RedPenNet for Grammatical Error Correction: Outputs to Tokens, Attentions to Spans

Bohdan Didenko
WebSpellChecker LLC / Ukraine
bogdan@webspellchecker.net

Andrii Sameliuk
WebSpellChecker LLC / Ukraine
andrii.sameliuk@webspellchecker.net

Abstract

The text editing tasks, including sentence fusion, sentence splitting and rephrasing, text simplification, and Grammatical Error Correction (GEC), share a common trait of dealing with highly similar input and output sequences. This area of research lies at the intersection of two well-established fields: (i) fully autoregressive sequence-to-sequence approaches commonly used in tasks like Neural Machine Translation (NMT) and (ii) sequence tagging techniques commonly used to address tasks such as Part-of-speech tagging, Named-entity recognition (NER), and similar. In the pursuit of a balanced architecture, researchers have come up with numerous imaginative and unconventional solutions, which we're discussing in the Related Works 4 section. Our approach to addressing text editing tasks is called RedPenNet and is aimed at reducing architectural and parametric redundancies presented in specific Sequence-To-Edits models, preserving their semi-autoregressive advantages. Our models achieve $F_{0.5}$ scores of 77.60 on the BEA-2019 (test), which can be considered as state-of-the-art the only exception for system combination (Qorib et al., 2022) and 67.71 on the UAGEC+Fluency (test) benchmarks.

This research is being conducted in the context of the UNLP 2023 workshop, where it will be presented as a paper for the Shared Task in Grammatical Error Correction (GEC) for Ukrainian. This study aims to apply the RedPenNet approach to address the GEC problem in the Ukrainian language. Public data related to this article may appear over time in this GitHub repository [1].

1 Introduction

The GEC challenge has been tackled with various techniques, including the traditional Autoregressive (AR) Neural Machine Translation (NMT) using the transformer architecture (Vaswani et al., 2017), as well as additional methods that we refer to collectively as Inference Optimized (IO). The existing IO methods for GEC can be broadly categorized into two groups, as described further.

The first group is non-autoregressive Feed Forward (FF) approaches which involve a single forward pass — through the model and provides token-level edit operations, such as the approach proposed in (Awasthi et al., 2019), (Omelianchuk et al., 2020). The advantage of FF approaches is their fast inference speed. However, their limitations lie in how they maintain consistency between interrelated edits, which leads to the need for iterative sentence correction approaches. The iterative sentence correction process solves some issues with interrelated corrections. However, it introduces new challenges. The absence of information about the initial input state could potentially lead to substantial modifications of the text meaning and structure, including rewording, word rearrangement, and the addition or removal of sentence components.

The second category consists of Inference Optimized Autoregressive (IOAR) models, which can be further separated into two subcategories: (i) sequence-to-edits (SeqToEdits). This category encompasses works such as (Malmi et al., 2019), (Chen et al., 2020), (Stahlberg and Kumar, 2020), and the RedPenNet model examined in this paper; (ii) the recently proposed Input-guided Aggressive Decoding (IGAD) approach (Ge et al., 2022), which has been proven effective for GEC tasks, as demonstrated in the study (Sun et al., 2021). More information about these model categories can be found in the Related Work 4 section.

In our study, we propose RedPenNet, which is an IOAR model of the SeqToEdits subtype. RedPenNet utilizes a single shallow decoder (Kasai et al., 2020) for generating both replacement tokens and spans. During the generation of edit tokens, the encoder-decoder attention weights are used to determine the edit spans. For these attentions, pre-

[1] https://github.com/WebSpellChecker/unlp-2023-shared-task

Proceedings of the Second Ukrainian Natural Language Processing Workshop (UNLP), pages 121–131

softmax logits are fed as inputs to a linear transformation which predicts the position of the edit in the source sentence. This approach is similar to the method described in Pointer Networks (Vinyals et al., 2015). Additionally, we train compact task-specific decoder BPE vocabularies to reduce the cost of the pre-softmax dot operation, making it more efficient for predicting replacement tokens. The RedPenNet model is also capable of tackling the challenge of Multilingual GEC (Rothe et al., 2021). To achieve this, specialized shallow decoders need to be trained for different languages. This gives the ability to use a single model with a multilingual pre-trained encoder and language-specific decoders.

Our proposed solution has a design that enables converting the input sequence into any output sequence, achieving competitive results in solving the GEC task.

2 RedPenNet

2.1 General

Instead of predicting the target sequence directly, the RedPenNet model generates a sequence of N 2-tuples $(t_n, s_n) \in V \times \mathbb{N}_0$ where t_n is a BPE token obtained from the pre-computed decoder vocabulary 3 and s_n denotes the span positions. In the RedPenNet approach, we define each of the J edit operations $\mathbf{e}$ as a sequence of C 2-tuples (t_c, s_c), where $2 \leq C \leq N$. The first token for each edit e_j is represented as $t_{c=0} = \texttt{SEP}$. As previously mentioned, in our approach, a single edit can consist of multiple tokens. However, to determine the span of a single edit, only two positions are required: s_{start} and s_{end}. To accomplish this, we impose the following constraints for each e_j: $s\_start = s_{c_0}$ and $s\_end = s_{c_1}$. The remaining s_c values for $c \in 2, ..., C$ are not considered. In RedPenNet, if a correction requires inserting text at a position n in the source sequence, it is expressed as $s_{start} = s_{end}$. To handle the deletion operation, a special token DEL $\in V$ is used, which is equivalent to replacing the span with an empty string. If the input text is error-free, RedPenNet generates an EOS token in the first AR step, thereby avoiding unnecessary calculations.

The iterative process of applying edits to the source sequence is illustrated in Algorithm 1. Also, the process of generating GEC edits using the RedPenNet architecture can be visualized with the help of the following illustration 1.

Algorithm 1 editsToCorrect()

$s\_start \leftarrow 0$
$s\_end \leftarrow 0$ { Initialize spans}
$\mathbf{y} \leftarrow \mathbf{x}$ {Initialize $\mathbf{y}$ as tokenized input}
$\mathbf{z} \leftarrow \epsilon$ {Initialize $\mathbf{z}$ edit seq with the empty string.}
for $n \leftarrow 1$ **to** N **do**
  if $t_n = \texttt{EOS}$ **then**
    return $\mathbf{y}$
  else if $t_n = \texttt{SEP}$ **then**
    $\mathbf{y}_{s\_start}^{s\_end} \leftarrow \mathbf{z}$
    $\mathbf{z} \leftarrow \epsilon$
    $s\_start \leftarrow s_n$
  else
    if $t_n \neq \texttt{DEL}$ **then**
      $\mathbf{z} \leftarrow \texttt{concat}(\mathbf{z}, t_n)$
    end if
    if $t_{n-1} = \texttt{SEP}$ **then**
      $s\_end \leftarrow s_n$
    end if
  end if
end for

In the case of RedPenNet, similar to the Seq2Edits approach (Stahlberg and Kumar, 2020), it is important to maintain a monotonic, left-to-right order of spans and ensure that SEP tokens are never adjacent to each other and the final edit token is always EOS. None of our models generated invalid sequences during inference without any constraints, as it is also the case with Seq2Edits.

2.2 Encoder

The utilization of pre-trained language models has been consistently shown to improve performance on a range of NLP downstream tasks, including GEC, as observed in numerous studies. To train RedPenNet, we deployed pre-trained models from the HuggingFace transformers library[2]. We have observed that models trained on Masked Language Modeling (MLM) tasks perform the best as encoders for the RedPenNet architecture. Therefore, in this work, we focus solely on this family of models. The availability of a range of models within the HuggingFace library offers the flexibility to choose a pre-trained model based on the required size, language, or a multilingual group. This opens up the potential for RedPenNet to (i) create multilingual GEC solutions using language-specific

[2] https://huggingface.co/models

Positions	1	2	3	4	5	5	6	7	8	9	10	11	12	13	14	15	16	17
Input Sentence	~~In~~	~~a~~	other	hand	∅	many	~~of~~	stars	sold	their	privacy	to	earn	more	and	more	money	.

Table 1: Visual representation of the tokenized encoder input sequence that includes visual markings, intended to improve the clarity of the editing process.

Autoregressive Steps	1	2	3	4	5	6	7	8
Input Tokens	SOS	SEP	On	the	SEP	,	SEP	DEL
Input Spans	0	1	3	∅	5	5	6	7
Output Tokens	SEP	On	the	SEP	,	SEP	DEL	EOS
Output Spans	1	3	∅	5	5	6	7	∅

Table 2: This example depicts a step-by-step demonstration of a RedPenNet autoregressive inference that encompasses multi-token edits, insertions and deletions.

decoders with a single multilingual encoder, and (ii) construct RedPenNet model ensembles based on different pre-trained models with comparatively less efforts.

2.3 Decoder

A transformer decoder stack was used as for the decoder in the RedPenNet model. During the training phase, the model learned an autoregressive scoring function $P(\mathbf{t}, \mathbf{s}|\mathbf{x}; \mathbf{\Phi})$, which is implemented as follows:

$$\mathbf{\Phi}* = \arg\max_{\mathbf{\Phi}*} \log P(\mathbf{t}, \mathbf{s}|\mathbf{x}; \mathbf{\Phi})$$
$$= \arg\max_{\mathbf{\Phi}*} \sum_{n=1}^{N} \log P(t_n, s_n|t_1^{n-1}, s_1^{n-1}, \mathbf{x}; \mathbf{\Phi})$$

where $\mathbf{t} = (t_1, ..., t_n)$ represents the sequence of ground-truth edit tokens with SEP tokens that are used to mark the start of each edit. Additionally, $\mathbf{s} = (s_1, ..., s_n)$ indicates the sequence of ground-truth span positions, which denote specific ranges in the input sequence $\mathbf{x}$.

In line with the standard Transformer architecture, the previous time step predictions are fed back into the Transformer decoder. At each step n, the feedback loop consists of the BPE token embedding of t_{n-1}, which is combined with a decoder-specific trainable positional encoding embedding p_{n-1}. The resulting sum is then concatenated with the span embedding of s_{n-1}.

The span embedding on step n can be defined as:

$$\mathbf{s\_emb}_n = s\_mask_n \cdot \mathbf{x\_emb}_{s_{n-1}}$$

$$s\_mask_n = \begin{cases} 0, & \text{if } n = 0 \\ 1, & \text{if } 0 < n \leq 2 \\ 0^{|tid_{n-1} - b|} & \\ +0^{|tid_{n-2} - b|}, & \text{otherwise} \end{cases}$$

where tid_n is an index of token t_n in decoder vocabulary V and is denoted as $\mathbf{tid} = \text{index}(\mathbf{t}, V)$, b is an index of SEP token in V and $\mathbf{x\_emb}_{s_{n-1}}$ is a vector embedding corresponding to $x_{s_{n-1}}$ token. In other words, there are two cases in how span embedding takes values depending on the preceding token sequence: (i) by using the embedding of the token $x_{s_{n-1}}$ from the encoder input sequence when $t_{n-2} = SEP \vee t_{n-1} = SEP$ or (ii) by using a zero-filled embedding ϵ with the same dimensions D as x_n. During training, similar to the MLM task, a binary spans target mask for spans sequences are used to regulate the given logic.

As a result, the inputs of the decoder at step $n-1$ can be expressed as follows:

$$\text{Concat}(\mathbf{t\_emb}_{n-1} + \mathbf{p}_{n-1}, \mathbf{s\_emb}_{n-1})$$
$$= [t\_emb_{n-1,1} + p_{n-1,1}, ..., t\_emb_{n-1,D} + p_{n-1,D},$$
$$s\_emb_{n-1,1}, ..., s\_emb_{n-1,D}] \in \mathbb{R}^{2D}$$

The technique of utilizing the pre-softmax attention weights from an encoder-decoder attention layer to represent the probabilities of positions in the input sequence was introduced in Pointer Networks (Vinyals et al., 2015) and later applied to the GEC task in the Seq2Edits approach (Stahlberg and Kumar, 2020). Additionally, to increase the number of trainable parameters at this stage, a dense layer has been added to the bottom of the spans output linear transform.

3 Training Decoder BPE Vocabularies

In the traditional implementation of the Transformer model, a shared source-target vocabulary is utilized for both the decoder and the encoder, as described in (Vaswani et al., 2017). It is evident that the pre-softmax linear transformation required to transform the decoder output into predicted next-token probabilities is computationally expensive.

Its computational complexity can be expressed as $O(d \cdot v)$, where d is the output dimension of the model and v is the decoder vocabulary size.

If the GEC task is approached by generating correction strings for the edits and using autoregressive decoding for this purpose, we tend to think that the information entropy of the generated sequences will be significantly lower compared to that of the input sequences. Our belief is based on the following two assumptions:

1. People tend to make mistakes in similar phrases and words.

2. Corrected versions of spelling words are statistically more frequent and can be represented by fewer BPE tokens.

Therefore, a smaller BPE vocabulary will be sufficient to create efficient representations of sequences of corrections. In section 5.1.2, we test this hypothesis on one of the languages, as the example shows.

4 Related Work

In the context of the GEC task, the closest family of approaches to RedPenNet is the Autoregressive approaches, specifically the SeqToEdits subtype. They include models such as Lasertagger, Erroneous Span Detection and Correction (ESD&ESC), and Seq2Edits comparison with which is important for understanding the impact of our work. These models share the advantage that the number of autoregressive steps are based on the number of necessary edits to the original text, rather than the length of the input text. We will evaluate each of these approaches to the GEC problem in this section. In this section, we will also discuss the Aggressive Decoding approach, which has evolved from the traditional sequence-to-sequence approach.

The Seq2Edits (Stahlberg and Kumar, 2020) approach predicts a sequence of N edit operations autoregressively from left to the right. Each edit operation is represented as a 3-tuple (tag, span, token) that specifies the action of replacing. The approach allows constructing an edit sequence for any pair (x, y). Tag prediction also improves explainability in the GEC task. For 3-tuple generation, a divided transformer decoder is used, and the tag and span predictions are located between its parts. Seq2Edits approach is similar to RedPenNet in the following (i) generation of spans and replacement tokens within the same autoregressive step, (ii) using Pointer Networks to predict spans. The difference between compared approaches is: 1. RedPenNet uses a single decoder stack to generate tokens and spans. 2. The Seq2Edits approach is different in terms of generating multi-token edits. According to (Bryant et al., 2017), an edit that has at least two tokens (multi-token edit) represents 10% of all edits from the CoNLL-2014 test set. As mentioned before, in the Seq2Edits approach, each step of the autoregressive process predicts an edit which consists of a 3-tuple (tag, end span, replacement token), where the replacement token is a sub-word. According to the cited articles (Stahlberg and Kumar, 2020), the Seq2Edits approach has the capability of representing multi-token edits as a list of single-token edits, where the tag and span remain unchanged, with only the replacement token being changed. In terms of the RedPenNet approach, a single edit can be represented by multiple autoregressive steps, allowing a more natural generation of multi-token edits. 3. In the RedPenNet approach, decoder-specific positional encodings are added to the decoder inputs at the bottom of the decoder stack. This allows the model to effectively utilize the order of multi-token edits. The approach presented in Seq2Edits does not clearly state the location in the divided Transformer decoder, where positional encodings can be utilized. The absence of such encodings can result in difficulty for the model in comprehending the order of the replacement tokens being inserted within the same span positions. 4. RedPenNet uses a pre-calculated, task-specific version of the BPE decoder vocabulary to generate edit tokens, thus reducing the cost of the pre-softmax linear transformation.

The Lasertagger (Malmi et al., 2019) approach deploys an autoregressive Transformer decoder to annotate the input sequence with tags from pre-calculated output vocabulary. With the limited size of the tags, vocabulary minimizes the cost of the pre-softmax linear transformation, making Lasertagger the fastest approach among the IOAR SeqToEdits architectures. However, the RedPenNet model presents several key differences: (i) it uses a BPE vocabulary instead of a tag vocabulary, (ii) it generates edits rather than tagging the input sequence, and (iii) it can produce a sequence of tokens for each edit.

In the ESD&ESC (Chen et al., 2020) approach, the task of solving the GEC editing problem is

divided into two subtasks: Erroneous Span Detection (ESD), where incorrect spans are identified through binary sequence tagging, and Erroneous Span Correction (ESC), where the correction of these spans is performed using a classic autoregressive approach that implies generation of edits for tokens surrounded by annotated span tokens. RedPenNet shares some similarities with this architecture, as it also utilizes autoregressive generation of a sequence of edit tokens, separated by control tokens that are part of the decoder vocabulary. The ESD&ESC approach differs from RedPenNet in several key aspects. 1. Firstly, RedPenNet predicts the span positions in a one-by-one manner at the decoder level, while the ESD&ESC approach uses a separate encoder to generate spans. However, the ESD approach has the same limitations as the FF family, since the ESD tags may not always be consistent, leading to difficulties in maintaining consistency between interrelated edits. The ESC decoder during generation will not have the capability to fully rectify the situation, as it will be confined to the range of the annotated span tokens. 2. RedPenNet approach is capable of decomposing neighboring errors in the input text into multiple edit operations, if necessary. Conversely, the ESD approach merges nearby errors in a single span.

The Aggressive Decoding (Sun et al., 2021) method accelerates the AR calculations for the task by utilizing the input tokens as drafted decoded tokens and autoregressively predicting only those portions that do not match. This leads to a significant improvement in inference speed. The disadvantage of the IGAD approach is that it requires the use of a shared vocabulary with the encoder during decoding. Therefore, even when the input and output sequences are the same, IGAD requires a significant number of floating point operations for the pre-softmax linear transformation in the decoder which is calculated using the formula: $O(v \cdot d \cdot l)$, where v is the vocabulary size, d is the model depth, and l is the input length. This problem becomes more obvious in the case of using pre-trained multilanguage models, which traditionally have larger encoder vocabularies and corresponding matrix embeddings. The impact of decoder vocabulary is analyzed in section 3. Additionally, since the length of the output sequence in IGAD is directly tied to the length of the input sequence, the issue of quadratic complexity in attention mechanisms remains in the decoder. This can be a challenge when dealing with long sequences and requires the use of specialized transformer architectures in the decoder.

It is worth mentioning, that the Highlight and Decode Technique described in our previous study (Didenko and Shaptala, 2019). Similar to the Erroneous Span Detection (ESD) component in the ESD&ESC approach, a binary sequence tagging model was used to identify incorrect spans. Subsequently, a broadcast binary sequence mask was element-wise multiplied to a special "highlight" embedding. The result of this operation was added to the encoder output at the bottom of the decoder stack. This allowed the decoder to predict the replacement tokens only for the "highlighted" spans. However, as outlined in the mentioned article, this approach had a list of limitations.

5 Experiments

5.1 UNLP 2023 Shared Task

The UNLP-2023 conference hosted the first Shared Task (Syvokon and Romanyshyn, 2023) in GEC for Ukrainian. One of the primary difficulties in addressing the GEC problem for the Ukrainian language lies in the scarcity of high-quality annotated training examples — a common issue for Non-English GEC. The Ukrainian language also poses an additional challenge due to its rich morphological structure and fusional nature. (Syvokon and Nahorna, 2021). The foundation of this Shared Task was established by Grammarly's efforts to develop a corpus that has been professionally annotated for GEC and fluency edits in the Ukrainian language, referred to as the UA-GEC corpus. The Shared Task consists of two tracks: (i) GEC-only, which focuses on automatically identifying and correcting grammatical errors in written text, and (ii) GEC+Fluency, which encompasses corrections for grammar, spelling, punctuation, and fluency. Given that the RedPenNet architecture is capable of handling any type of editing, including rephrasing, reordering words, and sentence splitting, we decided to participate in the GEC+Fluency track.

GEC+Fluency Baseline: Furthermore, the organizers offered a baseline model [3] based on facebook/mbart-large-50. This model was trained for a NMT task with the objective of autoregressively generating correct text from erroneous input. The score of baseline can be found in table 3.

[3] https://huggingface.co/osyvokon/mbart50-large-ua-gec-baseline

5.1.1 Data

In the case of GEC tasks, data is typically stored in the m2 format (Dahlmeier and Ng, 2012), where each instance consists of a source text and a list of edits required to transform it into the target text. To adapt the m2 training examples for the RedPenNet architecture, we (i) deployed the pre-trained encoder tokenizer to tokenize the erroneous input text, (ii) used the decoder tokenizer 3 to tokenize the edits correction strings, and (iii) converted the span offsets from the word count separated by spaces to the corresponding BPE tokens (sub-words) offsets.

UA-GEC: In the GEC+Fluency track of the Shared Task, the participants were given access to the gec-fluency public dataset [4]. The training data comprises 32,734 examples, where 15,161 contain at least one annotated error edit, while 17,573 are error-free. The evaluation dataset consists of 1,506 dev set and 1,350 test set instances. In this Shared Task, two annotators annotated all examples from the development set of the dataset and some examples from the training set. They also annotated all examples from the evolution dev sets, as well as some examples from the training data. For the Shared Task, all the data was tokenized using the stanza library [5]. To categorize the dataset edits by error types, we utilized a set of 20 tags. They included 14 grammar types and 6 fluency types.

Synthetic Data: Much of the research on the GEC problem shows that the use of pre-generated synthetic data reduces model training time and also improves overall quality. For the UNLP 2023 Shared Task, we generated over $160K$ Ukrainian erroneous data sentences based on error-free texts taken from data corpora presented on lang.org.ua [6] website. For our error generation approach, we utilized mbart-large-50 as a pre-train model, which we trained using the back translation method (Xie et al., 2018) on the training data from the UA-GEC dataset. Our task was to transduce the error-free input text sequence into the erroneous one. The synthetic data generation model was trained on a Google Colab Premium GPU instance for 8 epochs with a batch size of 4, a learning rate of $1e{-}5$, and a maximum input and output length of 128 tokens each. The performance of the RedPenNet architecture trained on this pre-training data is presented in Table 4.

[4] https://github.com/asivokon/unlp-2023-shared-task

[5] https://stanfordnlp.github.io/stanza/

[6] https://lang.org.ua/uk/corpora/

5.1.2 Decoder vocabulary for Ukrainian GEC

Multiple BPE decoder vocabularies were trained with varying sizes and evaluated based on the resulting output token count (refer to Figure 1). A training text file was created specifically for this purpose, consisting of correction strings extracted from the m2 edits. The (gec-fluency/train.m2) file from the UNLP-2023 Shared Task was used as the source. Also, we added additional 50,000 of the most frequent words from the Ukrainian Frequency dictionary of lexemes of artistic prose.[7] to the vocabularies training text file extracted from (gec-fluency/train.m2). The (gec-fluency/valid.m2) was utilized to evaluate and compare the different sizes of the decoder vocabularies.

The evaluation was performed by extracting and concatenating the correction strings from all edits for each annotated m2 sentence into a space-separated sequence. This sequence was then tokenized using different decoder vocabularies. Example: annotated m2 sentence:

> S Нечіткі бенефітс співпраці , натомість вихначені зобовязання
> A 5 6|||Spelling|||визначені|||...|||0
> A 6 7|||Spelling|||зобов'язання|||...|||0
> A 1 2|||Spelling|||бенефіціари|||...|||1
> A 5 6|||Spelling|||визначені|||...|||1
> A 6 7|||Spelling|||зобов'язання|||...|||1
> A 7 7|||Punctuation|||.|||...|||1

concatenated edits corrections:

> визначені зобов'язання бенефіціари визначені зобов'язання .

To compare the advantages of using a shorter task-specific decoder vocabulary for the Ukrainian GEC task, we will use the mbart-large-50 baseline model 5.1 as a reference. For this model, the number of operations required for the pre-softmax linear transformation is $(1024 \cdot 250,054)$ = 256,055,296 floating-point operations. In contrast, our trained vocabulary with a size of 16,384 performs the same task using only $(1024 \cdot 16,384)$ = 16,777,216 operations while maintaining a smaller encoding length than the baseline.

5.1.3 Model Configuration

Encoders: For the encoder part of RedPenNet, we chose pre-trained models from those available

[7] http://ukrkniga.org.ua/ukr_rate/hproz_92k_lex_dict_orig.csv

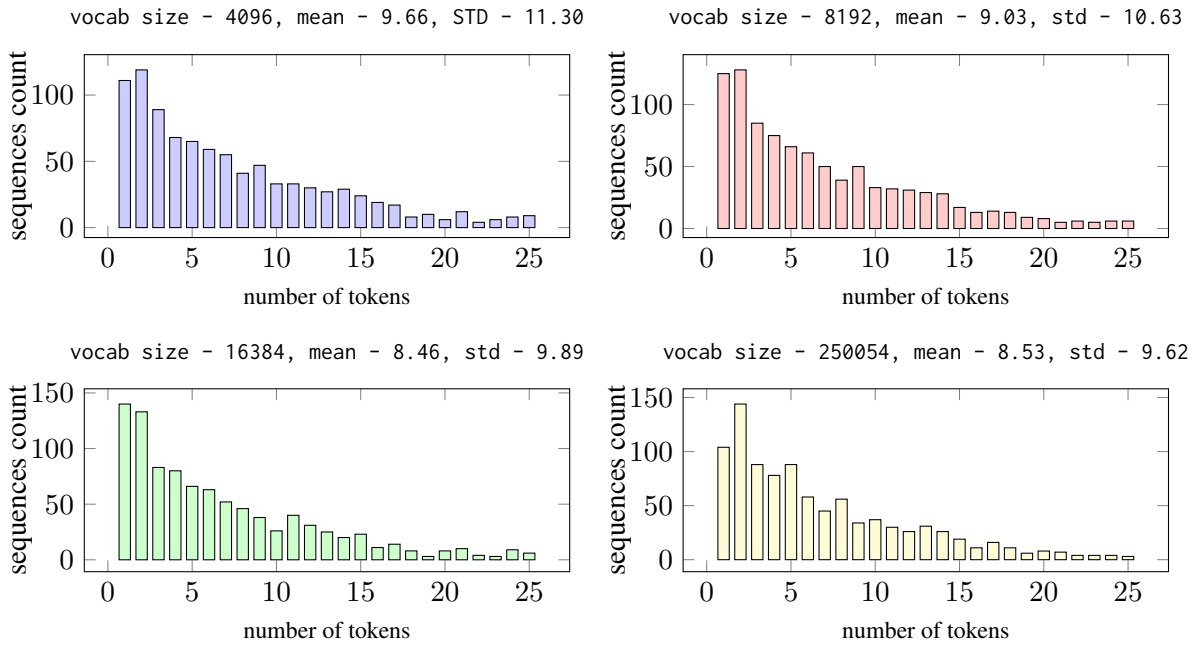

Figure 1: The x-axis depicts the number of tokens required to represent the concatenated correction of sequences for each m2 instance. The y-axis represents the total count of the concatenated corrections extracted from the (gec-fluency/valid.m2) that meet a specific number of tokens. The results show that as the vocabulary size increases, the mean number of tokens needed to encode one concatenated correction decreases. However, when the vocabulary reaches 16,384, a vocabulary trained on corrections and frequent words outperforms the native vocabulary of mbart-large-50 in terms of the mean parameter.

on the Hugging Face Hub [8]. Our main requirement during the selection process was that the models were trained on the Ukrainian language corpora. We built and compared a few models based on different encoders: **RPN($\mathbf{R_{LARGE}}$)** – RoBERTa Large [9] transferred to Ukrainian using the method from the NAACL2022 paper (Minixhofer et al., 2022), **RPN($\mathbf{XLM_{BASE}}$)** – a smaller version of the XLM-RoBERTa [10] model with only Ukrainian and some English embeddings left. Comparative results for these models can be seen in the table 3

Decoder: We utilized a shallow RedPenNet Decoder stack 2.3 for the decoder part of our architecture. It consists of two layers, and we kept the model depth and dropout parameters the same as the encoders. We utilized a previously computed decoder vocabulary (refer to Section 5.1.2) which was set to a size of $16,384$.

Setup: Tensorflow 2 on a Google Colab TPU instance was used for training and evaluation. In most of the combinations, we conducted pre-training on synthetic data for 20 epochs, followed by training on UA-GEC erroneous data for 30 epochs using a batch size of 32 and a learning rate of $2e{-}5$. Afterward, we fine-tuned the model on UA-GEC (erroneous + error-free) data for 5 epochs, with a batch size of 16 and a learning rate of $5e{-}6$. In the Results section 5.1.5, we present the results of the approaches that showed the best performance.

5.1.4 Evaluation

For the evaluation, the organizers of the Shared Task provided the script based on Errant [11]. Although Errant isn't able to handle specific error types in Ukrainian, it is common practice to use this library for other non-English languages, such as Spanish (Davidson et al., 2020). We have also evaluated scores on the free version of LanguageTool and Hunspell for comparison 3.

In RedPenNet, we implemented a *minimum edit probability* parameter to filter out low-probability edits and to improve precision at the cost of recall. To achieve this, we averaged the probabilities of all predicted edit tokens, as well as the predicted start span and end span for each edit. We assessed the probability of all edits in the model output and discarded those that have probabilities below the *min-*

[8] https://huggingface.co/models

[9] https://huggingface.co/benjamin/roberta-large-wechsel-ukrainian

[10] https://huggingface.co/ukr-models/xlm-roberta-base-uk

[11] https://github.com/chrisjbryant/errant

Approach	min edit prob	dev			test		
		P	R	$F_{0.5}$	P	R	$F_{0.5}$
Hunspell	-	12.9	04.0	08.9	-	-	-
Langtool(free)	-	21.8	05.9	14.2	-	-	-
Langtool(free)+Hunspell	-	19.1	09.0	15.6	-	-	-
MBart-50_{LARGE}		67.51	39.48	59.11	73.06	44.36	64.69
RPN(XLM_{BASE})	0.94	74.9	31.2	58.51	-	-	-
RPN(R_{LARGE})	0.95	75.31	35.11	**61.28**	76.54	41.93	**65.69**
$1\times$RPN(XLM_{BASE}) + $2\times$RPN(R_{LARGE})*	58.5/59	80.28	36.58	**64.80**	80.86	41.03	**67.71**

Table 3: displays the performance comparison between RedPenNet (RPN) and other existing public methods on the UA-GEC+Fluency dataset. The *min edit prob* column shows the edit probability threshold required for accepting an edit.

Approach	UA-GEC+Fluency (dev)		
	P	R	$F_{0.5}$
Synt. pre-train & freeze encoder	08.0	08.9	08.2
Synt. pre-train	07.18	**17,27**	08.13

Table 4: Performance of RPN(R_{LARGE}) after pre-training on synthetic data.

imum edit probability. All edits with probabilities surpassing the threshold were applied. A similar prediction filtering method for GEC was proposed in the GECToR paper and was called "Inference tweaking". And in both cases, the method proved to be effective in improving precision. We also experimented with an iterative correction process, where the output of a previous correction round is used as input for the next one.

Ensembles To create an ensemble of the RedPenNet models, we calculated the average edit probabilities and applied an algorithm that follows the subsequent scenario: 1. For matched edits, we summed their probabilities. 2. For intersecting edits, we choose the more probable one. 3. We kept all remaining non-intersecting edits in the result. Then we tuned the *minimum edit probability* parameter to maximize the F0.5 score on the UA-GEC+Fluency dev set.

5.1.5 Results

We began by pre-training models solely on erroneous data, as proposed in the GECToR research. During this stage, we froze encoders and used synthetic data for pre-training. In the next stage, we unfroze the encoders and trained the models on UA-GEC erroneous data. Our experiments indicate that a low learning rate of $\pm 5\text{e}{-6}$, a small batch size, a few training steps (less than epoch), and an increase in dropouts to ± 0.2 are useful during the initial stages of fine-tuning on a combination of (erroneous + error-free) data. This approach enables us to capture a good checkpoint when the model shifts from recall to precision.

To implement ensembles, we trained two RPN(R_{LARGE}) models and one RPN(XLM_{BASE}) model. The only difference between the two RPN(R_{LARGE}) models is that one of them was trained on erroneous data before being trained on (error-free + erroneous data). The model that was trained only on (error-free + erroneous data) has higher recall.

To enhance the quality of the results, we performed two rounds of iterative correction and applied the ensemble technique to the output of each round. During the first iteration, we set the *minimum edit probability* to 0.585, and for the second iteration, it was set to 0.59. During iterative correction, we selected the value of the *minimum edit probability* parameter that maximizes the precision score.

During the experiment, we demonstrated that our custom architecture, RedPenNet can be applied to the GEC task, with performance that competes with large Seq2Seq models like mbart-large-50 and significantly outperforms classical algorithmic approaches.

5.2 BEA 2019 Shared Task

To further demonstrate the capabilities of the RedPenNet architecture, we applied it to the BEA-2019 Shared Task on English GEC.

Data: The combination of erroneous data obtained from several sources was used for pre-training. We used 20 million samples from the synthetic *tagged corruption* dataset (Stahlberg and Kumar, 2021)[12], approximately 500K English samples from the (Rahman, 2022) study, and around 500K English samples from the *lang-8* dataset. The

[12] https://huggingface.co/datasets/liweili/c4_200m

data was sampled in the following proportions: 50% of *tagged corruption*, 25% of *lang-8*, and 25% of samples from the (Rahman, 2022) study. Only data samples that had at least one error were selected. After pre-training, we used the combination of *W&I+LOCNESS* train set with 13,574 sentences from CWEB(G+S) evaluation dataset (Flachs et al., 2020) [13] that are used as training data for fine-tuning.

Model Configuration: We trained several different-sized RedPenNet models: two based on XLNet [14] pre-trains - RPN(XLNBASE) and RPN(XLNLARGE), and two models based on Muppet Roberta [15]: RPN(MPRBASE) and RPN(MPRLARGE).

The decoder stack consists of two layers, and we utilized a pre-computed decoder vocabulary trained on text corrections extracted from `ABC.train.gold.bea19.m2`. The chosen vocabulary size is 8192.

For pre-training, we conducted 500K steps with a batch size of 128 for BASE models and 64 for LARGE, setting the learning rate to 3e−5. For fine-tuning, we performed 4-6 epochs (depending on the model) to obtain the maximum $F_{0.5}$ score on the *W&I+LOCNESS* dev set. During fine-tuning, we used a batch size of 32 and a learning rate of 5e−6 for all models.

Table 5: BEA-2019 (Test)

Model	P	R	$F_{0.5}$
(Qorib et al., 2022)∗	86.6	60.9	79.9
(Lichtarge et al., 2020)	75.4	64.7	73.0
(Omelianchuk et al., 2020)	79.4	57.2	73.7
(Stahlberg and Kumar, 2021)	77.7	65.4	74.9
(Rothe et al., 2021)	-	-	75.9
RPN(MPR_{BASE})	80,80	56,71	74,47
4×RPN ensemble	**86.62**	54.80	**77.60**

Table 6: A comparison of the performance of various modern GEC approaches, including RedPenNet on the BEA-2019 test set. (Qorib et al., 2022)∗ provides results of combination several systems outputs.

Evaluation and Results: We evaluated RedPenNet models on *W&I+LOCNESS* test set. For our best result, we used an ensemble of RPN(XLNBASE), RPN(XLNLARGE), RPN(MPRBASE) and RPN(MPRLARGE) models. We merged the output using the same scenario as for the UNLP 2023 Shared Task 5.1.4 and determined the best *minimum edit probability* to be 0.68. Interestingly, the second round of processing, in which the outputs from the previous round served as model inputs, did not lead to an improvement in the $F_{0.5}$ score. As it is shown in Table 6, our approach yields state-of-the-art results on BEA-2019 (Test) benchmark, surpassed only by the System Combination result by (Qorib et al., 2022). Furthermore, it is worth mentioning that the RedPenNet ensemble consisting of four BASE/LARGE models outperforms the BEA-2019 (Test) $F_{0.5}$ score of the T5-XXL 11B model from (Rothe et al., 2021) study.

6 Conclusion

While there has been a significant amount of research in the field and many tailored architectures have been proposed, a universally accepted neural architecture for text editing tasks that involves highly similar input and output sequences has yet to be established. This has prevented the creation of an industry standard that can be included in default toolkits for popular machine learning libraries and MLOps tools. Our proposed RedPenNet is an attempt to create a universal neural architecture that is not overloaded with design nuances and is capable of implementing any source-to-target transformation using a minimal number of autoregressive steps. The RedPenNet architecture is a classic transformer, and the only differences lie in how we form decoder input embeddings and interpret outputs and attention scores.

Limitations

While the RedPenNet approach has demonstrated several strengths, such as superior inference capabilities for seq2seq tasks with highly similar inputs and outputs, and some advantages over other SeqToEdits approaches highlighted in the Related Works 4 section, it is not without its limitations:

Due to the tailored architecture of RedPenNet, there are no off-the-shelf solutions for data preprocessing, training, and fine-tuning, as is the case of tasks such as common classification or sequence-to-sequence. Consequently, it is not possible to use convenient tools like the HuggingFace Estimator or cloud platforms for rapid model fine-tuning and deployment.

Additionally, the implementation of a non-greedy beam search approach is complicated by

[13] https://github.com/SimonHFL/CWEB

[14] https://huggingface.co/xlnet-large-cased

[15] https://huggingface.co/facebook/muppet-roberta-large

the presence of multiple sequence outputs.

One more fundamental limitation is that for each edit, the model needs to generate at least two tokens (SEP, token). This does not provide an advantage in reducing the number of autoregressive steps, particularly for short and error-crowded sentences.

Additionally, while RedPenNet has the ability to express any type of input sequence transformation through a number of editing operations, it may not be able to express a single "conceptual" edit, such as transferring a word within a sentence, using a single edit operation. In such cases, two edits — deletion and insertion — may be required to accomplish the desired transformation.

Ethics Statement

Our study focuses on the development of a neural architecture for text editing tasks. The research was conducted in accordance with ethical principles, and no sensitive or personal data was used or collected during the study. The UA-GEC dataset and corpora presented on lang.org.ua used in the study have been obtained from public sources, and their authors assure the privacy and confidentiality of the original texts. The results of the study are intended to improve the efficiency and accuracy of text writing and may be useful for other NLP tasks. We ensure that the study does not raise any ethical concerns or has no negative impact on individuals or groups.

Acknowledgments

We would like to acknowledge and give our thanks to WebSpellChecker LLC for the support and resources allocated to this project. We are also grateful to the WebSpellChecker team, especially Julia Shaptala and Viktoriia Biliaieva, for their assistance and advice during the competition. We are expressing our gratitude to the Program Committee reviewers for organizing the first Shared Task in Grammatical Error Correction (GEC) for Ukrainian, their guidance and insightful recommendations. Also, we would like to mention that the competition took place and this paper was written in Ukraine during wartime. We extend our sincere thanks to all Ukrainian defenders and all supporters of our country and people during these tough times. We also wish to acknowledge our friends who are defending the country with arms — Andrey Boychuk and Georgiy Bondarenko. We would like to honor the memory of Andrey Avilov who died during the liberation of Balakliya.

References

Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, and Vihari Piratla. 2019. Parallel iterative edit models for local sequence transduction. *CoRR*, abs/1910.02893.

Christopher Bryant, Mariano Felice, and Ted Briscoe. 2017. 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)*, pages 793–805, Vancouver, Canada. Association for Computational Linguistics.

Mengyun Chen, Tao Ge, Xingxing Zhang, Furu Wei, and Ming Zhou. 2020. Improving the efficiency of grammatical error correction with erroneous span detection and correction. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7162–7169, Online. Association for Computational Linguistics.

Daniel Dahlmeier and Hwee Tou Ng. 2012. 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*, pages 568–572, Montréal, Canada. Association for Computational Linguistics.

Sam Davidson, Aaron Yamada, Paloma Fernandez Mira, Agustina Carando, Claudia H. Sanchez Gutierrez, and Kenji Sagae. 2020. Developing NLP tools with a new corpus of learner Spanish. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 7238–7243, Marseille, France. European Language Resources Association.

Bohdan Didenko and Julia Shaptala. 2019. Multi-headed architecture based on BERT for grammatical errors correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 246–251, Florence, Italy. Association for Computational Linguistics.

Simon Flachs, Ophélie Lacroix, Helen Yannakoudakis, Marek Rei, and Anders Søgaard. 2020. Grammatical error correction in low error density domains: A new benchmark and analyses.

Tao Ge, Heming Xia, Xin Sun, Si-Qing Chen, and Furu Wei. 2022. Lossless acceleration for seq2seq generation with aggressive decoding.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2020. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation.

Jared Lichtarge, Chris Alberti, and Shankar Kumar. 2020. Data weighted training strategies for grammatical error correction. *Transactions of the Association for Computational Linguistics*, 8:634–646.

Eric Malmi, Sebastian Krause, Sascha Rothe, Daniil Mirylenka, and Aliaksei Severyn. 2019. Encode, tag, realize: High-precision text editing.

Benjamin Minixhofer, Fabian Paischer, and Navid Rekabsaz. 2022. WECHSEL: Effective initialization of subword embeddings for cross-lingual transfer of monolingual language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3992–4006, Seattle, United States. Association for Computational Linguistics.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. Gector – grammatical error correction: Tag, not rewrite.

Muhammad Qorib, Seung-Hoon Na, and Hwee Tou Ng. 2022. Frustratingly easy system combination for grammatical error correction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1964–1974, Seattle, United States. Association for Computational Linguistics.

Chowdhury Rafeed Rahman. 2022. Judge a sentence by its content to generate grammatical errors.

Sascha Rothe, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. 2021. 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)*, pages 702–707, Online. Association for Computational Linguistics.

Felix Stahlberg and Shankar Kumar. 2020. Seq2Edits: Sequence transduction using span-level edit operations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5147–5159, Online. Association for Computational Linguistics.

Felix Stahlberg and Shankar Kumar. 2021. 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*, pages 37–47, Online. Association for Computational Linguistics.

Xin Sun, Tao Ge, Furu Wei, and Houfeng Wang. 2021. Instantaneous grammatical error correction with shallow aggressive decoding.

Oleksiy Syvokon and Olena Nahorna. 2021. UA-GEC: grammatical error correction and fluency corpus for the ukrainian language. *CoRR*, abs/2103.16997.

Oleksiy Syvokon and Mariana Romanyshyn. 2023. The UNLP 2023 shared task on grammatical error correction for Ukrainian. In *Proceedings of the Second Ukrainian Natural Language Processing Workshop*, Dubrovnik, Croatia. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks.

Ziang Xie, Guillaume Genthial, Stanley Xie, Andrew Ng, and Dan Jurafsky. 2018. Noising and denoising natural language: Diverse backtranslation for grammar correction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 619–628, New Orleans, Louisiana. Association for Computational Linguistics.