EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

A multi-task learning framework for efficient grammatical error correction of textual messages in mobile communications

Fayu Pan*, Bin Cao and Jing Fan

*Correspondence: panfayu@zjut.edu.cn

Department of Computer Science and Technology, Zhejiang University of Technology, Hangzhou, China

Abstract

In mobile communications, plenty of textual messages need to be transmitted and processed rapidly. However, messages usually contain noise, which will affect the performance of related applications. Thus, we investigate grammatical error correction (GEC) to correct errors in messages. Unlike recent works, we focus on improving the efficiency of GEC because low time delay is significant in mobile communications. We propose a novel multi-task learning approach to GEC by detecting errors first and then making corrections. Two classifiers are used to serially detect sentence-level and token-level errors, so the correct content can be free from correction operations. We adapt a non-autoregressive decoder to parallelly generate corrected tokens, making the correction stage efficient. Experiments show that our approach is ten times faster than the traditional approach and can achieve a comparable GEC performance.

Keywords: Error detection, Grammatical error correction, Multi-task learning, Denoising, Efficient application

1 Introduction

With the development of mobile communications, reliable data transmission [1, 2] is available and network safety is guarded with anomaly detection [3], and thus applications like APIs recommendation [4] and QoS prediction [5] can be applied in industrial systems. However, the textual message, one of the main transmission objects of mobile communications, usually contains noise made by people's errant inputs. Noisy messages will increase the difficulty of reading comprehension and make applications output wrong results. To transmit accurate messages, we investigate grammatical error correction (GEC) to automatically detect and correct errors in textual messages.

Considering billions of messages are sent in the network every day and a low time delay is an essential requirement of mobile communications, the GEC system should not only improve messages' quality but also take efficiency into account. However, current works on GEC mainly focus on improving performance and use large deep learning models (e.g., T5 xxl with 11B parameters [6]), making the GEC system less efficient when inference. Although more powerful hardware and parallelization can alleviate this problem, it will increase costs significantly. Therefore, we aim to build an efficient GEC system with limited hardware that can correct errors rapidly.

Many works [6–8] consider GEC as a machine translation problem by regarding the noisy text as the source language and the error-free text as the target language. Following neural machine translation (NMT) approaches, a traditional GEC system is built with an attention-based sequence-to-sequence model [9], which contains an encoder for the input text's semantic representation and a decoder for the corrected text's generation. Since the decoding stage is auto-regressive, which means the generation of the current token relies on the previously generated ones, the inference speed is slow. Our experiment shows that the processing time of single text with NMT can reach hundreds of milliseconds, making GEC service unacceptable in mobile communications because the control plane latency requirement for 5G is only 50ms [10].

Recently, some works [11, 12] consider GEC as a local sequence transduction problem and GECToR [12] achieves state-of-the-art performance. They use sequence labeling (SL) models to detect and correct errors synchronously by predicting labels to each token in the text. Each label is usually composed of an edit tag (e.g., keep, delete, append, and replace) and one token. For example, 'Thanks you' can be corrected into 'Thank you' by assigning label 'replace||Thank' to token 'Thanks' and label 'keep' to token 'you.' Since the labeling stage can be parallel, SL-based approaches are more efficient than NMT.

However, there still exist some limitations to SL-based approaches. As the vocabulary of labels is built with the statistic of most common corrections from corpora, the model will suffer from the out-of-vocabulary (OOV) problem [13], which means the model is unable to correct the error when the requisite label is not in the pre-set vocabulary. Besides, SL-based approaches need multiple iterations of correction to correct some errors because these errors need more than one label. For example, in first iteration, 'Thank your help.' is corrected into 'Thanks your help.' with label 'replace||Thanks' to token 'Thank,' and then 'Thanks your help.' is corrected into 'Thanks for your help.' with label 'append||for' to token 'Thanks' in second iteration. In GECToR, a text could be corrected up to 5 iterations, which means to serially process 5 individual texts. It could be more efficient if all requisite tokens to an error are gained in one iteration. Last but not least, current SL-based approaches do not consider the correctness of the whole text and just label all text in token-level. In general, apart from encoding the text, the computational cost of token-level labeling is L (L is the length of the text) times higher than classifying a text. Apparently, the correct text should be free from token-level computation if it is previously classified as containing no errors. Considering that the correct text usually takes the majority part in the real scenario, SL-based approaches can be more efficient by checking text's correctness first.

To further improve the efficiency of GEC, we get inspiration from how people correct text and divide the task into three steps. Usually, given a text, we will first scan it and give a preliminary judgment of its correctness. If we consider the text incorrect, we will carefully check each token and find errors. Finally, we will generate corrections with the context of each error. To implement such progress, we build a multi-task learning framework and integrate three tasks into one model: sentence error discrimination (SED), grammatical error detection (GED), and GEC. SED and GED are two auxiliary tasks for error detections, and thus we only need to correct detected errors in GEC. We apply SpanBERT [14] as the encoder to extract sentence-level and token-level features from the source sentence. For SED, a discriminator is designed to classify the sentence and it will output a probability of correctness. The sentence is considered to be correct if the probability is higher than our pre-set threshold. Then we use a detector to handle GED, which detects errors in token-level. Each token will be classified as a correct token or one of our defined error types. Next, we record each detected error's start and end positions in the source sentence and use a corrector to generate corrections with each error's context information. We represent each error with three features as the hidden state of the error itself and the information before and after the error. Then we send error features with position embeddings into a feed-forward network to concurrently generate all tokens in a correction, which avoids the requirement of multiple iterations of correction. Finally, we can modify the source sentence by replacing incorrect tokens with generated corrections, thus getting the corrected sentence.

Following our previous work [15], we complete the implementation details of our model and conduct more experimental analysis in this paper. We make three new improvements in this paper: changing words' representation, assigning loss weights to the tasks of our model, and changing the training strategy. Besides, we notice that text error detection is important but achieves little attention in recent research, so we make an empirical study on error detection. With the improvements in this paper, the $F_{0.5}$ score of our approach has changed from 50.4 to 53.0 in CoNLL-14 dataset [16] and from 52.1 to 56.1 in BEA-19 test dataset [17]. Despite that our approach's performance cannot reach the current state-of-the-art performance, our approach shows advantages in processing speed. Our contributions in this paper are as follows:

1. We propose a novel multi-task learning approach to GEC, and our approach is also suitable for both sentence and token-level error detection. Our approach can detect and correct partial errors in messages, thus improving the performance of textual applications in mobile communications.
2. Our model is efficient in correcting messages and can satisfy mobile communications' low time delay requirement. Our model is at least ten times faster than the traditional NMT approach and nearly 30% faster than GECToR, a SL-based approach, in processing single text.
3. We make an empirical study on pre-trained language models' performance in the task of SED and show the power of grid search in improving its performance. Moreover, we show more research should be done to improve SED.

The organization of this paper is as follows. Section 2 introduces the definition of our problem and our approach to hierarchically solving GEC. Section 3 reports our modifications to previous work and the experimental results of our method in terms of performance and efficiency. Besides, current methods' performance in SED is explored in Sect. 3. Section 4 introduces related works and Sect. 5 concludes the paper.

2 Method

2.1 Problem definition

GEC can be defined as converting an input text sequence $X = (x_1, \ldots, x_n)$ into the target sequence $Y = (y_1, \ldots, y_m)$. X equals Y when the input is correct. Instead of generating

Y from scratch, our goal is to directly modify X and eliminate the differences between X and Y since their similarity is over 80% [7]. In order to only correct the errors in the incorrect sentences, we introduce two auxiliary tasks named SED and GED for sentence-level and token-level error detection, respectively. Thus our approach to GEC can be divided into three steps: checking sentence's correctness, finding errors' positions, and generating corrections. As a result, correct sentences will not be modified and incorrect sentences can be corrected by replacing errors with corrections.

For the SED task, we have to predict whether the sentence is correct or not and it is a binary classification problem. A bool type value Y^{tf} is defined as the label of the sentence's correctness, and 1 represents correct.

GED is originally used for finding incorrect tokens in the sentence, but we can get each error's position by post-processing its results. Note that the error mentioned here is defined in span level, which means one error contains all continuous incorrect tokens. Instead of a correct or incorrect label, we assign each token with an error type to define incorrect tokens because error types are more specific. According to the ERRANT toolkit for GEC [18], which classifies edits for corrections as unnecessary, missing, and replacement edits, we define three error types as redundancy (R), missing (M), and word selection (WS) errors. We apply BIO tags [19] to label the start and the inner of an error because one error can contain more than one token. The target of GED is to get a sequence of labels Y^{et}, in which y_i^{et} represents the label for token x_i. Once we get Y^{et}, we can know which token is incorrect and simply find errors' positions.

The last task is to generate corrections for those detected errors in GED, and we construct a structure called *patch* to describe the correction. Each *patch* is operated on X and is a triple patch $= (s, e, Y\prime)$, which means the content between x_s and x_e (not include x_s and x_e) should be corrected into $Y' = (y_i, ..., y_j)$. Since we can obtain the positions s and e from the previous task, we only need to generate Y' in this step. Then we can correct incorrect sentences according to the modifications of *patches*.

2.2 Preprocessing

According to our definition of GEC, three target training labels need to be exacted from the parallel training corpus. Initially, the training data should be formatted into source sentence X and target sentence Y pairs. Suppose more than one target sentence is annotated. In that case, we will choose the target sentence with minimal Levenshtein distance, which means minimal modifications need to be applied to the source sentence to make it correct. Besides, A special token '[CLS]' is added to the front of each sentence for the SED task, and '[SEP]' is appended to the back of each sentence as the symbol of end.

By checking whether X is equal to Y, we can get the sentence-level label Y^{tf}. Then we need to find errors in the incorrect sentence and get target corrections by comparing X with Y. We apply SequenceMatcher[1] to get the transductive operations of converting X to Y. Each operation is a quintuple $(\mathrm{tag}, s_X, e_X, s_Y, e_Y)$ where *tag* represents the transductive relationship from $(x_{s_X}, ..., x_{e_X-1})$ to $(y_{s_Y}, ..., y_{e_Y-1})$. The progress of getting target

[1] https://docs.python.org/3/library/difflib.html#difflib.SequenceMatcher

Table 1 Examples of data conversion

X	[CLS] He is the first of all in the list . [SEP]
Y^{et}	[O O O O O B-R I-R O O O O O]
Y	[CLS] He is the first in the list . [SEP]
Operations	(‘delete’, 5, 7, 5, 5)
Patches	[(4, 7, (‘[SEP]’))]
X	[CLS] This car is more faster than that [SEP]
Y^{et}	[O O O O B-WS O O O B-M]
Y	[CLS] This car is much faster than that one . [SEP]
Operations	(’replace’, 4, 5, 4, 4) (’insert’, 8, 8, 8, 10)
Patches	[(3, 5, (‘much’,‘[SEP]’)), (7, 8, (‘one’, ‘.’, ‘[SEP]’))]

X and *Y* are inputs. Y^{et} and *patches* are outputs

labels of incorrect sentences is shown in Algorithm 1, and Table 1 shows two conversion examples.

```
Algorithm 1: Conversion for Incorrect Training Data
Input  : $X = (\text{‘[CLS]’}, x_1, ..., x_n, \text{‘[SEP]’})$, $Y = (\text{‘[CLS]’}, y_1, ..., y_m, \text{‘[SEP]’})$
Output: $Y^{et}$ and $patches$
Let $Y^{et} = [O] \times (n+2)$, $patches = []$.
$operations = SequenceMatcher(X, Y)$
for $(tag, s_X, e_X, s_Y, e_Y)$ in $operations$ do
    if $tag ==$ ‘delete’ then
        $Y^{et}[s_X]$=B-R                                   // Redundancy Error
        $Y^{et}[s_X + 1 : e_X]$=I-R
        $patches \leftarrow (s_X - 1, e_X, (\text{‘[SEP]’}))$
    else if $tag ==$ ‘insert’ then
        $Y^{et}[s_X]$=B-M                                   // Missing Error
        $patches \leftarrow (s_X - 1, e_X, (Y[s_Y : e_Y], \text{‘[SEP]’}))$
    else if $tag ==$ ‘replace’ then
        $Y^{et}[s_X]$=B-WS                                  // Word Select Error
        $Y^{et}[s_X + 1 : e_X]$=I-WS
        $patches \leftarrow (s_X - 1, e_X, (Y[s_Y : e_Y], \text{‘[SEP]’}))$
return $Y^{et}$ and $patches$
```

We use BIO tags to label errors, in which ‘B’ represents the beginning of an error, ‘I’ represents the inner of an error, and ‘O’ represents not an error. Usually, we need two plus one types of tags to label one kind of error. Taking the first example in Table 2 as an example, we use ‘B-R’ and ‘I-R’ to label a redundancy error and other correct tokens are labeled with ‘O.’ Although there are three kinds of errors, only 6 tags (‘B-R,’‘I-R,’‘B-M,’‘B-WS,’‘B-WS,’‘O’) are defined in our approach and the ‘I-M’ label is not existing because the missing error only occurs between two words and we just label ‘B-M’ on the second word. The added token ‘[SEP]’ at the end of *X* and *Y* can handle the situation of missing words at the end of a sentence. Compared with previous works on token-level error detection [20, 21], which label correct tokens with 0 and incorrect tokens with 1, our approach can not only find incorrect tokens but also detect error types. Previous works also label missing errors by labeling 1 to the first token after the missing error, but this will cause a problem that we cannot know whether the token is incorrect or a missing error occurs before the token. On the contrary, our approach can avoid this problem.

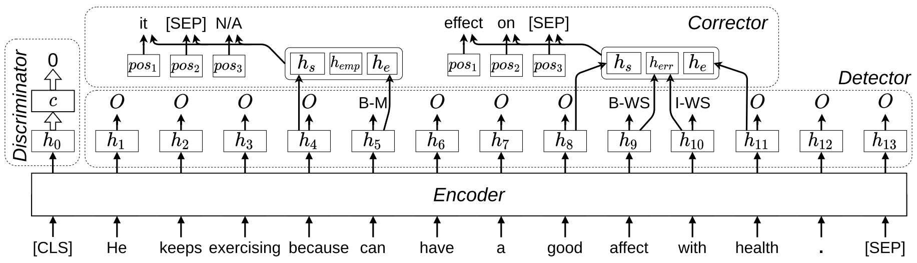

Fig. 1 The architecture of our model. The input sentence is first checked to be incorrect by the Discriminator, then located errors' positions by the Detector, and finally corrected by the Corrector. Two errors are detected and the maximum correction length *M* is set to 3 in this example. *N/A* refers to any token as it occurs after token '[EOP]'

In addition to the error type labels Y^{et}, the *patches* for error corrections are also collected from the transductions. To control the correction's length, the special token '[SEP],' which is '[EOP]' in our previous work, is added to the end of each patch. We change '[EOP]' to '[SEP]' because '[SEP]' is originally designed as a symbol of end in SpanBERT, while '[EOP]' is designed by us and its embedding vector is not pre-trained. The start position s in the *patch* is one position in front of the first incorrect token of the error and e is one position behind the last incorrect token for easier access to the error's surrounding context and it can describe the missing error's position. Different from our previous work, tokens in the patch for the missing error are changed from ('[NONE],''[EOP]') to ('[SEP]') for concision.

2.3 Architecture

It can be less efficient if we use three separate models to handle our tasks, so we design a multi-task learning framework to integrate three tasks because they all need to semantically represent the source sentence first. The architecture of our model is shown in Fig. 1. Three headers, named Discriminator, Detector, and Corrector, are involved in processing the corresponding tasks by sharing one powerful encoder.

2.3.1 Encoder

SpanBERT [14], instead of the widely used pre-trained language model BERT [22], is applied as the encoder in our model because SpanBERT can achieve better performance in downstream NLP tasks than BERT[14] and the pre-training task of SpanBERT is similar to our error correction task. The architecture of SpanBERT is the same as BERT, which is a stack of several transformer encoder [9] layers, but SpanBERT does not involve the next sentence prediction task during pre-training. Moreover, SpanBERT enhances the masked language model in pre-training by masking a random span length of words and predicting the masked words with span boundary objective.

Since SpanBERT is self-supervised pre-trained with a large number of corpora, it can provide a reliable encoding of text and we only need to fine-tune it with our tasks. Given a text, the first step is tokenization which breaks down the text into the smallest unit (token). Then we embed each token to represent each token with a vector. Finally, we send all tokens' vectors into the encoder. After encoding, each input token x_i will be converted to a vector h_i which can represent its semantic information in the text.

2.3.2 Discriminator

The discriminator is used to check whether the input sentence is correct. We follow the general usage of BERT or SpanBERT for sentence classification, which is to pool the hidden state of token '[CLS]' and feed it into the classifier to get the sentence's probability of correctness $P^{\mathrm{tf}}(X)$:

$$c = \tanh(W^{\mathrm{pool}}h_0 + b^{\mathrm{pool}})P^{\mathrm{tf}}(X) = \mathrm{sigmoid}(W^{\mathrm{cls}}c + b^{\mathrm{cls}}) \tag{1}$$

where h_0 is the hidden state of token '[CLS]' and the output c can represent the embedding of the whole sentence. We manually set a threshold $P^{\mathrm{tf}}_{\mathrm{thres}} \in (0,1)$ here to control the boundary of classification, and the sentence is regarded as correct if $P^{tf}(X)$ is higher than $P^{\mathrm{tf}}_{\mathrm{thres}}$. According to the target domain, a referenced threshold can be obtained by grid search in the development set with the highest accuracy. If the sentence is detected to be correct, we will early stop the whole procedure and output the source sentence as the GEC result.

2.3.3 Detector

To find errors in the incorrect sentence, we apply a sequence labeling layer to do token-level error detection, which is to classify each token based on the hidden states h_i. Conditional Random Field (CRF) [23] is applied to add constraints to the final output. Thus, an unreasonable output sequence (e.g., label 'I-R' appears after 'O') will be less likely to occur. The progress of detecting each token's error type $P^{\mathrm{et}}(x_i)$ can be described as:

$$P^{\mathrm{et}}(x_i) = \mathrm{CRF}(W^{\mathrm{et}}h_i + b^{\mathrm{et}}), i \in [1, n+1] \tag{2}$$

The index i does not start from 0 here as the target label of the first token '[CLS]' is always 'O.'

The post-processing step after error detection is to locate each error's start and end position, which is similar to the conversion from Y^{et} to *patches*' positions in Table 1. For a redundancy error, if x_i is labeled 'B-R' and $(x_{i+1}, ..., x_j)$ are labeled 'I-R,' then we set one *patch*'s start position $s = (i-1)$ and end position $e = (j+1)$ ($j = i$ when 'O' is after 'B-R'). The process is the same for word selection errors, but the end position should not move one place back for each detected missing error because it is at the right position. Besides, all predicted labels can be 'O' if the discriminator wrongly classifies a correct sentence or the detector fails to find errors in the incorrect sentence. In such cases, we will consider the sentence correct.

2.3.4 Corrector

Our corrector is inspired by the pre-training task of SpanBERT and can use detected errors' surrounding context to generate corrections. Through error detection, we can locate each error with the start position s and the end position e in the sentence. Therefore, h_s can represent the context before the error, and h_e can represent the context after the error, while the average hidden states between h_s and h_e represent the error itself. In brief, we can represent a span-level error r with:

$$r = [h_s, h_{\text{err}}, h_e] \quad \text{where}$$
$$h_{\text{err}} = \begin{cases} \text{mean}([h_{s+1}, ..., h_{e-1}]); , & se + 1 \\ h_{\text{emp}}, & s = e + 1 \end{cases} \tag{3}$$

in which h_{emp} is a learnable vector. It is used for missing errors because the encoder cannot represent the missing tokens.

Our main innovation compared with SpanBERT is that we additionally involved the middle state h_{err} for the representation of a span. Following SpanBERT, we use position embeddings and two layers of feed-forward network with layer normalization to generate the target correction $Y' = (y'_1, ..., y'_p)$:

$$P(y'_i) = MLP([r, E^{\text{pos}}(i)]), i \in [1, p] \tag{4}$$

where p is the length of target tokens in a *patch* and $E^{\text{pos}}(i)$ represents the position embedding for the ith token to generate. The correcting stage is non-autoregressive and could run in parallel to improve efficiency.

However, unlike SpanBERT, which can control the output's length with each masked span's length, we do not know the real length of each correction when inference. Thus we manually set a constant M and let p equal M when inference, which limits all corrections' maximum length to M. Besides, with the special token '[SEP]' at the end of each patch, we can control the output length when inference by clipping tokens before '[SEP].'

Finally, we construct the *patch* for each error with positions s and e from the detector and correct tokens Y' from the corrector. Then we only need to replace incorrect tokens according to the patch, and the source sentence X can be corrected to the target sentence Y.

2.4 Multi-task learning

There are three tasks in our approach, and each task has a corresponding loss function. We use the discriminator to do SED and the loss loss^{dis} is defined with binary cross-entropy loss since we output the probability to a binary classification problem. The detector handles GED and its loss loss^{det} is calculated with the path search score of the CRF layer. Cross-entropy loss loss^{cor} is used for the corrector by comparing target tokens and predicted logits. We integrate three tasks into one multi-task learning model, and the overall loss function is defined as:

$$\text{loss} = \alpha\text{loss}^{\text{dis}} + \beta\text{loss}^{\text{det}} + \gamma\text{loss}^{\text{cor}} \tag{5}$$

where hyperparameters α, β, and γ are the weights for the corresponding tasks. Besides, we limit the sum of all weights to the number of tasks, which is 3 in our model. To those correct inputs during training, only $loss^{dis}$ is calculated.

In addition to manually setting weights, we also design a strategy to automatically calculate each loss's weight when training. As we know, the higher the loss, the worse the performance. Since all three tasks are important to our final results and the difficulty of each task is different, we expect that the task with higher loss can have a higher loss

Table 2 A Statistic of Used Corpora

Corpus	Sentence number	Avg. token number	Avg. error ratio	Error number	Avg. patch length
Training sets					
FCE	28,350	16.0	62.47	2.20	2.17
Lang-8	1,037,561	11.4	47.98	1.94	2.36
NUCLE	57,151	20.3	37.87	1.97	2.12
W+L	34,308	18.3	66.26	2.46	2.20
Development sets					
CoNLL-13[24]	1381	21.1	81.4	2.60	2.13
BEA-19 (dev)	4384	19.8	64.3	2.38	2.21
Evaluation Sets					
CoNLL-14 A1	1312	23.0	72.2	2.21	2.11
CoNLL-14 A2	1312	23.0	86.1	2.68	2.14
BEA-19 (test)	4477	19.1	–	–	–

weight so that the loss can drop faster. Thus, the strategy to calculate weights is defined as:

$$\begin{aligned} \text{total\_loss} &= \text{loss}^{\text{dis}} + \text{loss}^{\text{det}} + \text{loss}^{\text{cor}} \\ \alpha &= 3 \times \text{loss}^{\text{dis}}/\text{total\_loss} \\ \beta &= 3 \times \text{loss}^{\text{det}}/\text{total\_loss} \\ \gamma &= 3 \times \text{loss}^{\text{cor}}/\text{total\_loss} \end{aligned} \tag{6}$$

where 3 is the number of tasks. With this strategy, the losses' weights can be dynamically upgraded during training.

The training stage of three tasks can be parallel as we know which inputs are incorrect and where the incorrect tokens are located. Differently, the inference stage is sequential and we process sentences in the order of the encoder, discriminator, detector, and corrector, but we can filter those inputs which are detected to be correct by the discriminator to save time.

3 Experiments

3.1 Datasets

Following the restricted track of BEA-2019 shared task for GEC [17], we use FCE [25], NUCLE [26], Lang-8 [27], and W &I+LOCNESS (W+L) [17] corpora as our training sets. The development sets are composed of CoNLL-2013 [24] test set and BEA-2019 development set because we want our model can generalize to different kinds of errors. Same as recent works, we evaluate our model's GEC performance on CoNLL-2014 test set [16] with MaxMatch (M^2) scorer[28] and BEA-2019 test set with ERRANT scorer. Two annotators annotate CoNLL-2014 and the M^2 scorer will compare each sentence from the model's output with two annotations and record the result with better performance. The evaluation of BEA-2019 is operated on CodaLab[2]. We use CoNLL-2014 test

[2] https://competitions.codalab.org/competitions/20228

set to evaluate the performance in the SED task. The statistic of used datasets is shown in Table 2. Error Ratio means the percentage of incorrect sentences. Error Number means the average error number in an incorrect sentence. Patch Length means the average token number in patches (with '[SEP]').

3.2 Implementation details

We implement our model with Transformers[3]. Words in the text are tokenized into WordPiece format [29], and the vocabulary for embedding and generating contains 28,996 tokens. With WordPiece and a large vocabulary, our approach is less likely to suffer from OOV problem in correction generation stage because almost all words can be obtained with WordPieces (e.g., word 'Advantage,' which is not in the vocabulary, can be generated with WordPieces 'Ad,' '##vant,' and '##age,' in which '##' means the content after it should connect to previous WordPiece.).

We apply AdamW [30] optimizer and use warm-up for the first 5% of total training steps. We set the maximum length M of each correction to 4 according to the average length of patches as shown in Table 2. As the sentence-level error ratio can be quite different according to the corpus, we apply grid search in the corresponding development set to determine threshold $P^{\text{tf}}_{\text{thres}}$ and then use it for evaluation.

Different from our previous work [15], we re-implement the code of our model and make three main improvements as follows:

1. We change the representation of words. As mentioned above, words will be tokenized into WordPieces and then sent to the encoder. In our previous work, we represent each word with corresponding WordPieces' hidden states, which means the operation objects of the detector and the corrector are WordPieces instead of words. For example, we should assign three labels to the word 'Advantage,' which is tokenized into three WordPieces, in error detection. In this paper, we represent each word with its first WordPiece's hidden state, which is inspired by GECToR. We implement it by a selection operation after encoding and the input of the encoder is the same as the previous. Such an improvement can reduce the difficulty of our tasks because previously, we needed to correctly label all WordPieces of a word to find an error, but we only need to label one currently.

2. We change the training strategy of our model. In our previous work, we train the model with all training datasets for 3 epochs with a learning rate of 3×10^{-5}. In this paper, we divide the training procedure into two stages. In the first stage, we train our model with dataset Lang-8 for 5 epochs with a learning rate of 3×10^{-5} and choose the checkpoint which has the highest $F_{0.5}$ in merged two development datasets as the model's parameters. In the second stage, we use the other three training datasets to further train our model for 5 epochs with a learning rate of 1×10^{-5} and choose the best checkpoint as our final model's parameters. The first stage is for coarse training because all parameters except the encoder are randomly initialized, so we use a large learning rate and the largest dataset Lang-8. The second stage is for further fine-tuning, so we use a small learning rate and the rest training datasets.

[3] https://huggingface.co/transformers/index.html

Table 3 Experimental results on GEC

Work	Method	CoNLL-14			BEA-19		
		P	R	$F_{0.5}$	*P*	*R*	$F_{0.5}$
Zhao et al. [7]	NMT	65.5	33.2	54.9	-	-	-
LaserTagger [31]*	SL	50.9	26.9	43.2	53.4	38.5	49.6
GECToR [12]**	SL	**66.8**	33.7	**55.8**	**64.2**	47.0	**59.8**
Chen et al. [32]	SL+NMT	66.0	24.7	49.5	62.7	38.6	55.7
Previous (BERT)[15]	SL	55.1	32.3	48.3	50.6	40.5	48.2
Previous (SpanBERT)[15]	SL	57.2	34.1	50.4	54.5	44.3	52.1
Our approach (BERT)	SL	56.3	35.0	50.2	54.8	48.2	53.4
Our approach (SpanBERT)	SL	59.8	**36.5**	53.0	57.8	**50.4**	56.1

* The result is implemented by Chen et al. [32]

** The result is implemented by us with official released code with BERT

3. We assign loss weights to three tasks in our model when training. We test the effect of loss weights and try to manually or automatically set weights, which is introduced in sect. 2.4. We find manually setting weights works better and the values of α, β, γ are set to 0.5, 1, and 1.5 respectively in our final model. We will discuss more about loss weights in sect. 3.5.

3.3 Grammatical error correction results

The GEC performance of our model is shown in Table 3. Bold values in the table represent the best performance in each column. All results listed here are the best non-ensemble results from each work and pseudo data is not used. We compare our results with recent works without using pseudo data for further pre-training because the amount of used pseudo data among different works can vary greatly (from 9M to 260M). There are two main categories of methods for comparison, which are based on NMT or SL. We classify our approach as a SL-based method as we apply SL model for token-level error detection, and we do not use the auto-regressive decoder (e.g., LSTM) for error correction.

As shown in Table 3, GECToR has the best performance and our approach can only achieve a medium performance in both CoNLL-2014 and BEA-2019 test sets. However, our approach achieves great improvement in both precision and recall compared with our previous work because of the changes mentioned in sect. 3.2. The gap between our approach and GECToR is reduced. Our approach shows weakness in precision but is outstanding in recall, which means our approach prefers to regard the sentence as wrong and make corrections. We also compare BERT and SpanBERT's fine-tuning performance in our approach. SpanBERT can achieve apparent improvement in both precision and recall, which shows that the pre-training task of SpanBERT is more suitable for our approach.

In Table 4, we further compare our approach's performance with GECToR in BEA-2019 test set, which is calculated online with the ERRANT toolkit. Our approach outperforms GECToR in both token-level and span-level detection but shows a huge drop in the final span-level correction aspect, which can indicate that our detector performs well, but the corrector fails to correct all detected errors. One reason for this can be

Table 4 Detailed results on BEA-19

Work	Evaluation aspect	*P*	*R*	$F_{0.5}$
Our approach(SpanBERT)	Span-level correction	57.8	50.4	56.1
	Span-level detection	78.2	63.6	**74.8**
	Token-level detection	88.0	66.7	**82.7**
GECToR[12]	Span-level correction	64.2	47.0	**59.8**
	Span-level detection	75.2	52.4	69.2
	Token-level detection	84.4	54.2	76.0

Table 5 Detector's results on finding error positions

	CoNLL-14 A1			CoNLL-14 A2		
	P	*R*	$F_{0.5}$	*P*	*R*	$F_{0.5}$
BERT	43.4	31.6	40.4	59.2	29.8	49.5
SpanBERT	43.4	32.1	40.6	59.4	30.4	49.9

that the released SpanBERT-base model does not include the weight of the pre-training task, and we cannot transfer its obtained knowledge of text generation. Therefore, the problem of generating corrections under a certain context is difficult to our model and should be further improved.

Before error correction, the discriminator and the detector are applied for hierarchical error detections in our approach. The discriminator is related to the task of SED, and its accuracy in CoNLL-2014 test set with SpanBERT is 94.05%. For the detailed result in SED, we will discuss it in sect. 3.6. Since the error type label is only defined in our work and the discriminator will filter some correct sentences, which means not all sentences in the test set are processed by the detector, it is hard to compare the detector's performance with other works. Besides, the detector serves as the function of providing error positions to the corrector, so we only report the detector's results on finding error positions, which is shown in Table 5. The results are not decisive to the final performance of GEC because the corrector may fix errors made by the detector (e.g., the detector regard a correct token as an error and the corrector fix the mistake by generating the original correct token).

3.4 Inference time analysis

Efficiency is significant to mobile communications where time delay must be controlled to an acceptable level. Since GEC serves as an intermediate procedure in communications, its inference speed must be as fast as possible. We analyze the inference time of our approach on CoNLL-2014 test set and make comparisons with other approaches to explore whether our approach is efficient. All experiments are done with one NVIDIA V100 GPU in CUDA 10.2 environment[4] and we run three times with each setting of batch size to reduce error.

[4] We conduct the experiment with the instance type ecs.gn6e-c12g1.3xlarge in Alibaba Cloud.

Table 6 Modules' inference time on CoNLL-14

Module	**Inference time (s)**			
	1	**8**	**16**	**32**
Encoder	14.26	2.03	1.00	0.53
Discriminator	0.56	0.08	0.05	0.03
Detector	2.84	0.98	0.76	0.63
Corrector	1.49	0.32	0.18	0.11
Total	20.76	3.96	2.75	2.47

1/8/16/32 refers to the setting of batch size

Table 7 Inference time and performance on CoNLL-14

Work	**Parameter**	**Setting**	**Inference time (ms/s)**				**Performance**		
	Number		**1**	**8**	**16**	**32**	***P***	***R***	$F_{0.5}$
Zhao et al.[7]*	97M	Beam 1	245/321	317/52	353/29	415/17	63.3	33.7	53.9
		Beam 4	431/566	646/106	743/61	878/36	64.4	33.3	54.3
Chen et al.[32]	334M	Beam 5	104/137	207/34	256/21	390/16	66.0	24.7	49.5
GECToR[12]*	112M	4 iters	20.65/27.09	41.59/6.82	60.37/4.98	101.95/4.18	**66.8**	33.7	**55.8**
Previous[15]	133M	–	**14.89/19.53**	**24.09/3.95**	37.56/3.08	70.24/2.88	57.2	34.1	50.4
Our approach	133M	–	15.82/20.76	24.15/3.96	**33.54/2.75**	**60.24/2.47**	59.8	**36.5**	53.0

1/8/16/32 refers to the setting of batch size

* Trained and evaluated with official released code. Pre-training and ensemble are not applied

We first analyze the time cost of each module in our model and the results are shown in Table 6. The encoder takes the major part of total inference time because the SpanBERT we applied takes up the whole model's majority of parameters. The discriminator takes the least time in our model, and it can save the whole model's inference time by filtering correct sentences. Actually, 160 sentences are discriminated to be correct, which means nearly 12% of the test set can be free from running token-level error detection and error correction. Considering the error ratio in this test set is high, more sentences can be early output by our model in less noisy scenarios, which can further improve our approach's efficiency. The cost of the detector is high as we use the CRF layer to optimize the output sequence. The corrector does not take much time because it is non-autoregressive and it only needs to generate corrections for detected errors instead of generating a whole sentence like NMT. Besides, the total inference time is not equal to the sum of four modules' times, and the difference value is made by post-processing and data transmission between GPU and CPU.

Next, we compare our approach's inference time with three typical GEC systems in Table 7. The time on the left of the slash is the average milliseconds of processing one batch and the right one is the total seconds of processing all sentences. The work of Zhao et al. [7] is based on NMT, and it uses 6 transformer layers for both encoder and decoder. GECToR [12] is based on SL method with BERT as encoder. The work of Chen et al. [32] is a hybrid model with one SL model (based on Roberta [33]) for token-level error detection and one NMT model (based on Transformer (big)[9]) for error correction. Our experiment on inference time can compare with the results in [32] because we use

Table 8 Inference time (second) with different sentence lengths

	5	10	15	20	25	30
GECToR	3.59	4.20	5.00	5.68	6.48	7.27
Our approach	1.22	1.57	2.07	2.47	2.83	3.32

The batch size is set to 32

the same type of GPU with same CUDA setting to test and all models are implemented with Pytorch. Besides, our approach's inference time is different from our previous work because we re-implement our code and our current performance is significantly changed compared to the previous.

As shown in Table 7, our approach and GECToR are much faster than Zhao et al. [7] and Chen et al. [32]. Although beam search is widely applied for improving the performance of NMT models, the cost of time can increase multiply compared with greed search (beam size equals one). Besides, the NMT model's inference speed in processing a single sentence is up to several hundreds of milliseconds, which can limit the GEC model's application in mobile combinations. Although a larger batch size can reduce the total running time, the inference time of each batch increases, so we cannot get the result of a single text faster than one sentence per batch. The hybrid model by Chen et al. [32] is proposed to improve the efficiency of GEC and it does show improvement to the NMT-based model, but its inference speed is much slower than ours. Finally, we compare our approach with GECToR to show if our approach is more efficient in SL-based approaches. Since GECToR can only correct each incorrect token with one token in one iteration, it needs multiple iterations to maximize performance, which is to re-correct those sentences detected as incorrect. Considering the numbers of corrections made by GECToR are 740, 944, 993, 1004, and 1005 from 1 iteration to 5 iterations respectively, we choose the setting of 4 iterations for comparison. However, our approach can correct an error with a maximum of $(M-1)$ tokens in one turn, which is more efficient. We can see our approach is nearly 30% faster than GECToR and the inference time per sentence is only 15.82 milliseconds when batch size is set to 1, which indicates that our approach is more suitable for improving the text quality in real-time communication scenarios.

To further explore the efficiency of our approach, we compare our approach with GECToR in processing sentences of different lengths. We test the inference time for sentences of lengths 5, 10, 15, 20, 25, and 30. For each length, we randomly sample 2000 samples of that length from Lang-8 dataset for testing. The results are shown in Table 8. Our approach shows a significant advantage in inference time compared to GECToR. Besides, our approach's time increases for every 5 increases in length, which is less than GECToR. Thus the time difference increases with the increase in length. This experiment shows that our approach is efficient in correcting sentences with different lengths.

3.5 Ablation study

In order to evaluate the influence of different strategies and settings on our approach, we perform an ablation study to our approach on the CoNLL-2014 test set and the results are presented in Table 9. Each setting in this table is based on the previous one.

Table 9 Ablation study results on CoNLL-14

	P	*R*	$F_{0.5}$
Our approach	59.8	36.5	**53.0**
w/o loss weights	59.7	36.4	53.0
w/o two stages training	58.0	34.4	51.0
w/o CRF layer	58.1	29.6	48.7
w/o discriminator	62.5	16.5	40.1
w/o correct sentences	56.4	26.4	46.0

Table 10 Impact of loss weights

Type	α	β	γ	*P*	*R*	$F_{0.5}$
Automatic	–	–	–	57.1	35.1	50.7
Manual	0.5	1	1.5	59.8	36.5	**53.0**
Manual	1	1	1	59.7	36.4	53.0
Manual	1.5	1	0.5	57.4	35.3	51.0

The first line in Table 9 is our final approach, which sets loss weights 0.5, 1, 1.5 to α, β, γ and uses two stages training strategy. After removing loss weights, which is to set all loss weights to 1, there is a slight decrease in performance. Then we further remove the two stages training strategy. This is to train the model with all training datasets for 5 epochs and select the checkpoint with the highest score on development datasets. We can see there is a 2.0 points decrease in $F_{0.5}$, which proves that the two stages training strategy can help improve GEC performance. Compared with our previous work, which uses all training datasets to train the model for 3 epochs and achieves a $F_{0.5}$ score of 50.4, we can see a longer training time with a selection strategy on parameters can help improve performance. Next, we evaluate the influence of the CRF layer, which is involved in ruling the output of the detector. We can see the recall drops a lot without the CRF layer, leading to the decrease in $F_{0.5}$. This proves that the CRF layer is significant for our model in finding errors. Then we directly remove the discriminator to see its impact. We can see the precision increases, but the recall drops a lot, making the $F_{0.5}$ score decrease 8.6 points. It happens because about half of the sentences in training datasets are correct, which means the target labels of the detector are all tag 'O.' Thus the detector is less likely to detect errors. However, when we use the discriminator, the detector only processes incorrect sentences when training, so the recall value is higher. This can be proved with the last row in Table 9, in which correct sentences are not used in training and the recall is higher than the row above it.

We study the impact of loss weights and the results are shown in Table 10. Because the three tasks are performed serially in our approach, we manually set three combinations of loss weights with ratios of 1:2:3, 1:1:1, and 3:2:1. We can see the automatic strategy with Eq. 6 does not perform well and the manually set weights can affect the final performance. Despite that the setting of 0.5, 1, 1.5 does not show a significant

Table 11 SED results on CoNLL-14

Method	Thres*	CoNLL-14 A1			CoNLL-14 A2			Merge
		P	*R*	Acc	*P*	*R*	Acc	Acc
BERT	0.79	80.2	94.9	**79.4**	93.3	92.6	87.9	**94.13**
SpanBERT	0.89	79.2	96.3	79.0	92.3	94.1	88.1	94.05
BERT	0.89	76.7	96.6	76.4	90.5	95.6	87.6	92.0
RoBERTa	0.89	79.4	95.7	79.0	92.6	93.5	87.9	92.9
SpanBERT	0.90	75.6	98.1	75.8	89.9	97.8	**88.6**	92.5
LSTM	0.73	72.7	**99.0**	72.4	86.6	**98.9**	85.9	89.3
BERT	0.50	**86.0**	79.7	76.0	**96.7**	75.1	76.4	77.3
LSTM	0.50	76.2	85.0	70.0	89.4	83.6	77.4	78.2

P and *R* represent the precision and recall of incorrect sentences

* Threshold is determined by grid search in development set, except the last two rows

difference from the setting of 1, 1, 1, we use the previous one in our final model because the result in Table 4 shows that our corrector needs improvement.

3.6 Empirical results on sentence error detection

We additionally study current pre-training models' performance on the task of SED because error detection is more important in communications. Once the message is detected with errors, we can withdraw it to avoid unnecessary operations with the wrong message. Besides, SED can be practical in improving the efficiency of GEC. If sentences are previously well distinguished with SED, there is no necessity to correct those correct sentences and thus this can reduce the work of error correction and save time.

We fine-tune pre-trained models and test BERT, RoBERTa[33], and SpanBERT's performance in CoNLL-2014 test set. We also build a baseline model with Bi-LSTM and 300d pre-trained word vectors[5] for comparison. The training data is all training sets in Table 2. Since the test set has two annotations, one sentence can be considered correct by one annotator and incorrect by another. Thus we not only evaluate the performance according to each annotation but also evaluate with the two merged annotations, which regards a prediction right if it matches any annotation.

The results are shown in Table 11, and all pre-training models outperform the baseline model in accuracy. By comparing the bottom two groups in Table 11, we can see applying grid search on the development set to determine the threshold can significantly increase the recall of incorrect sentences and can improve accuracy. However, the precision drops, which means many correct sentences are predicted with high correctness. The top group's results come from our multi-task learning models and the thresholds are precise to three decimals as the predicted logits in the development set are extremely low. This can indicate the multi-task learning approach does not provide a good correctness distribution for the development set, but grid search helps to find a boundary and makes the final results comparable to single models.' Despite that the merged accuracy is over 90%, the performance of all pre-trained models is not satisfying when considering the incorrect sentences' percentage in the test set.

[5] http://nlp.stanford.edu/data/glove.6B.zip

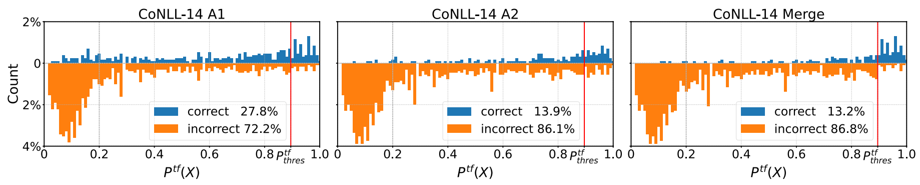

Fig. 2 A statistic of BERT's outputs for SED on CoNLL-14

We further visualize the output logits of the BERT model (third row in Table 11) in Fig. 2 to analyze the distribution of CoNLL-2014 test set. RoBERTa and SpanBERT show similar distributions as BERT. In theory, for binary classification, there should be a clear boundary that can unambiguously divide the correct sentences and incorrect ones. However, the boundary does not exist in Fig. 2. Despite the threshold P^{tf}_{thres} obtained by grid search can provide a fair boundary, it cannot improve the discriminant ability of the model. Besides, the distribution of incorrect sentences shows a long tail with a high probability of correctness, which indicates that there are a considerable number of sentences that cannot be well distinguished.

In a brief summary for the SED task, the threshold found by grid search can make the discriminator adaptive to the target dataset and improve performance, but current pre-trained language representation models still lack discriminant ability. Thus sentence error detection remains a research problem to be solved.

4 Related work

Many works have been taken to improve GEC since the CoNLL-2014 [16] and BEA-2019[17] share tasks were organized. The current main methods of handling GEC can be classified as machine translation and sequence labeling.

Machine translation methods are first widely applied for GEC because the task can be considered as translating an incorrect sentence to a correct one. To improve the performance of GEC, the structure of encoder and decoder in NMT has changed from RNN [34] to Transformer [7]. As a result, the model's parameter number increases a lot. Current well-performed NMT-based systems [8, 35] are mainly based on the Transformer (big) [9] architecture, which has nearly twice parameters than the base version of BERT. However, machine translation approaches are considered a low-resource problem [36] even if the current public training dataset has over one million examples. Thus there are many works on data augmentation for GEC. Ge et al. [37] proposed three strategies for data augmentation with fluency boost learning. Zhao et al. [7] generated pseudo data by directly deleting, inserting, replacing, and shuffling words in the correct sentence with a certain probability. Since current works on NMT-based systems are mainly focused on higher performance, efficiency becomes a problem to them, which can prevent their application in mobile communications.

Recently, local sequence transduction [11] becomes a breakthrough for GEC and it aims at only correcting the errors in the sentence instead of generating a new sentence. As local sequence transduction can be regarded as a sequence labeling problem [38], the calculation cost can be extremely reduced, thus improving the efficiency of GEC. LaserTagger [31] and PIE [11] were proposed for GEC contemporaneously, and they are both based on SL method. LaserTagger tries to assign an operation to each token and each operation is a basic tag (keep or delete) combined with a phrase. Thus correction can be made by deleting and adding a phrase to the source sentence according to each token's operation. PIE shares a similar idea with LaserTagger, but it uses copy, delete, append, and replace as the basic tag for each token and it additionally involves format transformations for certain grammatical errors. GECToR [12] integrates advantages from LaserTagger and PIE and achieves current state-of-the-art performance. GECToR builds a more comprehensive vocabulary of format transformations and limits the number of tokens in each label to one. Iterative correction and ensemble learning are applied by GECToR to further improve performance. Despite these methods are more efficient than NMT, they can only correct limited kinds of errors and they all do not consider the sentence-level error.

Most recently, Chen et al. [32] proposed a span-level error detection and correction model to improve the efficiency of NMT-based GEC models, which involved one SL model for token-level error detection and one NMT model for correction. We share the same idea of correcting errors based on the detected errors' positions, but they use two separate models to achieve this, which means they need to encode one sentence twice. Moreover, they do not consider the correctness of sentences, and the correction generation stage is still auto-regressive. Thus their approach is less efficient than ours.

Researches on GEC are not only limited to English, and Chinese Grammatical Error Diagnosis (CGED) [39] is one similar task for Chinese. CGED is focused on detecting and identifying errors in Chinese corpus, and it also includes a sub-task of generating corrections, but the current performance on correction is far from satisfying. CGED additionally defines word ordering errors compared with our three types and has four types of errors. Since the word ordering error can be more flexible than other types (e.g., swapping two neighboring tokens or moving one phrase from back to front), we consider it as a redundancy error and a missing error.

SED and GED are two auxiliary tasks of GEC and aim at detecting sentence-level errors and token-level errors. SED is less concerned in recent years may because it is just a binary classification problem, but SED is not solved perfectly as shown in our experiment. To the best of our knowledge, only Asano et al. [40] involved SED for GEC in recent years. They build a proficiency prediction model to enhance the performance of SED, but it requires the category information of writers. GED is currently regarded as a sequence labeling problem and many works follow the label definition of Rei and Yannakoudakis [20], which classifies each token into a correct or incorrect one and label incorrect to the token after the missing error's gap. However, the incorrect label cannot tell if the token is incorrect or a missing error occurs before the token. Meanwhile, our approach solves the problem with BIO tags and error types. GECToR [12] also introduces a GED task and it defines a token as incorrect if the label assigned for correction is not 'keep,' which is reasonable to their approach. Features of each token are quite important to the performance of GED and Kaneko et al. [41] prove that the quality of word embeddings can affect results. Bell et al. [21] involve contextual embeddings from BERT to a Bi-LSTM model for GED and prove contextual embeddings can help to improve the performance. Despite we combine these two tasks in our approach to improve efficiency, the power of error detection is not fully explored.

5 Conclusion

To efficiently detect and correct noisy messages in mobile communications, this paper investigates GEC and proposes a novel hierarchical approach by checking both sentence-level and token-level errors and generating corrections for detected errors. Error detections and correction are integrated into one multi-task learning model, each module is designed for a specific task and is efficient, and thus our approach shows an outstanding inference speed with general performance in GEC. We additionally analyze current pre-training models' performance in SED and find they cannot provide a clear boundary to correct and incorrect sentences.

Abbreviations

GEC	Grammatical error detection
SED	Sentence error detection
GED	Grammatical error detection
NMT	Neural machine translation
SL	Sequence labeling

Acknowledgements
This research was partially supported by National Natural Science Foundation of China (No. 62276233) and Key Research Project of Zhejiang Province (2022C01145).

Author Contributions
FP and BC conceived and designed the study. JF suggested the extra study in error detection. FP performed the experiments. FP and BC wrote the paper. BC and JF reviewed and edited the manuscript. All authors read and approved the final manuscript.

Availability of data and materials
The datasets used in this paper are available in https://www.cl.cam.ac.uk/research/nl/bea2019st/#data. Our code and trained model are published at https://github.com/AnticPan/Hierarchical-GEC/tree/v2.

Declarations

Competing interests
The authors declare that there is no conflict of interest.

Received: 25 May 2022 Accepted: 21 September 2022
Published online: 01 October 2022

References

1. H. Gao, C. Liu, Y. Yin, Y. Xu, Y. Li, A hybrid approach to trust node assessment and management for vanets cooperative data communication: Historical interaction perspective. IEEE Trans. Intell. Transp. Syst. (2021). https://doi.org/10.1109/TITS.2021.3129458
2. H. Gao, W. Huang, T. Liu, Y. Yin, Y. Li, Ppo2: location privacy-oriented task offloading to edge computing using reinforcement learning for intelligent autonomous transport systems. IEEE Trans. Intell. Transp. Syst. (2022). https://doi.org/10.1109/TITS.2022.3169421
3. H. Gao, B. Qiu, R.J. Duran Barroso, W. Hussain, Y. Xu, X. Wang, Tsmae: a novel anomaly detection approach for internet of things time series data using memory-augmented autoencoder. IEEE Trans. Netw. Sci. Eng. (2020). https://doi.org/10.1109/TNSE.2022.3163144
4. Y. Yin, Q. Huang, H. Gao, Y. Xu, Personalized apis recommendation with cognitive knowledge mining for industrial systems. IEEE Trans. Ind. Inform. **17**(9), 6153–6161 (2021). https://doi.org/10.1109/TII.2020.3039500
5. Y. Yin, Z. Cao, Y. Xu, H. Gao, R. Li, Z. Mai, Qos prediction for service recommendation with features learning in mobile edge computing environment. IEEE Trans. Cognit. Commun. Network. **6**(4), 1136–1145 (2020). https://doi.org/10.1109/TCCN.2020.3027681
6. S. Rothe, J. Mallinson, E. Malmi, S. Krause, A. Severyn, A simple recipe for multilingual grammatical error correction, in *Proceedings of the 59th annual meeting of the association for computational linguistics and the 11th international joint conference on natural language processing*, vol. 2: Short Papers, (Association for Computational Linguistics, 2021), pp. 702–707. https://doi.org/10.18653/v1/2021.acl-short.89. https://aclanthology.org/2021.acl-short.89
7. W. Zhao, L. Wang, K. Shen, R. Jia, J. Liu, Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data, in *Proceedings of the 2019 conference of the North*, (Association for Computational Linguistics, Minneapolis, Minnesota, 2019), pp. 156–165. https://doi.org/10.18653/v1/N19-1014

8. M. Kaneko, M. Mita, S. Kiyono, J. Suzuki, K. Inui, Encoder-decoder models can benefit from pre-trained masked language models in grammatical error correction, in *Proceedings of the 58th annual meeting of the association for computational linguistics*, (2020), pp. 4248–4254. https://doi.org/10.18653/v1/2020.acl-main.391
9. A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, u. Kaiser, I. Polosukhin, Attention is all you need, in *Proceedings of the 31st international conference on neural information processing systems. NIPS'17*, (Curran Associates Inc., Red Hook, NY, USA, 2017), pp. 6000–6010
10. Y. Park, 5g vision and requirements of 5g forum, korea, in *ITU-R WP5D Workshop* (2014)
11. A. Awasthi, S. Sarawagi, R. Goyal, S. Ghosh, V. Piratla, Parallel iterative edit models for local sequence transduction. in *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, (Association for Computational Linguistics, Hong Kong, China, 2019), pp. 4260–4270. https://doi.org/10.18653/v1/D19-1435
12. K. Omelianchuk, V. Atrasevych, A. Chernodub, O. Skurzhanskyi, GECToR – grammatical error correction: tag, not rewrite. in *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for building educational applications*, (Association for Computational Linguistics, Seattle, WA, USA →, 2020), pp. 163–170. https://doi.org/10.18653/v1/2020.bea-1.16
13. Tsvetkov, Y., Dyer, C.: Lexicon stratification for translating out-of-vocabulary words, in *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing, vol. 2: Short Papers*. (Association for Computational Linguistics, Beijing, China, 2015), pp. 125–131. https://doi.org/10.3115/v1/P15-2021. https://aclanthology.org/P15-2021
14. M. Joshi, D. Chen, Y. Liu, D.S. Weld, L. Zettlemoyer, O. Levy, SpanBERT: improving pre-training by representing and predicting spans. Trans. Assoc. Comput. Linguist. **8**, 64–77 (2020)
15. F. Pan, B. Cao, Efficient grammatical error correction with hierarchical error detections and correction, in *2021 IEEE international conference on web services (ICWS)* (2021), pp. 525–530. https://doi.org/10.1109/ICWS53863.2021.00073
16. H.T. Ng, S.M. Wu, T. Briscoe, C. Hadiwinoto, R.H. Susanto, C. Bryant, The CoNLL-2014 shared task on grammatical error correction, in *Proceedings of the eighteenth conference on computational natural language learning: shared task*, (Association for Computational Linguistics, Baltimore, Maryland, 2014), pp. 1–14. https://doi.org/10.3115/v1/W14-1701
17. C. Bryant, M. Felice, E. Andersen, Ø, T. Briscoe, The BEA-2019 shared task on grammatical error correction, in *Proceedings of the fourteenth workshop on innovative use of NLP for building educational applications*, (Association for Computational Linguistics, Florence, Italy, 2019), pp. 52–75. https://doi.org/10.18653/v1/W19-4406
18. C. Bryant, M. Felice, T. Briscoe, Automatic annotation and evaluation of error types for grammatical error correction, in *Proceedings of the 55th annual meeting of the association for computational linguistics, vol. 1: Long Papers*, (Association for Computational Linguistics, Vancouver, Canada, 2017), pp. 793–805. https://doi.org/10.18653/v1/P17-1074. https://aclanthology.org/P17-1074
19. L. Ramshaw, M. Marcus, Text chunking using transformation-based learning, in *Third Workshop on Very Large Corpora* (1995). https://aclanthology.org/W95-0107
20. M. Rei, H. Yannakoudakis, Compositional sequence labeling models for error detection in learner writing, in *Proceedings of the 54th annual meeting of the association for computational linguistics, vol. 1: Long Papers*, (Association for Computational Linguistics, Berlin, Germany, 2016), pp. 1181–1191. https://doi.org/10.18653/v1/P16-1112
21. S. Bell, H. Yannakoudakis, M. Rei, Context is key: grammatical error detection with contextual word representations, in *Proceedings of the fourteenth workshop on innovative use of NLP for building educational applications*, (Association for Computational Linguistics, Florence, Italy, 2019), pp. 103–115. https://doi.org/10.18653/v1/W19-4410
22. J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, BERT: pre-training of deep bidirectional transformers for language understanding, in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, vol. 1 Long and Short Papers*, (Association for Computational Linguistics, Minneapolis, Minnesota, 2019), pp. 4171–4186. https://doi.org/10.18653/v1/N19-1423
23. C. Sutton, A. McCallum, An introduction to conditional random fields. arXiv (2010). https://doi.org/10.48550/ARXIV.1011.4088. https://arxiv.org/abs/1011.4088
24. H.T. Ng, S.M. Wu, Y. Wu, C. Hadiwinoto, J. Tetreault, The CoNLL-2013 shared task on grammatical error correction, in *Proceedings of the seventeenth conference on computational natural language learning: shared task*, (Association for Computational Linguistics, Sofia, Bulgaria, 2013), pp. 1–12
25. H. Yannakoudakis, T. Briscoe, B. Medlock, A new dataset and method for automatically grading ESOL texts, in *Proceedings of the 49th annual meeting of the association for computational linguistics: human language technologies*, (Association for Computational Linguistics, Portland, Oregon, USA, 2011), pp. 1–12
26. D. Dahlmeier, H.T. Ng, S.M. Wu, Building a large annotated corpus of learner English: the NUS corpus of learner English, in *Proceedings of the eighth workshop on innovative use of NLP for building educational applications*, (Association for Computational Linguistics, Atlanta, Georgia, 2013), pp. 22–31
27. T. Mizumoto, M. Komachi, M. Nagata, Y. Matsumoto, Mining revision log of language learning SNS for automated Japanese error correction of second language learners, in *Proceedings of 5th international joint conference on natural language processing*, (Asian Federation of Natural Language Processing, Chiang Mai, Thailand, 2011), pp. 147–155
28. D. Dahlmeier, H.T. Ng, Better evaluation for grammatical error correction, in *Proceedings of the 2012 conference of the North American chapter of the association for computational linguistics: human language technologies*, (Association for Computational Linguistics, Montréal, Canada, 2012), pp. 568–572
29. Y. Wu, M. Schuster, Z. Chen, Q.V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, S. Łukasz Kaiser, Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C.Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, J. Dean, Google's neural machine translation system: bridging the gap between human and machine translation (2016). arXiv:1609.08144
30. I. Loshchilov, F. Hutter, Decoupled weight decay regularization (2019). arXiv:1711.05101
31. E. Malmi, S. Krause, S. Rothe, D. Mirylenka, A. Severyn, Encode, tag, realize: high-precision text editing, in *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, (Association for Computational Linguistics, Hong Kong, China, 2019), pp. 5054–5065. https://doi.org/10.18653/v1/D19-1510

32. M. Chen, T. Ge, X. Zhang, F. Wei, M. Zhou, 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)* (2020), pp. 7162–7169. https://doi.org/10.18653/v1/2020.emnlp-main.581
33. Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, RoBERTa: a robustly optimized BERT pretraining approach (2019). arXiv:1907.11692
34. Z. Yuan, T. Briscoe, Grammatical error correction using neural machine translation, in *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, (Association for Computational Linguistics, San Diego, California, 2016), pp. 380–386. https://doi.org/10.18653/v1/N16-1042
35. R. Grundkiewicz, M. Junczys-Dowmunt, K. Heafield, Neural grammatical error correction systems with unsupervised pre-training on synthetic data, in *Proceedings of the fourteenth workshop on innovative use of NLP for building educational applications*, (Association for Computational Linguistics, Florence, Italy, 2019), pp. 252–263. https://doi.org/10.18653/v1/W19-4427
36. M. Junczys-Dowmunt, R. Grundkiewicz, S. Guha, K. Heafield, Approaching neural grammatical error correction as a low-resource machine translation task, in *Proceedings of the 2018 conference of the North American chapter of the association for computational linguistics: human language technologies, vol. 1 Long Papers*, (Association for Computational Linguistics, New Orleans, Louisiana, 2018), pp. 595–606. https://doi.org/10.18653/v1/N18-1055
37. T. Ge, F. Wei, M. Zhou, Fluency boost learning and inference for neural grammatical error correction, in *Proceedings of the 56th annual meeting of the association for computational linguistics, vol. 1: Long Papers*, (Association for Computational Linguistics, Melbourne, Australia, 2018), pp. 1055–1065. https://doi.org/10.18653/v1/P18-1097
38. J. Ribeiro, S. Narayan, S.B. Cohen, X. Carreras, Local string transduction as sequence labeling, in *Proceedings of the 27th international conference on computational linguistics*, (Association for Computational Linguistics, Santa Fe, New Mexico, USA, 2018), pp. 1360–1371
39. G. Rao, E. Yang, B. Zhang, Overview of NLPTEA-2020 shared task for chinese grammatical error diagnosis, in *Proceedings of the 6th workshop on natural language processing techniques for educational applications*, (Association for Computational Linguistics, Suzhou, China, 2020), pp. 25–35
40. H. Asano, M. Mita, T. Mizumoto, J. Suzuki, The AIP-tohoku system at the BEA-2019 shared task, in *Proceedings of the fourteenth workshop on innovative use of NLP for building educational applications*, (Association for Computational Linguistics, Florence, Italy, 2019), pp. 176–182. https://doi.org/10.18653/v1/W19-4418
41. M. Kaneko, Y. Sakaizawa, M. Komachi, Grammatical error detection using error- and grammaticality-specific word embeddings, in *Proceedings of the eighth international joint conference on natural language processing, vol. 1: Long Papers*, (Asian Federation of Natural Language Processing, Taipei, Taiwan, 2017), pp. 40–48

Publisher's Note