintaining its diversity. So therefore the performance of a BERT-based model on a binary classification task, for distinguishing correct from incorrect sentences, is enhanced.

Creating synthetic datasets, however, introduces additional complexities. A reliable collection of correctly written sentences is as critical as the foundation for injecting errors. Without such data, synthetic datasets risk inheriting inaccuracies, which can negatively affect model performance. Validating large-scale datasets manually is infeasible, while automated tools often fail to capture subtle grammatical nuances and stylistic inconsistencies. This challenge is particularly noticeable with German grammar, where grammar depends on intricate rules involving case, gender, and tense agreement [7].

By addressing these challenges, this thesis seeks to enhance the Rotstift dataset and contribute to the development of more effective GEC systems.

1.4 Outline

Chapter 2

Related Work

Data augmentation plays a critical role in GEC systems, addressing data scarcity while enhancing model robustness and performance. Recent studies have proposed innovative approaches to generate high-quality synthetic data that mirrors real-world grammatical error distributions.

Contextual Data Augmentation Wang et al. (2024) [8] introduced a contextual data augmentation technique that combines rule-based substitutions with model-generated errors to create synthetic data reflecting authentic error patterns. To ensure data quality, their approach incorporates a relabeling-based data cleaning method, which mitigates the impact of noisy labels. Experimental results on standard GEC datasets demonstrated that this technique significantly improves model performance, achieving state-of-the-art results with minimal reliance on additional synthetic data.

Dynamic Data Augmentation Ye et al. (2023) [9] proposed MixEdit, a dynamic data augmentation method designed to generate realistic grammatical errors without requiring extensive monolingual corpora. Their method evaluates augmentation strategies using two metrics: Affinity, which measures how closely the augmented data matches real-world errors, and Diversity, which assesses the variation in generated errors. The results showed that a well-balanced augmentation strategy—characterized by high Affinity and optimal Diversity—substantially enhances the performance of GEC models, while also complementing traditional augmentation techniques.

Cycle Self-Augmenting Tang et al. (2023) [2] addressed the issue of robustness in GEC models by introducing the Cycle Self-Augmenting (CSA) method. This

approach leverages self-generated data during the post-training phase and incorporates regularization data for cyclical training. The CSA method not only improves model performance on clean datasets but also enhances robustness against adversarial inputs, demonstrating its effectiveness in maintaining high accuracy across varying data conditions.

These studies collectively highlight the importance of tailored data augmentation strategies in GEC. By generating high-quality synthetic data aligned with real-world error patterns, these approaches enable GEC models to perform more effectively, even in low-resource settings. Moreover, methodologies such as MixEdit and CSA provide critical insights into balancing data quality and robustness, ensuring that augmented datasets contribute meaningfully to model training.

Chapter 3

Theoretical Foundations

All models discussed in this thesis are based on the Transformer [5]. Section 3.1 provides an overview of the Transformer and key mechanisms, such as Tokenizer, Embeddings, Positional Encoding and Attention to establish a theoretical basis for the models utilized. The Hugging Face Transformers library is introduced in section 3.2 as it was used for fine-tuning pretrained models on downstream tasks. Finally, section 3.3 focuses on the selected models, mBERT and CANINE. mBERT was chosen for its robust multilingual capabilities while CANINE was selected for its character-level processing, which aligns with the character-level errors targeted for injection in this thesis.

3.1 Transformer

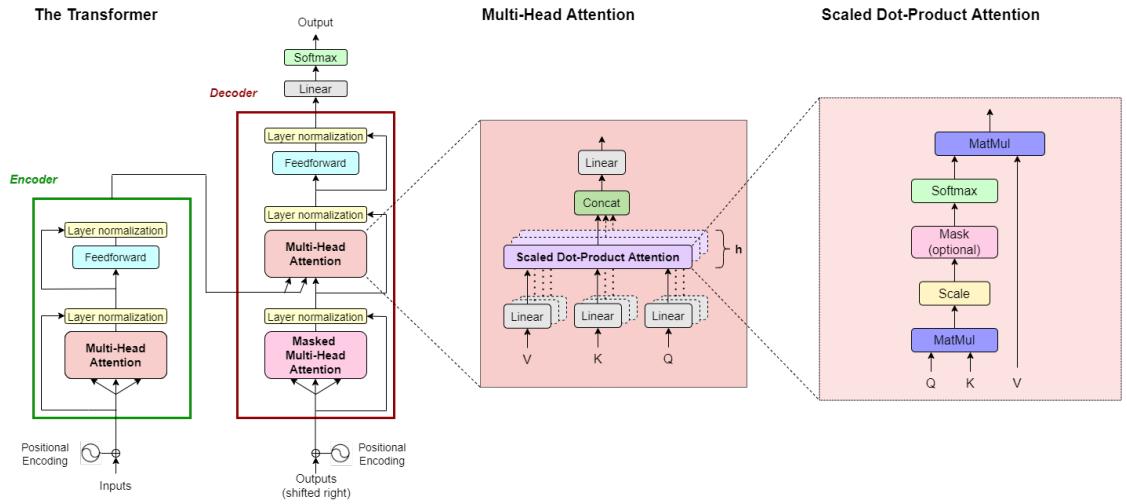


Figure 3.1: Illustration of the Transformer from [10]. Left is a Transformer with a single encoder-decoder stack depicted. In the center, the Multi-Head Attention component is shown, and on the right, the Scaled Dot-Product Attention mechanism is illustrated.

The Transformer architecture, introduced by Vaswani et al. [5] in their seminal work *Attention Is All You Need*, represents a breakthrough in processing sequential data. Unlike earlier architectures such as Recurrent Neural Networks (RNNs) or Long Short-Term Memory (LSTM)[11] networks, the Transformer processes input sequences in parallel, enabling greater computational efficiency and improved handling of long-range dependencies [5].

As shown in Figure 3.1, the Transformer consists of two main components: The encoder and the decoder. This thesis focuses solely on the encoder as the selected models for enhancing the Rotstift dataset employ an encoder-only architecture. The encoder transforms an input sentence into a continuous representation that captures contextual relationships between tokens, which serves as the basis for downstream tasks. Before being fed into the encoder, an input sentence x is transformed into an input matrix $x \in \mathbb{R}^{n \times d}$, where $n, d \in \mathbb{N}$. This transformation occurs in three distinct steps. The first two steps, tokenization and embedding, are detailed in subsection 3.1.1. The third step, discussed in subsection 3.1.2, introduces Positional Encoding to incorporate information about the order of the words in the input sentence. Once this transformation is complete, the resulting input matrix x is ready to be processed by the encoder, as described in subsection 3.1.5.

3.1.1 Tokens and Embeddings

Tokenization and embedding done by a Tokenizer and Embedding-Layer respectively form the first step of the Transformer's processing pipeline, as they transform textual data into discrete units of numerical representations.

Tokenizer

Tokenization breaks down a sentence into smaller components called tokens, which can be words, subwords or characters, depending on the strategy used. Each token is mapped to a unique identifier in a predefined vocabulary. For example, the sentence *"Cows eat plants"* might be tokenized into [Cows, eat, plants], each assigned an ID.

Embedding

Once an input sentence is tokenized into a sequence of n tokens, each token is then mapped to a numerical vector through an embedding layer [12]. An embedding is understood as the numerical representation that describes the token in a high dimensional vector space. These vectors are represented as $e \in \mathbb{R}^d$, where d is the embedding dimension. All token embeddings are stored in an Embedding Matrix of size $V \times d$, where V is the vocabulary size, and d is the dimensionality of the embeddings.

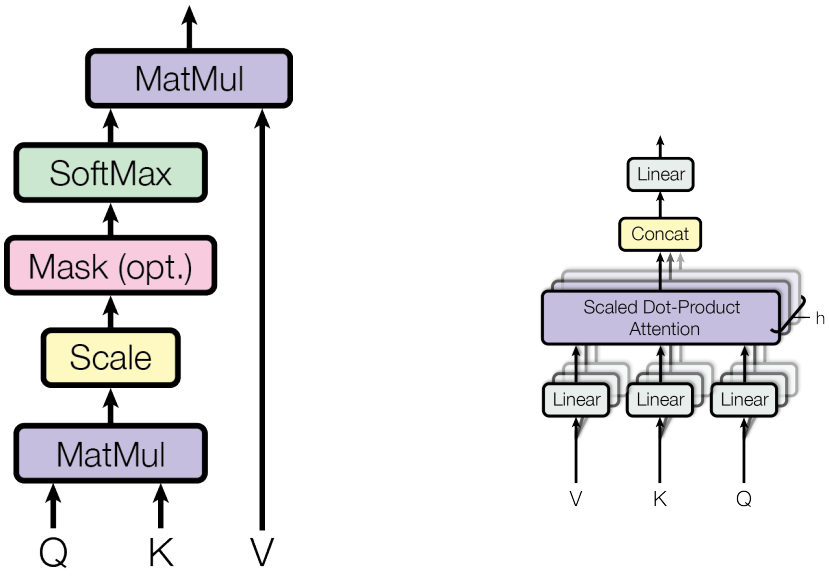
Tokens with similar meanings or contexts are positioned closer together in the vector space $\mathbb{R}^d$, facilitating the encoding of semantic relationships. For instance, in the embedding space, the vector difference between "man" and "grandpa" should be smaller than the difference between "man" and "cow". After applying an embedding to our sequence of n tokens we obtain the input embedding $x \in \mathbb{R}^{n \times d}$.

3.1.2 Positional Encoding

Transformers lack inherent mechanisms to capture the order of tokens in a sequence. To address this, Vaswani et al. [5] utilizes Positional Encoding, which adds positional information to token embeddings. The importance of positional information is evident in examples such as the sentences "Cows eat plants" and "Plants eat cows.". Although these sentences contain the same words, their meanings differ significantly due to the word order.

For each token at position $pos \in [1, \ldots, n]$ in a input sequence with n tokens, a positional encoding vector $PE \in \mathbb{R}^d$ is computed using sine and cosine functions.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d}) \qquad PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d}),$$



(a) Scaled Dot-Product Attention

(b) Multi-Head Attention

Figure 3.2: Illustrations from *Attention is all you need* by Vaswani et al. [5], showing the pipeline of the Scaled Dot-Product Attention and Multi-Head Attention.

where i is the embedding dimension. The sum of PE and the embedding vector at *pos* ensures the order is encapsulated in the embedding vector. This procedure is done before they are fed into the encoder as shown in the Figure 3.1.

The choice of sine and cosine functions allows the model to extrapolate to sequences longer than those seen during training [5]. Additionally, their periodicity enables the Transformer to capture patterns in the input sequence that depend on both absolute and relative positions of tokens. [5].

3.1.3 Attention

As previously discussed, the Transformer architecture is built around the core concept of Self-Attention, also referred to as intra-Attention. Self-Attention builds up on the concept of Attention, first introduced by Bahdanau et al. [13].

Vaswani et al. [5] describes Attention as a function that maps a Query q and Key-Value pairs k, v to an output, where q, k, v and the resulting outputs are vectors. The output is computed as a weighted sum of the Value v, where the weights are determined by a compatibility function that measures the similarity between the Query q and its corresponding Key k. In the Transformer, Self-Attention is implemented using the Scaled Dot-Product Attention, the compatibility function, illustrated in Figure 3.1.3 (a).

This mechanism calculates the weights, which quantify the influence of a specific token on every token in the sequence including itself.

For instance, depending on the context the word *Apple* can be a fruit or a tech company. In the sentence *I want to eat an apple* the word eat defines *Apple* as a fruit. The Self-Attention would therefore place a higher weight on eat for *Apple* to shift the attention there and generating a so-called contextual embedding for each word in the sentence.

Building on the concept of Scaled Dot-Product Attention, the Transformer extends its capability through the Multi-Head Attention, which allows the model to retrieve the context from multiple angles simultaneously.

Scaled Dot-Product

Before Self-Attention can be applied, the input matrix $x \in \mathbb{R}^{n \times d}$ into three distinct matrices Queries (Q), Keys (K), and Values (V) as shown in 3.1.3 (b).

$$Q = x \times W_Q, \qquad K = x \times W_K, \qquad V = x \times W_V,$$

where $W_Q, W_K \in \mathbb{R}^{d \times d_k}$ and $W_V \in \mathbb{R}^{d \times d_v}$.

These transformations are achieved using three separate linear layers as depicted in Figure 3.1. Each layer with its own learnable weight matrix without adding the biases.

To obtain the attention weights first the dot product is calculated for each query $q_i \in Q$ from the Query $Q \in \mathbb{R}^{n \times d_k}$ and all the keys $k_i \in K^T$ from the Key matrix $K \in \mathbb{R}^{n \times d_k}$, where $i \in [1, \ldots, n]$. The result is then scaled by $\frac{1}{\sqrt{d_k}}$ and passed through a softmax function to generate the weights for the corresponding value $v_i \in V$ from the Value matrix $V \in \mathbb{R}^{n \times d_v}$ to be multiplied.

$$\text{Attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

In practice, this process is performed using matrix multiplication to yield a contextualized embedding called *head* of the input matrix x, with each row representing a token from the input sequence.

Multi-Head Attention

Multi-Head Attention is a extension of Self-Attention that allows the model to contextualize each token from various angles simultaneously. The Multi-Head At-

tention is formally defined as follows:

$$\text{MultiHead} = \text{Concat}(head_1, \ldots, head_h)W^O,$$
$$\text{where head}_\text{i} = \text{Attention}(XW_i^Q, XW_i^V, XW_i^V) \; and \; i \in [1, \ldots, h]$$

Capturing the relationships between the tokens from different angles is achieved by employing multiple *heads* simultaneously. These heads are then concatenated and multiplied by $W^O \in \mathbb{R}^{hd_v \times d}$ to form the final output, where h is the number of *heads*. This final projection ensures the output has dimensions $n \times d$ aligning it with the original input dimensionality, where d represents the embedding dimension and n the number of tokens in the input sequence.

3.1.4 Position-wise Feed-Forward Networ

The Position-wise Feed-Forward Network (FFN) is a component of the Transformer architecture, introduced by Vaswani et al. [5]. It is applied independently to each token's embedding in a sequence. Unlike the attention mechanism, which models relationships between tokens in a sequence, the FFN focuses on enhancing individual token representations, capturing feature-level dependencies without considering context from neighboring tokens.

3.1.5 Encoder

The encoder consists of six identical layers, each processing the entire input sentence x simultaneously. Each contains two main components as shown in Figure 3.1:

- Multi-Head Attention (h=8)
- Position-wise Feed-Forward Network

As shown in the figure 3.1 a residual connections[14] is wrapped around each component followed by a normalization layer [15] to stabilize training and improve computational efficiency.

3.2 Hugging Face Transformer library

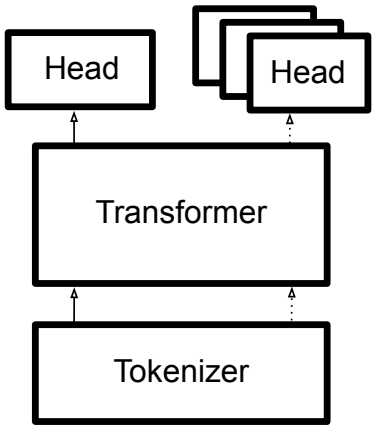


Figure 3.3: Illustration from [16] showing the components implemented by Hugging Face's Transformers library. Each Transformer-based model can be augmented with prebuilt heads, crafted to produce outputs for a tasks. Furthermore, each model is employed in conjunction with a compatible tokenizer.

The Transformers library, developed by Hugging Face [16], provides a powerful interface for working with pretrained models such as mBERT and CANINE. It enables seamless integration of pretrained language models into diverse NLP tasks, including fine-tuning for classification, token tagging or text generation.

3.2.1 Tokenizer

Each Hugging Face model includes a tokenizer aligned with its pretrained vocabulary and structure. This ensures consistency between training and inference, facilitating accurate downstream task processing..

3.2.2 Heads

Hugging Face supports multiple task-specific heads, which are common in NLP such as text classification and token tagging. These heads extend the base model by adding output layers tailored to the specific tasks as shown in 3.3.

Binary Text Classification is task that involves predicting one of two possible classes for a given input text. For instance, determining whether a review is "positive" or "negative" is a common application.

Token Classification involves assigning labels to individual tokens in an input sequence, making it suitable for tasks such as named entity recognition (NER) or part-of-speech (POS) tagging. The model predicts a label for each token, enabling it to identify entities like names, locations, or other categories [16].

3.3 Models

This section outlines the Transformer-based models utilized in this thesis: mBERT and CANINE. Both models are supported by the Hugging Face Transformers library and rely exclusively on the encoder stack of the Transformer architecture [5, 17, 18].

3.3.1 mBERT

The Bidirectional Encoder Representations from Transformers (BERT) model introduced by Devlin et al. [17] marked a turning point in Natural Language Processing (NLP) by enabling bidirectional context-aware representation learning. Building on the Transformer encoder architecture, BERT is designed to capture relationships between tokens in a sequence by attending to both preceding and succeeding tokens.

Model Architecture

mBERT's architecture mirrors the original BERT model, consisting of multiple Transformer encoder layers. Since BERT is focused on understanding and encoding text, it only utilizes the encoder stack of the Transformer. Each layer incorporates self-attention mechanisms and feedforward networks to capture dependencies between tokens, regardless of their position in the input sequence [5]. This architecture allows mBERT to process multilingual input text effectively, creating context-aware embeddings that generalize across languages.

Bidirectionaly

BERT and mBERT use a bidirectional attention mechanism, processing the entire input sequence simultaneously to capture dependencies in both directions. This is achieved through Masked Language Modeling (MLM), where a portion of the tokens in the input sequence is masked, and the model is trained to predict them based on surrounding context. This training objective enables the model to learn richer representations.

For example, in a hypothetical sentence such as "The cat sat on the [MASK]," the model can use both the left and right context to infer that the missing token is likely

"mat." This demonstrates how BERT's bidirectional attention mechanism enables it to consider the full context of a sentence during prediction.

3.3.2 CANINE

The *Character-Aware Neural Information Encoder* (CANINE) introduced by Clark et al. [18] is a Transformer-based encoder-only model that processes text directly at the character level, eliminating the need for explicit tokenization or fixed vocabularies.

Model Architecture

CANINE employs a unique architecture that integrates three Transformer encoders [18], each serving distinct purposes within the processing pipeline:

Initial Shallow Encoder consisting of a single layer, is applied to the character embeddings. This encoder uses local attention to contextualize the embeddings within a limited window, capturing fine-grained local dependencies.

Deep Encoder down-samples the input step, reducing the sequence length for efficiency. A deep Transformer encoder, similar in structure to a traditional BERT encoder, processes this reduced sequence. The deep encoder captures long-range dependencies and provides global contextualization.

Final Shallow Encoder is the next step after the deep encoder upsampled the sequence back to its original length. A second shallow encoder, again consisting of a single layer, is then applied to generate the final contextualized character embeddings.

Tokenization-Free Processing

A significant innovation of CANINE is its ability to process text without explicit tokenization [18]. Traditional language models often rely on subword tokenization methods, which can be suboptimal for certain languages and may struggle with out-of-vocabulary words [18]. CANINE addresses this challenge by operating directly on character sequences, thereby eliminating the need for predefined vocabularies or tokenization steps [18]. During pretraining, CANINE employs a Masked Language Modeling (MLM).

3.4 Definitions

This section introduces linguistic definitions, evaluation metrics that are utilized in this thesis and establishes what Supervised Learning (SL) is.

3.4.1 Linguistic Definitions

Hyphenation divides words into their syllables, ensuring each part contains at least one vowel (*ge-hen, Mit-tag*). Single consonants between vowels move to the next line (*Ka-me-ra*), while double consonants are split (*Mit-tag*).

Vowels are in German *a, e, i, o, u* including *ä, ö, ü*. Vowels are essential for forming syllables and can be short or long in duration.

Consonants are all the other letters excluding vowels. They often occur at the beginning or end of syllables.

Diphthongs are combinations of two vowels within the same syllable, where the sound glides from one vowel to another. In German, the common diphthongs are *au, ei,* and *eu* (or *äu*), as in the words *Haus*, *mein*, and *Freund*.

Compound words are words formed by combining two or more words to create a new word. In German, compound words are common and often consist of nouns, verbs, or adjectives joined together without spaces, such as *Haustür* (house door) or *Arbeitsplatz* (workplace).

ß-Rule in German, ß is used after a long vowel or diphthong within the same syllable, as in *Maß* or *Fleiß*. After a short vowel, *ss* is used instead, as in *Mas-se*.

3.4.2 Evaluation Metrics

The evaluation metrics that are used to quantify the performance of an model is presented in this section. Beforehand a clarification what *True Positive* (TP), *False Positive* (FP), *True Negative* (TN) and *False Positive* (FP) is given. To do this a scenario is needed where a model predicts whether a given sentence is written correctly or not. In this case, Positive refers to predictions assigned to the class correct and Negative refers to predictions assigned to the class incorrect. If the prediction aligns with the label assigned to the input then it is a True Positive or True Negative. On the other hand, a False Positive occurs when the model predicts a sentence as correct but it is actually incorrect and False Negative occurs when a sentence is predicted as incorrect but is actually correct.

Accuracy measures the proportion of correctly classified instances (both correct and incorrect) out of the total number of instances. It is calculated as:

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

Precision Precision is metric that quantifies the frequency with which a machine learning model accurately predicts the positive class.

$$\text{Precision} = \frac{TP}{TP + TN}$$

Recall represents a measure of the frequency with which a model accurately detects positive cases among all genuine positive examples in the dataset.

$$\text{Recall} = \frac{TP}{TP + FN}$$

3.4.3 Supervised Learning

In this thesis the downstream tasks that are used to fine-tune the pretrained models are supervised. This means the dataset the model is trained on model on labeled. Meaning that data provides an input and the desired output (labels). The model can adjust its prediction by comparing its prediction with the label. This comparison is done with a loss function.

Chapter 4

Preprocessing

As outlined in section 1.2, the challenges inherent in this thesis require a meticulously executed preprocessing procedure. In Section 4.1, the method employed to prepare the data for fine-tuning and for generating synthetic datasets is described in detail. In section 4.2, the selected external source for injecting the error patterns identified in the Rotstift dataset is presented, along with the rationale behind its selection.

4.1 Cleaning

The Rotstift dataset contains incorrect sentences and their corresponding corrected versions. To ensure the data quality required for fine-tuning the dataset underwent preprocessing steps to address inconsistencies and noise.

The correct textual data and their corresponding incorrect counterparts were divided into sentences using the Python library *sentence-splitter*[1]. This process generated one list of sentences for each input source. The resulting lists were then compared pairwise. Sentences at the same position in the two in the lists that did not match character-by-character are retained for further processing. Those are the sentences that are written incorrectly or correctly.

To resolve encoding inconsistencies, the textual data was converted to ASCII format and non-printable characters were removed. The complete set of printable characters used in this step is detailed in appendix A.1. After cleaning, the dataset was ready for the fine-tuning phase.

[1] `https://github.com/mediacloud/sentence-splitter`

4.2 Synthetic Data Source

To augment the Rotstift dataset with synthetically generated errors, an external source of grammatically correct sentences was required. The selection criteria for this source were as follows:

Correctness The dataset must consist of grammatically correct sentences to ensure the reliability of the injected errors. Manual verification was impractical due to the dataset's required size, so the source must inherently meet high grammatical standards.

Open Source To avoid legal issues, the source must be in the public domain or open source, allowing unrestricted use for research purposes.

The selected dataset was derived from the publicly available decisions [2] of the Federal Supreme Court of Switzerland (FSCS), known as *Bundesgerichtsentscheide (BGE)*. The FSCS employs a rigorous multi-layered review process involving judges and clerks, ensuring high grammatical precision and consistency in its published decisions[3]. Only decisions written in German were used in this study. The processed FSCS dataset served as the foundation for injecting errors systematically.

[2]Webscraped from: `https://bger.li/`

[3]`https://www.ch-info.swiss/en/edition-2021/die-gerichte-des-bundes/fakten`

Chapter 5

Methodology

This chapter begins by outlining the two-phase strategy used to inject errors into the FSCS dataset: A prediction phase and a subsequent modification phase. In section 5.1, the overall solution is introduced, explaining how a fine-tuned CANINE model [18] is used to detect potential error positions for each category. This section also includes the rationale for selecting CANINE, emphasizing its character-level processing capacity to handle typographical issues in German text.

Section 5.1 covers the token classification process, where CANINE is adapted to label individual characters as either "retain" or "modify." The labeling procedure, discussed in subsection 5.1.3, constructs training labels by comparing pairs of incorrect and correct sentences from the Rotstift dataset. These labels guide the model in learning, where errors are most likely to occur, and Generating Labels for Error Injection describes how the trained model's outputs are then applied to the FSCS dataset to predict positions for artificial errors.

Building on these predictions, section 5.2 details how the algorithm modifies tagged positions to inject the identified errors. First, Deletion removes the target character. Next, Replacement swaps the character with a plausible alternative derived from a curated dictionary. The algorithm for Whitespace errors is illustrated in figure 5.1, demonstrating how spacing around words, commas and hyphens is manipulated to produce irregularities. Finally, the rules for handling ß and ss are explained, highlighting the validation step that ensures ß is only inserted where linguistically appropriate.

5.1 Solutions

The solution proposed uses a fine-tuned CANINE [18] on each error category except ß-ss to predict potential error positions in text. During the prediction phase, the model was trained to classify whether each character in a sentence should be retained or modified. The modification phase applies predefined rules to alter characters at the predicted positions for each error category.

5.1.1 Reasoning for model selection

The CANINE model was chosen for its ability to process text at the character level without requiring tokenization. Errors such as typos, substitutions, and deletions occur at the character level, and tokenization could split characters crucial for error detection and injection. CANINE's architecture avoids this issue, making it well-suited for languages with complex orthographies like German. Furthermore, its demonstrated effectiveness in handling diverse text structures aligns closely with the task requirements [18].

5.1.2 Character Classification

The CANINE model was fine-tuned for a token classification task at character level. This approach directly aligns with the character-level errors targeted in this thesis. Fine-tuning involved the following steps:

Input Data The correct sentences from the Rotstift dataset are used.

Labels For each sentence, a label vector was created with the same length as the input sentence (in characters, including whitespaces). Each character was assigned one of two labels called **retain** and **modify**.

Loss Function The Cross-Entropy Loss is used and is defined as

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\log(\hat{y}_{i,c}),$$

where N is the number of samples, C is the number of classes, $y_{i,c}$ is the true label, and $\hat{y}_{i,c}$ is the predicted probability for class c. was used to optimize the model during fine-tuning. In this case $C = 2$ and the resulting formula is calculates as followed:

$$(y\log(p) + (1-y)\log(1-p))$$

5.1.3 Labeling

The label vectors were generated by comparing incorrect sentences from the Rotstift dataset with their corresponding corrected versions. Each character in the correct sentence received a label. To generate labels for training, each incorrect sentence was compared character by character with its corresponding correct sentence. Differences between the two were identified, and any character in the incorrect sentence that did not match the correct version was labeled as **modify**. Characters that matched were labeled as **retain**.

5.1.4 Character Classification for error injection

After fine-tuning, the model was applied to the FSCS dataset of grammatically correct sentences. For each sentence, the model produced a label vector indicating potential positions for error injection. These predictions provided the foundation for generating realistic synthetic datasets with injected errors.

5.2 Modifications

Once a position is identified as *modify*, predefined rules are applied to introduce errors specific to each category. The following subsections outline the rules implemented for different error types.

Deletion

The modification process for deletions is straightforward and intuitive: The tagged character is directly removed from the text.

Replacement

When the model processes a text, it identifies locations requiring modifications but does not suggest specific replacements. Randomly replacing the tagged characters is ineffective. It risks disrupting the patterns inherent to specific error categories and impeding generalization. To address this issue, a dictionary is created for each character set, consisting of three elements: The incorrect character at the center, along with its preceding and succeeding characters. This dictionary is derived from mismatches between the correct and incorrect texts in the Rotstift dataset. For each character set a list of replacement candidates is generated. The corresponding list is created by the mismatch on character level between the center character in the correct written sentences and the character in the incorrect written sentences.

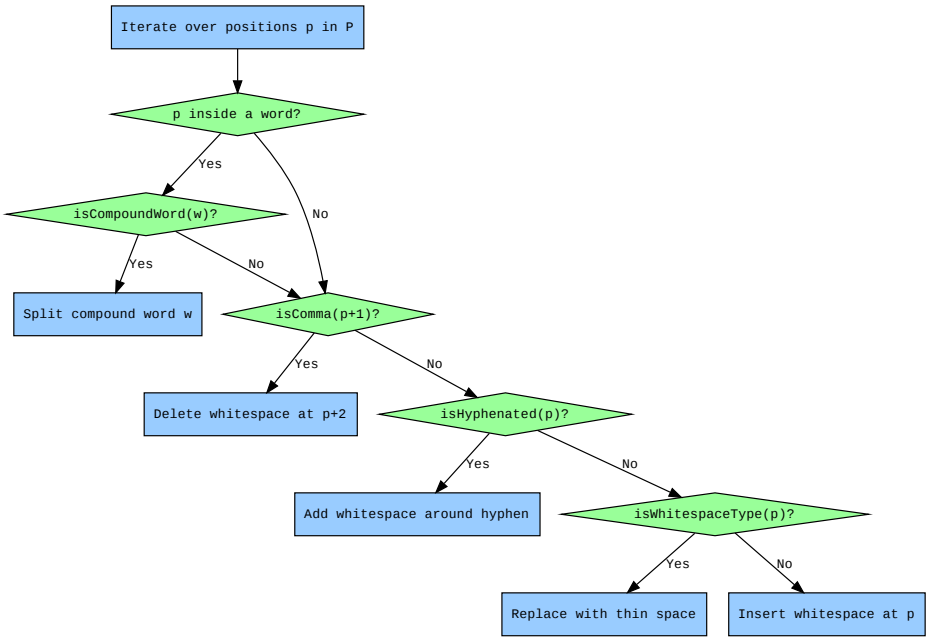


Figure 5.1: Illustrates the flow of algorithm to inject whitespace errors

An injection is done when a character set from the dictionary matches a tagged character along with its preceding and succeeding character. The central character in the set is replaced with a randomly chosen character from the corresponding list of candidates. This dictionary contains 1,569 unique character sets and their respective replacement options. As an example, table A in the appendix illustrates the first 50 entries.

Whitespace

This subsection presents an algorithm for injecting whitespace-related errors into a sentence. Given a sentence S and a set of tagged positions P, the algorithm iterates over each position $p \in P$ and applies one of several whitespace transformations. By selectively inserting, deleting, or altering whitespace in different contexts, it can simulate various typographical errors or stylistic inconsistencies. The overall flow of this process is illustrated in figure 5.1.

Word Boundaries If p in S lies inside a word w, the algorithm checks whether w is a compound word . If so, it splits w at each boundary with whitespace. The words then are cased accordingly to German grammar.

Comma Whitespace If the position before p in S is a comma, any whitespace immediately following that comma is removed, thereby introducing spacing

errors.

Hyphen Spacing If p contains a hyphen, the algorithm adds extra whitespace around that hyphen, creating abrupt or undesirable spacing breaks.

Special Whitespace Types If p points to whitespace in S, the algorithm deliberately replaces it with either no space or an incorrect type of space.

Default Insertion In cases not covered above, the algorithm inserts an extra whitespace at Position p in the Sentence s

5.2.1 ß and ss

For this error category, identifying patterns in the positioning of error injections is unnecessary. As a result, a fully algorithmic approach was adopted.

In Switzerland, the *ß*-rule is not applied universally, necessitating an additional validation step before injecting these errors. To address this, the following algorithm was developed to determine whether *ß* can validly replace *ss*:

Algorithm 1 Validation of ß-rule

Input: Word w containing *ss*
Output: Boolean indicating whether *ss* can be replaced by *ß*
hyphenate$(w) = \{s_1, \ldots, s_n\}$
for all $s_i \in$ hyphenate(w) **do**
  if *ss* $\in s_i$ **then**
    if The preceding character to *ss* is a long vowel or diphthong **then**
      return True
    end if
  end if
end for
return False

Chapter 6

Experimental Evaluation

This chapter presents the conducted experiments and the corresponding results. In section 6.1 the general experiment setup is presented. Then, section 6.2 focuses on establishing a baseline performance using the only the cleaned data stemming from the Rotstift dataset. The performance is measured on a binary text classification task using the mBERT and quantified by the evaluation metrics Accuracy, Recall and Precision.

6.1 Experiment Setup

In 6.1.1 the general experiment setup is described that is applied to all experiments. Then the technologies that are utilized are presented in Subsection 6.1.2

6.1.1 General Approach

A given dataset consist of sentences and the corresponding label. The label is correct or incorrect. Further the dataset is balanced meaning one half is labeled correct and the other half incorrect. Then the dataset is divided into a test set and training set and both are roughly balanced too. The test set consist of 40% of the datasets size and 10% of the training set is used for validation during fin-tuning. With that said the model is then fine-tuned on a binary classification task to identify if a given sentence is grammatically correct or incorrect. The performance is then evaluated on the test set.

Category	Test Set	Training Set	Validation Set
Deletion	915	1229	137
Replacement	366	327	38
ß-ss	38	51	6
Whitespace	962	1297	144

Table 6.1: Showing the size of the test, train and validation set in each error category

6.1.2 Technological Stack

The experiments are implemented in Python using the Transformers- [16], Dataset- and Evaluation library by Hugging Face. The training is executed on Google Colab with a NVIDIA A100 GPU.

6.2 Baseline

This section presents how the baseline is determined to compare and quantify the influence of the enhancement. This is done by fine-tuning a mBERT model on datasets for each error category. The performance on the test set for each error category is quantified by Accuracy, Precision and Recall. The size of the dataset in each error category is listed in the Table 6.1. To be clear for each dataset a newly initialized mBERT is used.

6.2.1 Special Token

For the mBERT that is fine-tuned on the Whitespace dataset there was an additionally step. As mentioned in 3.2 the corresponding Tokenizer to the pretrained model is provided. The Tokenizer that is used for mBERT is a subword Tokenizer [18]. The information about whitespace is lost during tokenizing, which is actually crucial in this setting. Therefore a special Token [SPACE] was added and every whitespace was replaced with this token before tokenizing. Therefore the model needs to create a embedding for this token.

6.2.2 Hyperparameters

The hyperparameters that are used for fine-tuning $\text{mBERT}^{\text{Base}}$ for each error category on the cleaned Rotstift dataset are shown in the Table 6.2.

Category	**Learning Rate**	**Batch Size**	**Epochs**	**Weight Decay**
Deletion	10^{-6}	32	20	0.01
Replacement	10^{-6}	16	10	0.01
ß-ss	$5 * 10^{-5}$	16	8	0
Whitespace	10^{-6}	32	20	0.01

Table 6.2: Hyperparameter used for each error category to train mBERT

Category	Accuracy	Recall	Precision
Deletion	50.71%	38.52%	53.50%
Replacement	41.58%	41.31%	43.95%
ß-ss	50.00%	47.61%	55.55%
Whitespace	50.10%	54.64%	47.04%

Table 6.3: Results from mBERT^{Base}

6.2.3 Results

The Table 6.3 shows, that Deletion and Whitespace achieve the highest Accuracy, likely because they have the most training data, thus generalizing better. By contrast, Replacement does not even reach 50% Accuracy, which is worse the randomly guessing given the balanced dataset. In ß-ss the model is exactly 50% which indicates that it does not generalize. As illustrated in the confusion matrices in the appendix B.2 Deletion and Whitespace maintain a high number of correct classifications while ß-ss suffers from more misclassifications.

6.3 Experiment Enhanced Dataset

In this section, the experimental evaluation is replicated under the same conditions and hyperparameter settings outlined in Table 6.2. The only modification lies in the size of the training set, which is doubled by incorporating the synthetically generated dataset for each category. The new training set maintain its balanced property. Importantly, the test set remains unchanged thereby facilitating a direct comparison between the original and modified experiments. This approach allows

Category	Accuracy	Recall	Precision
Deletion	54.09%	46.84%	48.98%
Replacement	47.54%	61.02%	50.63%
ß-ss	53.00%	55.93%	51.03%
Whitespace	48.65%	52.21%	43.32%

Table 6.4: Results from mBERT^{Enhanced}

for an assessment of the impact of the enhanced dataset on the overall performance of the model.

6.4 Enhanced Dataset Experiment

The results in Table 6.4 reveal a nuanced impact of dataset enhancement. Accuracy shows a slight improvement for Deletion, Replacement, and ß-ss but stagnates or slightly declines for Whitespace. However, recall consistently improves across all categories, suggesting the enhanced dataset increased the model's sensitivity to identifying incorrect sentences.

Notably, the category ß-ss now achieves an accuracy of 53.00%, crossing the 50% threshold, and a recall of 55.93%, which indicates better generalization to this error type. This improvement underscores the potential of the enhanced dataset in addressing weaker categories, although further efforts are required to balance recall and precision.

The Whitespace category exhibits the lowest precision at 43.32%, highlighting a trade-off between increased recall and a rise in false positives. These results suggest that while the enhanced dataset improves error detection, it does not consistently translate into higher accuracy due to the accompanying increase in false alarms.

6.4.1 Original Dataset (Table 6.3)

The results from the baseline dataset show that Deletion and Whitespace achieve the highest accuracy, likely benefiting from more abundant training examples and less intrinsic complexity. By contrast, Replacement and ß-ss fail to exceed 50% accuracy, performing worse than random guessing on a balanced dataset. As illustrated in the confusion matrices (Appendix B.2), Deletion and Whitespace exhibit higher correct

classification rates, while ß-ss suffers from a greater proportion of misclassifications.

6.4.2 Enhanced Dataset (Table 6.4)

Doubling the dataset size with synthetic examples introduces noticeable changes. Accuracy improves modestly for Deletion and ß-ss, with ß-ss surpassing the 50% mark, indicating progress toward generalization. Replacement sees significant improvement in recall, rising to 61.02%, while Whitespace exhibits minor changes. These findings suggest that while the enhanced dataset increases recall for all categories, accuracy gains remain limited. The synthetic data appears to amplify error detection but also contributes to more false positives, preventing accuracy from scaling proportionally.

6.4.3 Comparison

Comparing the original and enhanced datasets highlights key trends:

Improved Recall The enhanced dataset consistently boosts recall across all categories, indicating that the model identifies a greater number of incorrect sentences.

Minimal Accuracy Gains Accuracy remains largely stagnant or sees only minor improvements, as increased recall is offset by a rise in false positives.

Category-Specific Effects ß-ss benefits most from the enhanced dataset, showing notable improvements in accuracy and recall, suggesting the added synthetic data helped the model generalize better for this category.

Trade-Offs Deletion and Whitespace continue to dominate in accuracy, but their performance gains remain limited, suggesting that the augmented dataset alone cannot fully address existing bottlenecks.

Chapter 7

Future Work

Overall, the experiments demonstrate that enhancing datasets with synthetic examples can improve recall, particularly for under performing categories. However, the modest accuracy gains indicate that future work should explore more diverse and sophisticated augmentation techniques to reduce false positives and translate recall improvements into broader performance gains. Future work can explore integrating seq2edit models, which are designed to directly predict edit operations rather than full-text rewrites, offering a more efficient approach to error correction tasks. Leveraging such models may reduce computational overhead while maintaining high accuracy, particularly for localized error types like Deletion and Replacement. Additionally, incorporating generative approaches, such as the one demonstrated by Malmi et al.[19] in their work on LASERTagger could further enhance the diversity and quality of synthetic datasets by generating context-aware edits. These methods can complement rule-based techniques by producing realistic training data that better aligns with the natural variability in text errors. Combining these strategies with larger, multilingual transformer models could also enable cross-lingual generalization and improved performance in diverse application settings. Lastly, employing active learning mechanisms to iteratively improve the model by incorporating user corrections during deployment could provide a robust pathway for refining both detection and correction capabilities in dynamic environments.

Appendix A

Tables

Table A.1: Printable ASCII Characters (Character Codes 36–127)

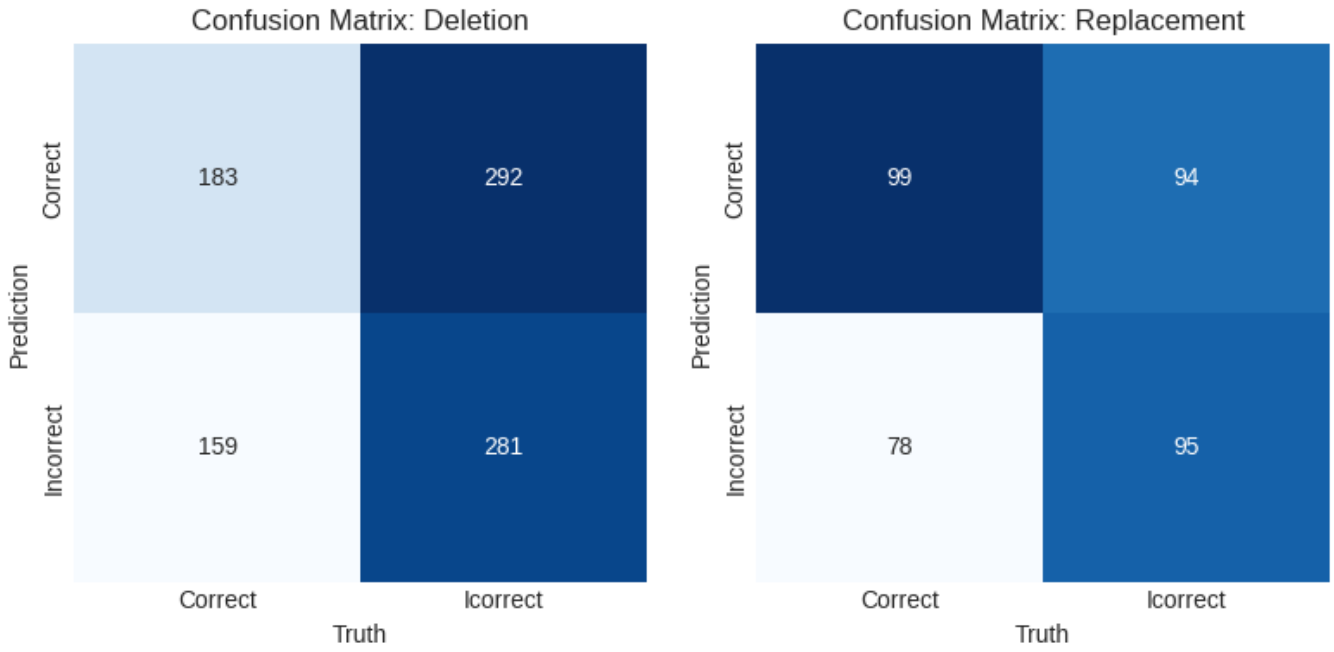
Code	**Character**
36	$
37	%
38	&
39	’
40	(
41	)
42	*
43	+
44	,
45	-
46	.
47	/
48–57	0–9
58	:
59	;
60	¡
61	=
62	¿
63	?
64	@
65–90	A–Z
91	[
92	\
93	]
94	ˆ
95	_
96	`
97–122	a–z
123	{
124	—
125	}
126	˜

Character Set	Top Replacements
Red	[é, R, d]
itä	[i, d]
nzi	[t, i, n, e]
tzu	[t, u]
zut	[z]
ute	[u, e, G]
tei	[t, i, w]
eil	[e, l, t]
ile	[i]
len	[l, n, u]
das	[d, s, e, m]
ass	[a]
Sie	[e, S]
ab	[a]
dem	[d, m]
Dez	[D]
eze	[e, c]
zem	[z]
emb	[e]
mbe	[m]
ber	[b, r, s, ü]
die	[d, e, t]
Gel	[G]
ele	[e]
leg	[l, n]
ege	[e]
gen	[g, n, o, S]
enh	[i, e, h]
nhe	[e, n]
hei	[i, h]
eit	[t, e, r, s]
hab	[h, b]
abe	[e, a, h]
ben	[n, b, a, w]
gab	[b, g, e]
end	[d, e, h, l]
rcg	[g]
ode	[e, o]
der	[r, d, n, o, i, u, s, l]
per	[p, r]

Table A.2: Character sets and their replacements (first 40)

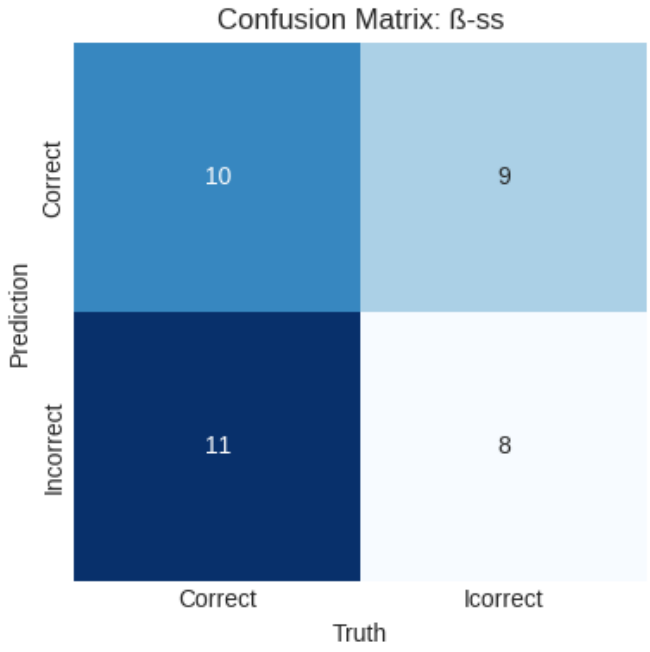
Appendix B

Results

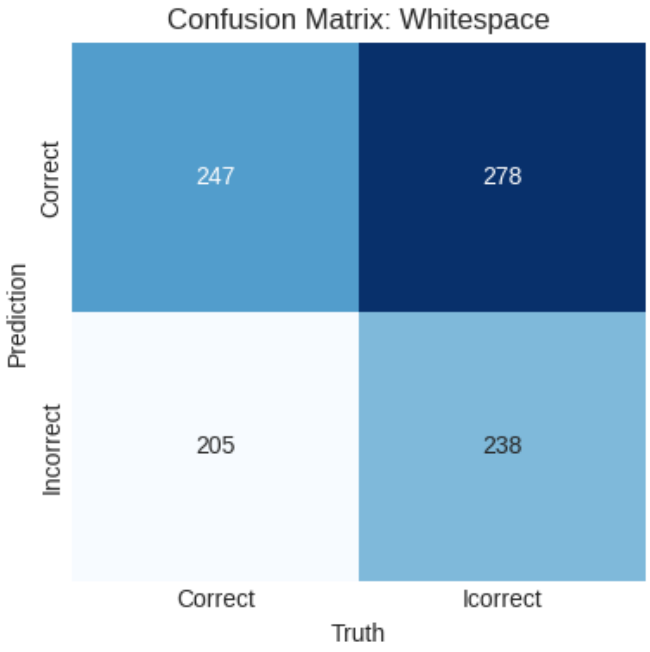


(a) Deletion

(b) Replacement

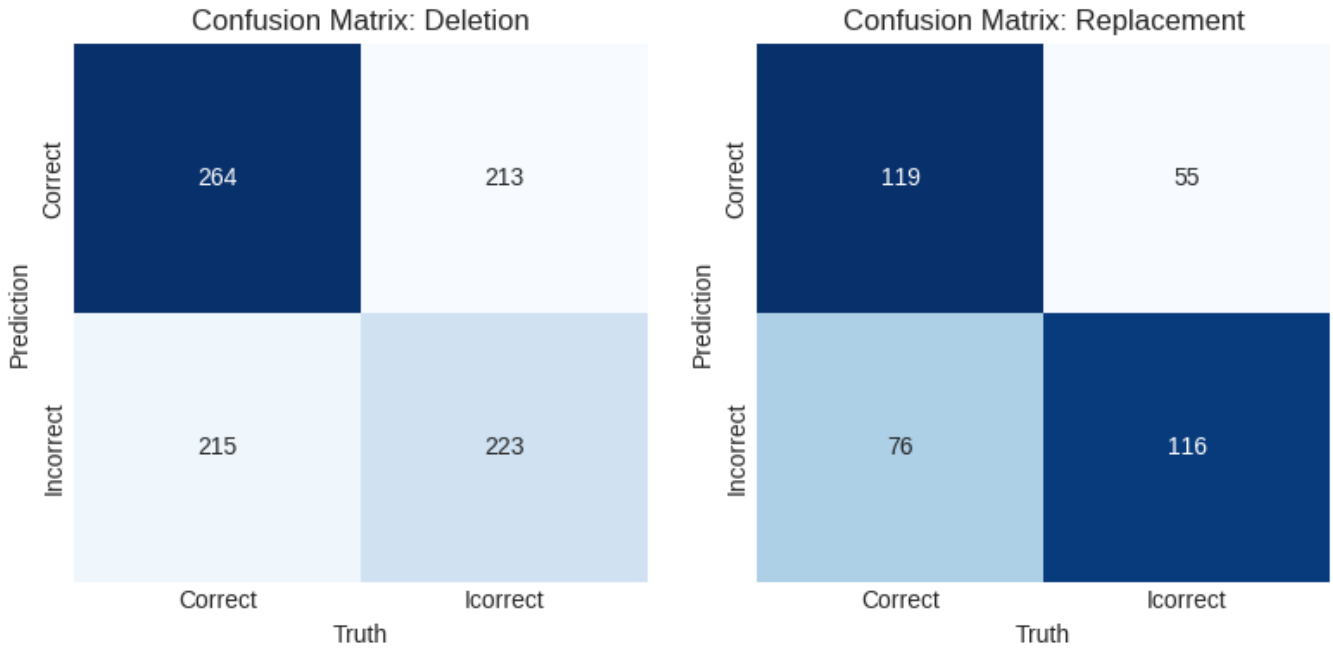


(c) ß-ss



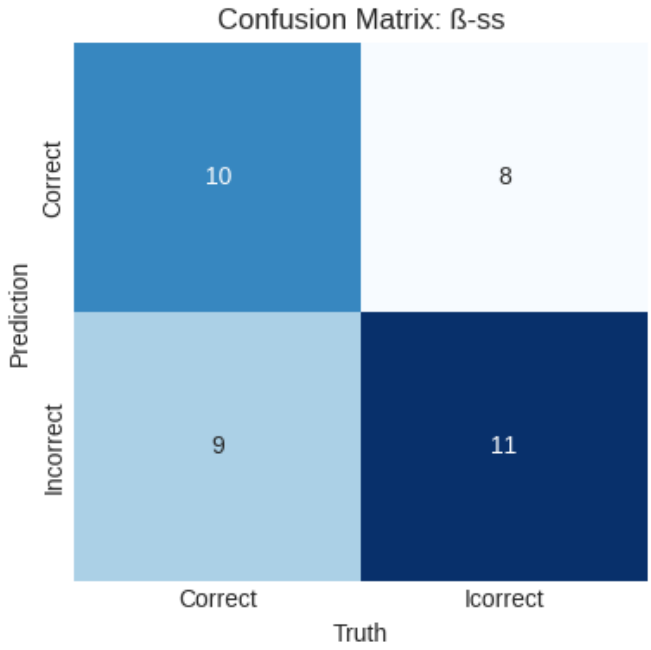
(d) Whitespace

Figure B.1: Confusion Matrices for mBERT$^{\text{Base}}$

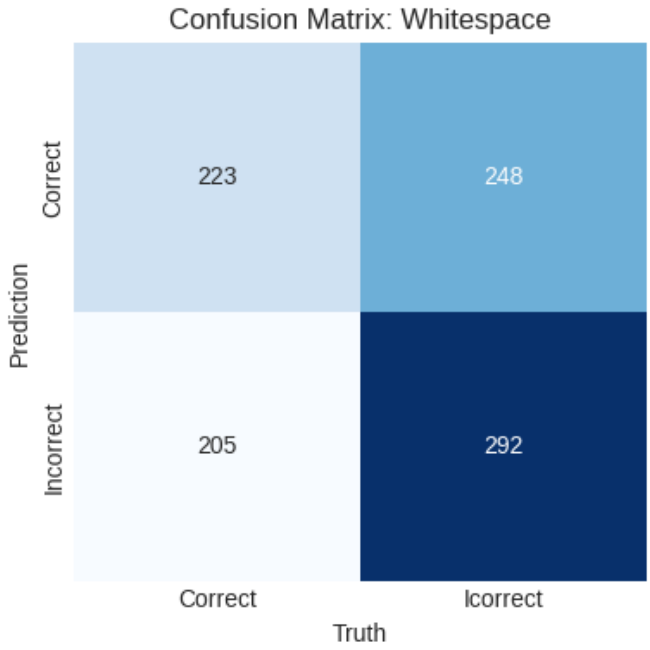


(a) Deletion

(b) Replacement



(c) ß-ss



(d) Whitespace

Figure B.2: Confusion Matrices for mBERT$^{\text{Enhanced}}$

Bibliography

[1] Corina Masanti, Hans-Friedrich Witschel, and Kaspar Riesen. Novel benchmark data set for automatic error detection and correction. In Elisabeth Métais, Farid Meziane, Vijayan Sugumaran, Warren Manning, and Stephan Reiff-Marganiec, editors, *Natural Language Processing and Information Systems*, pages 511–521, Cham, 2023. Springer Nature Switzerland.

[2] Zecheng Tang, Kaifeng Qi, Juntao Li, and Min Zhang. Beyond hard samples: Robust and effective grammatical error correction with cycle self-augmenting, 2023.

[3] Wei Zhao, Liang Wang, Kewei Shen, Ruoyu Jia, and Jingming Liu. Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 156–165. Association for Computational Linguistics, 2019.

[4] Jakub Náplava, Milan Straka, Jana Straková, and Alexandr Rosen. Czech grammar error correction with a large and diverse corpus. *Transactions of the Association for Computational Linguistics*, 10:452–467, 2022.

[5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[6] `https://prg.inf.unibe.ch/index.php/research/a-prose---advanced-proofreading-services/` Accessed on 12-15-2024.

[7] Sebastian Kürschner and Damaris Nübling. The interaction of gender and declension in germanic languages. *Folia linguistica*, 45(2):355–388, 2011. Peer-Review vorhanden.

[8] Yixuan Wang, Baoxin Wang, Yijun Liu, Qingfu Zhu, Dayong Wu, and Wanxiang Che. Improving grammatical error correction via contextual data augmentation, 2024.

[9] Jingheng Ye, Yinghui Li, Yangning Li, and Hai-Tao Zheng. Mixedit: Revisiting data augmentation and beyond for grammatical error correction, 2023.

[10] Anne-Sofie Maerten and Derya Soydaner. From paintbrush to pixel: A review of deep neural networks in ai-generated art, 2024.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 11 1997.

[12] Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, and Ivan V. Oseledets. Tensorized embedding layers for efficient model compression. *CoRR*, abs/1901.10787, 2019.

[13] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[15] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

[16] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[18] Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. CANINE: pre-training an efficient tokenization-free encoder for language representation. *CoRR*, abs/2103.06874, 2021.

[19] Eric Malmi, Sebastian Krause, Sascha Rothe, Daniil Mirylenka, and Aliaksei Severyn. Encode, tag, realize: High-precision text editing. *CoRR*, abs/1909.01187, 2019.