Neural and FST-based approaches to grammatical error correction

Zheng Yuan♦, Felix Stahlberg♦, Marek Rei♦, Bill Byrne♦, Helen Yannakoudakis♦

♦Department of Computer Science & Technology, University of Cambridge, United Kingdom
♦ALTA Institute, University of Cambridge, United Kingdom

{zheng.yuan, marek.rei, helen.yannakoudakis}@cl.cam.ac.uk,
{fs439, bill.byrne}@eng.cam.ac.uk

Abstract

In this paper, we describe our submission to the BEA 2019 shared task on grammatical error correction. We present a system pipeline that utilises both error detection and correction models. The input text is first corrected by two complementary neural machine translation systems: one using convolutional networks and multi-task learning, and another using a neural Transformer-based system. Training is performed on publicly available data, along with artificial examples generated through back-translation. The \( n \)-best lists of these two machine translation systems are then combined and scored using a finite state transducer (FST). Finally, an unsupervised re-ranking system is applied to the \( n \)-best output of the FST. The re-ranker uses a number of error detection features to re-rank the FST \( n \)-best list and identify the final 1-best correction hypothesis. Our system achieves 66.75% \( F_{0.5} \) on error correction (ranking 4th), and 82.52% \( F_{0.5} \) on token-level error detection (ranking 2nd) in the restricted track of the shared task.

1 Introduction

Grammatical error correction (GEC) is the task of automatically correcting grammatical errors in written text. In this paper, we describe our submission to the restricted track of the BEA 2019 shared task on grammatical error correction (Bryant et al., 2019), where participating teams are constrained to using only the provided datasets as training data. Systems are expected to correct errors of all types, including grammatical, lexical and orthographical errors. Compared to previous shared tasks on GEC, which have primarily focused on correcting errors committed by non-native speakers (Dale and Kilgarriff, 2011; Dale et al., 2012; Ng et al., 2013, 2014), a new annotated dataset is introduced, consisting of essays produced by native and non-native English language learners, with a wide coverage of language proficiency levels for the latter, ranging from elementary to advanced.

Neural machine translation (NMT) systems for GEC have drawn growing attention in recent years (Yuan and Briscoe, 2016; Xie et al., 2016; Ji et al., 2017; Sakaguchi et al., 2017; Chollampatt and Ng, 2018; Junczys-Dowmunt et al., 2018), as they have been shown to achieve state-of-the-art results (Ge et al., 2018; Zhao et al., 2019). Within this framework, error correction is cast as a monolingual translation task, where the source is a sentence (written by a language learner) that may contain errors, and the target is its corrected counterpart in the same language.

Due to the fundamental differences between a “true” machine translation task and the error correction task, previous work has investigated the adaptation of NMT for the task of GEC. Byte pair encoding (BPE) (Chollampatt and Ng, 2018; Junczys-Dowmunt et al., 2018) and a copying mechanism (Zhao et al., 2019) have been introduced to deal with the “noisy” input text in GEC and the non-standard language used by learners. Some researchers have investigated ways of incorporating task-specific knowledge, either by directly modifying the training objectives (Schmaltz et al., 2017; Sakaguchi et al., 2017; Junczys-Dowmunt et al., 2018) or by re-ranking machine-translation-system correction hypotheses (Yannakoudakis et al., 2017; Chollampatt and Ng, 2018). To ameliorate the lack of large amounts of error-annotated learner data, various approaches have proposed to leverage unlabelled native data within a number of frameworks, including artificial error generation with back translation (Rei et al., 2017; Kasewa et al., 2018), fluency boost learning (Ge et al., 2018), and pre-training with denoising autoencoders (Zhao et al., 2019).
targeting all errors may not necessarily be the best approach to the task, and that different GEC systems may be better suited to correcting different types of errors, and can therefore be complementary (Yuan, 2017). As such, hybrid systems that combine different approaches have been shown to yield improved performance (Felice et al., 2014; Rozovskaya and Roth, 2016; Grundkiewicz and Junczys-Dowmunt, 2018). In line with this work, we present a hybrid approach that 1) employs two NMT-based error correction systems: a neural convolutional system and a neural Transformer-based system; 2) a finite state transducer (FST) that combines and further enriches the n-best outputs of the NMT systems; 3) a re-ranking system that re-ranks the n-best output of the FST based on error detection features.

The remainder of this paper is organised as follows: Section 2 describes our approach to the task; Section 3 describes the datasets used and presents our results on the shared task development set; Section 4 presents our official results on the shared task test set, including a detailed analysis of the performance of our final system; and, finally, Section 5 concludes the paper and provides an overview of our findings.

2 Approach

We approach the error correction task using a pipeline of systems, as presented in Figure 1. In the following sections, we describe each of these components in detail.

2.1 The convolutional neural network (CNN) system

We use a neural sequence-to-sequence model and an encoder–decoder architecture (Cho et al., 2014; Sutskever et al., 2014). An encoder first reads and encodes an entire input sequence \( x = (x_1, x_2, ..., x_n) \) into hidden state representations. A decoder then generates an output sequence \( y = (y_1, y_2, ..., y_m) \) by predicting the next token \( y_t \) based on the input sequence \( x \) and all the previously generated tokens \( \{y_1, y_2, ..., y_{t-1}\} \):

\[
p(y) = \prod_{t=1}^{m} p(y_t|\{y_1, ..., y_{t-1}\}, x) \tag{1}
\]

Our convolutional neural system is based on a multi-layer convolutional encoder–decoder model (Gehring et al., 2017), which employs convolutional neural networks (CNNs) to compute intermediate encoder and decoder states. The parameter settings follow Chollampatt and Ng (2018) and Ge et al. (2018). The source and target word embeddings have size 500, and are initialised with fastText embeddings (Bojanowski et al., 2017) trained on the native English Wikipedia corpus (2, 405, 972, 890 tokens). Each of the encoder and decoder is made up of seven convolutional layers, with a convolution window width of 3. We apply a left-to-right beam search to find a correction that approximately maximises the conditional probability in Equation 1.

**BPE** is introduced to alleviate the rare-word problem, and rare and unknown words are split into multiple frequent subword tokens (Sennrich et al., 2016b). NMT systems often limit vocabulary size on both source and target sides due to the computational complexity during training. Therefore, they are unable to translate out-of-vocabulary (OOV) words, which are treated as unknown tokens, resulting in poor translation quality. As noted by Yuan and Briscoe (2016), this problem is more serious for GEC as non-native text contains, not only rare words (e.g., proper nouns), but also misspelled words (i.e., spelling errors).

In our model, each of the source and target vocabularies consist of the 30K most frequent BPE tokens from the source and target side of the parallel training data respectively. The same BPE operation is applied to the Wikipedia data before being used for training of our word embeddings.

**Copying mechanism** is a technique that has led to performance improvement on various monolingual sequence-to-sequence tasks, such as text summarisation, dialogue systems, and paraphrase generation (Gu et al., 2016; Cao et al., 2017). The idea is to allow the decoder to choose between simply copying an original input word and outputting a translation word. Since the source and target sentences are both in the same language (i.e., monolingual translation) and most words in
the source sentence are correct and do not need to change, GEC seems to benefit from the copying mechanism.

Following the work of Gu et al. (2016), we use a dynamic target vocabulary, which contains a fixed vocabulary learned from the target side of the training data plus all the unique tokens introduced by the source sentence. As a result, the probability of generating any target token $p(y_t\{y_1, ..., y_{t-1}\}, x)$ in Equation 1 is defined as a “mixture” of the generation probability $p(y_t, g\{y_1, ..., y_{t-1}\}, x)$ and the copy probability $p(y_t, c\{y_1, ..., y_{t-1}\}, x)$:

$$p(y_t\{y_1, ..., y_{t-1}\}, x) = p(y_t, g\{y_1, ..., y_{t-1}\}, x) + p(y_t, c\{y_1, ..., y_{t-1}\}, x)$$ (2)

**Multi-task learning** has found success in a wide range of tasks, from natural language processing (NLP) (Collobert and Weston, 2008) and speech recognition (Deng et al., 2013) to computer vision (Girshick, 2015). Multi-task learning allows systems to use information from related tasks and learn from multiple objectives, which leads to performance improvement on individual tasks. Recently, Rei (2017) and Rei and Yannakoudakis (2017) investigated the use of different auxiliary objectives for the task of error detection in learner writing.

In addition to our primary error correction task, we propose two related auxiliary objectives to boost model performance:

- **Token-level labelling**

  We jointly train an error detection and error correction system by providing error detection labels. Instead of only generating a corrected sentence, we extend the system to additionally predict whether a token in the source sentence is correct or incorrect.

- **Sentence-level labelling**

  A binary classification task is also introduced to predict whether the original source sentence is grammatically correct or incorrect. We investigate the usefulness of sentence-level classification as an auxiliary objective for training error correction models.

Labels for both auxiliary error detection tasks are generated automatically by comparing source and target tokens using the ERRANT automatic alignment tool (Bryant et al., 2017). We first align each token $x_i$ in the source sentence $x$ with a token $y_j$ in the target sentence $y$. If $x_i = y_j$, the source token $x_i$ is correct; while if $x_i \neq y_j$, the source token $x_i$ is incorrect. Similarly, the source sentence $x$ is correct if $x = y$, and incorrect otherwise.

**Artificial error generation** is the process of injecting artificial errors into a set of error-free sentences. Compared to standard machine translation tasks, GEC suffers from the limited availability of large amounts of training data. As manual error annotation of learner data is a slow and expensive process, artificial error generation has been applied to error correction (Felice and Yuan, 2014) and detection (Rei et al., 2017) with some success. Following the work of Rei et al. (2017), we treat error generation as a machine translation task, where a grammatically correct sentence is translated to an incorrect counterpart. We built an error generation system using the same network architecture as the one described here, with error-corrected sentences as the source and their corresponding uncorrected counterparts written by language learners as the target. The system is then used to collect the $n$-best outputs: $y_1^o, y_2^o, ..., y_n^o$, for a given error-free native and/or learner sentence $y$. Since there is no guarantee that the error generation system will inject errors into the input sentence $y$ to make it less grammatically correct, we apply “quality control”. A pair of artificially generated sentences $(y_k^o, y)$, for $k \in \{1, 2, ..., n\}$, will be added to the training set of the error correction system if the following condition is met:

$$\frac{f(y)}{f(y_k^o)} \leq \sigma$$ (3)

where $f(y)$ is the normalised log probability of $y$:

$$f(y) = \frac{\sum_{i=1}^m \log(P(y_i|y_{<i}))}{m}$$ (4)

This ensures that the quality of the artificially generated sentence, as estimated by a language model, is lower compared to the original sentence. We use a 5-gram language model (LM) trained on the One Billion Word Benchmark dataset (Chelba et al., 2014) with KenLM (Heafield, 2011) to compute $P(y_i|y_{<i})$.

The $\sigma$ in Equation 3 is a threshold used to filter out sentence pairs with unnecessary changes; e.g., [I look forward to hearing from you. → I am looking forward to hearing from you.]. It is an av-
eraged score learned on the development set:

\[
\sigma = \frac{\sum_{i=1}^{N} f(y_i)}{N}
\]

(5)

where \((x, y)\) is a pair of parallel sentences in the development set, and \(N\) is the total number of pairs.

### 2.2 The neural Transformer-based system

Besides the convolutional system from the previous section, we also use the purely neural Transformer-based system of Stahlberg and Byrne (2019). They use an ensemble of four Transformer (Vaswani et al., 2017) NMT and two Transformer LM models in Tensor2Tensor (Vaswani et al., 2018) transformer big configuration. The NMT models are trained with back-translation (Sennrich et al., 2016a; Rei et al., 2017; Kasewa et al., 2018) and fine-tuning through continued training. For a detailed description of this system we refer the reader to Stahlberg and Byrne (2019).

### 2.3 FST-based system combination

Stahlberg et al. (2019) demonstrated the usefulness of FSTs for grammatical error correction. Their method starts with an input lattice \(I\) which is generated with a phrase-based statistical machine translation (SMT) system. The lattice \(I\) is composed with a number of FSTs that aim to enrich the search space with further possible corrections. Similarly to Bryant and Briscoe (2018), they rely on external knowledge sources like spell checkers and morphological databases to generate additional correction options for the input sentence. The enriched lattice is then mapped to the sub-word level by composition with a mapping transducer, and re-scored with neural machine translation models and neural LMs.

In this work, rather than combining SMT and neural models, we use the framework of Stahlberg et al. (2019) to combine and enrich the outputs of two neural systems. The input lattice \(I\) is now the union of two \(n\)-best lists – one from the convolutional system (Section 2.1), and one from the Transformer-based system (Section 2.2). After composition, we re-score the enriched input lattice \(I\) with the system described in Section 2.2. The FST-based system combination uses 7 different features: the convolutional system score, the LM and NMT scores from the Transformer-based system, the edit distance of hypotheses in \(I\) to the input sentence, substitution and deletion penalties for the additional correction options from the FST framework, and the word count. Following Stahlberg et al. (2019); Stahlberg and Byrne (2019), we scale these features and tune the scaling weights on the BEA-2019 development set using a variant of Powell search (Powell, 1964). We use OpenFST (Allauzen et al., 2007) as backend for FST operations, and the SGNMT decoder (Stahlberg et al., 2017, 2018) for neural decoding under FST constraints.

### 2.4 Re-ranking FST output

Yannakoudakis et al. (2017) found that grammatical error detection systems can be used to improve error correction outputs. Specifically, they re-rank the \(n\)-best correction hypotheses of an SMT system based on error detection predictions. Following this work, we also deploy a re-ranking component which re-ranks the \(n\)-best correction hypotheses of the FST system (Section 2.3) based on error detection predictions output by an error detection system.

#### Error detection.

Our system for grammatical error detection is based on the model described by Rei (2017).\(^1\) The task is formulated as a sequence labeling problem – given a sentence, the model assigns a probability to each token, indicating the likelihood of that token being incorrect in the given context (Rei and Yannakoudakis, 2016). The architecture maps words to distributed embeddings, while also constructing character-based representations for each word with a neural component. These are then passed through a bidirectional LSTM, followed by a feed-forward layer and a softmax layer at the output.

In addition to neural text representations, we also include several external features into the model, designed to help it learn more accurate error detection patterns from the limited amounts of training data available:

- Two binary features indicating whether two publicly available spell-checkers – HunSpell\(^2\) and JamSpell\(^3\) – identify the target word as a spelling mistake.

\(^1\)https://github.com/marekrei/sequence-labeler
\(^2\)http://hunspell.github.io/
\(^3\)https://github.com/bakwc/JamSpell
• The POS tag, NER label and dependency relation of the target word based on the Stanford parser (Chen and Manning, 2014).

• The number of times the unigram, bigram, or trigram context of the target word appears in the BNC (Burnard, 2007) and in ukWaC (Ferraresi et al., 2008).

• Contextualized word representations from ELMo (Peters et al., 2018).

The discrete features are represented as 10-dimensional embeddings and, together with the continuous features, concatenated to each word representation in the model. The overall architecture is optimized for error detection using cross-entropy. Once trained, the model returns the predicted probabilities of each token in a sentence being correct or incorrect.

Re-ranker. We generate the list of the 8 best correction hypotheses from our FST system, and then use the following set of error detection-based features to assign a new score to each hypothesis and determine a new ranking:

1. Sentence correctness probability: the error detection model outputs a probability indicating whether a token is likely to be correct or incorrect in context. We therefore use as a feature the overall FST sentence probability, calculated based on the probability of each of its tokens being correct: \( \sum_w \log P(w) \)

2. Levenshtein distance (LD): we first use LD to identify 1) which tokens in the original/uncorrected sentence have been corrected by the FST candidate hypothesis, and 2) which tokens in the original/uncorrected sentence our detection model predicts as incorrect (i.e., the probability of being incorrect is > 0.5). We then convert these annotations to binary sequences — i.e., 1 if the token is identified as incorrect, and 0 otherwise — and use as a feature the LD between those binary representations. Specifically, we would like to select the candidate FST sentence that has the smallest LD from the binary sequence created by the detection model, and therefore use as a feature the following: \( \frac{1.0}{LD + 1} \)

3. False positives: using the binary sequences described above, we count the number of false positives (FP) on token-level error detection by treating the error detection model as the “gold standard”. Specifically, we count how many times the candidate FST hypothesis disagrees with the detection model on the tokens identified as incorrect, and use as a feature the following: \( \frac{1.0}{FP + 1.0} \)

We use a linear combination of the above three features together with the original score given by the FST system for each candidate hypothesis to re-rank the FST system’s 8-best list in an unsupervised way. The new 1-best correction hypothesis \( c^* \) is then the one that maximises:

\[
c^* = \arg \max_c \sum_{i=1}^{K} \lambda_i h_i(c)
\]  

where \( h \) represents the score assigned to candidate hypothesis \( c \) according to feature \( i \); \( \lambda \) is a weighting parameter that controls the effect feature \( i \) has on the final ranking; and \( K = 4 \) as we use a total of four different features (three features based on the detection model, and one which is the original score output by the FST system). \( \lambda \)s are tuned on the development set and are set to \( \lambda = 2.0 \) for features 1. and 2., \( \lambda = 3.0 \) for feature 3. and \( \lambda = 1.5 \) for the original FST score.

3 Experiments and results

3.1 Datasets and evaluation

In the restricted track, participating teams were constrained to use only the provided learner datasets:4

• Cambridge English W&I corpus

Cambridge English Write & Improve (W&I)5 (Yannakoudakis et al., 2018) is an online web platform that assists non-native English learners with their writing. Learners from around the world submit letters, stories, articles and essays for automated assessment in response to various prompts. The W&I corpus (Bryant et al., 2019) contains 3,600 annotated submissions across 3 different CEFR6 levels: A (beginner), B (intermediate), and C (advanced). The data has been

---

4We note that there are no restrictions on the use of NLP tools (e.g., POS taggers, parsers, spellcheckers, etc.), nor on the amount of unannotated data that can be used, so long as such resources are publicly available.

5https://writeandimprove.com/

6https://www.cambridgeenglish.org/exams-and-tests/cefr/
split into training (3,000 essays), development (200 essays), and test (200 essays) partitions.

- **LOCNESS**
  The LOCNESS\(^7\) corpus (Granger, 1998) consists of essays written by native English students. A subsection of 100 essays has been manually annotated, and equally partitioned into development and test sets.

- **FCE**
  The First Certificate in English (FCE) corpus (Yannakoudakis et al., 2011) is a subset of the Cambridge Learner Corpus (CLC) that consists of 1,244 exam scripts written by learners of English sitting the FCE exam.

- **NUCLE**
  The National University of Singapore Corpus of Learner English (NUCLE) (Dahlmeier et al., 2013) contains 1,400 essays written by undergraduate students at the National University of Singapore who are non-native English speakers.

- **Lang-8 Corpus of Learner English**
  Lang-8\(^8\) is an online language learning website which encourages users to correct each other’s grammar. The Lang-8 Corpus of Learner English (Mizumoto et al., 2011; Tajiri et al., 2012) refers to an English subsection of this website (can be quite noisy).

Additional resources used in our system include:

- **English Wikipedia corpus**
  The English Wikipedia corpus (2,405,972,890 tokens in 110,698,467 sentences) is used to pre-train word embeddings for the convolutional neural system. We also use it as error-free native data for artificial error generation (see Section 2.1).

- **One Billion Word Benchmark dataset**
  A LM is trained on the One Billion Word Benchmark dataset, which consists of close to a billion words of English taken from news articles on the web, to evaluate the quality of artificially generated sentence pairs. A filtered version (768,646,526 tokens in 30,301,028 sentences) is used as input to the error generation model in Section 2.1.

   In order to cover the full range of English levels and abilities, the official development set consists of 300 essays from W&I (A: 130, B: 100, and C: 70) and 50 essays from LOCNESS (86,973 tokens in 4,384 sentences).

   The ERRANT scorer (Bryant et al., 2017) is used as the official scorer for the shared task. System performance is evaluated in terms of span-level correction using $F_{0.5}$, which emphasises precision twice as much as recall.

3.2 Training details

   The convolutional NMT model is trained with a hidden layer size of 1,024 for both the encoder and the decoder. Dropout at a rate of 0.2 is applied to the embedding layers, convolutional layers and decoder output. The model is optimized using Nesterov’s Accelerated Gradient Descent (NAG) with a simplified formulation for Nesterov’s momentum (Bengio et al., 2013). The initial learning rate is set to 0.25, with a decaying factor of 0.1 and a momentum value of 0.99. We perform validation after every epoch, and select the best model based on the performance on the development set. During beam search, we keep a beam size of 12 and discard all other hypotheses.

   The grammatical error detection system was optimized separately as a sequence labeling model. Word embeddings were set to size 300 and initialized with pre-trained Glove embedding (Pennington et al., 2014). The bi-LSTM has 300-dimensional hidden layers for each direction. Dropout was applied to word embeddings and LSTM outputs with probability 0.5. The model was optimized with Adam (Kingma and Ba, 2015), using a default learning rate 0.001. Training was stopped when performance on the development set did not improve over 7 epochs.

3.3 Individual system performance

   Individual system performance on the development set is reported in Table 1, where ‘CNN’ refers to the convolutional neural system, and ‘Transformer’ refers to the Transformer-based neural system. These results are based on the 1-best output from each system, although the $n$-best

\(^7\)https://uclouvain.be/en/research-institutes/ilc/cecl/locness.html

\(^8\)https://lang-8.com/
<table>
<thead>
<tr>
<th>System</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>P</th>
<th>R</th>
<th>F0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CNN</td>
<td>1697</td>
<td>2455</td>
<td>20</td>
<td>41.72</td>
<td>22.46</td>
<td>35.61</td>
</tr>
<tr>
<td>Transformer</td>
<td>234</td>
<td>2162</td>
<td>3506</td>
<td>53.17</td>
<td>32.90</td>
<td>47.34</td>
</tr>
</tbody>
</table>

Table 1: Span-level correction results for individual systems on the development set. TP: true positives, FP: false positives, FN: false negatives, P: precision, R: recall.

lists are used later for system combination.

3.4 Pipelines

Since corrections made by the convolutional neural system and the Transformer-based system are often complementary, and re-scoring has been proven to be useful and effective for error correction, we investigated ways to combine corrections generated by both systems. Table 2 shows results for different combinations, where ‘CNN’ refers to the convolutional neural system, ‘Transformer’ refers to the Transformer-based system, subscript ‘10-best’ indicates the use of the 10-best list of correction candidates from the system, ‘+’ indicates a combination of corrections from different systems, and ‘>’ indicates a pipeline where the output of one system is the input to the other.

4 Official evaluation results

Our submission to the shared task is the result of our best hybrid system, described in Section 2 and summarised in Figure 1. Similar to the official development set, the test set comprises 350 texts (85,668 tokens in 4,477 sentences) written by native and non-native English learners.

Systems were evaluated using the ERRANT scorer, with span-based correction $F_{0.5}$ as the primary measure. In the restricted track, where participants were constrained to use only the provided training sets, our submitted system ranked fourth\(^9\) out of 21 participating teams. The official results of our submission in terms of span-level correction, span-level detection and token-level detection, including our system rankings, are reported in Table 3. It is worth noting that our correction system yielded particularly high performance on error detection tasks, ranking third on span-level detection and second on token-level detection. We believe that much of the success in error detection can be credited to the error detection auxiliary objectives introduced in the convolutional neural system (see Section 2.1) and the error detection features used in our final re-ranking system (see Section 2.4).

We also report span-level correction performance in terms of different CEFR levels (A, B, and C),\(^10\) as well as on the native texts only (N) in Table 4. Our final error correction system performs best on advanced learner data (C), achieving an $F_{0.5}$ score of 73.28, followed by intermediate learner data (B), native data (N), and lastly beginner learner data (A). The difference between the highest and lowest $F_{0.5}$ scores is 8.12 points. We also note that the system seems to be handling errors made by native students effectively even though it has not been trained on any native parallel data. Overall, we observe that our system generalises well across native and non-native data, as well as across different proficiency/CEFR levels.

In order to better understand the performance of our hybrid error correction system, we perform a detailed error analysis. This helps us understand the strengths and weaknesses of the system, as well as identify areas for future work. Error type-specific performance is presented in Table 5. We can see that our system achieves the highest results on VERB:INFL (verb inflection) errors with an $F_{0.5}$ of 93.75. However, the result is not truly representative as there are only 8 verb inflection errors in the test data, and our system successfully corrects 6 of them. The error type that follows is ORTH (orthography), which comprises case and/or whitespace errors. A high precision score of 89.11 is observed, suggesting that our system is particularly suitable for these kind of errors. We also observe that our system is effective at correcting VERB:SVA (subject–verb agreement) errors, achieving an $F_{0.5}$ of 80.08. Results for ADJ:FORM (adjective form; $F_{0.5}$=78.95) and CONTR (contraction; $F_{0.5}$=77.92) are high; however, these error types only account for small fractions of the test set (0.188% and 0.245% respectively).

The worst performance is observed for type CONJ (conjunction), with an $F_{0.5}$ of 28.46. Our system successfully corrected 7 conjunction errors, while missed 20 and made 17 unnecessary changes. We note that our system is less effective at correcting open-class errors

\(^9\)The system is tied for third place as the difference in $F_{0.5}$ is negligible.

\(^10\)https://www.cambridgeenglish.org/exams-and-tests/cefr/
such as NOUN (noun; $F_{0.5}=34.75$), OTHER (other; $F_{0.5}=38.95$), VERB (verb; $F_{0.5}=39.80$), ADJ (adjective; $F_{0.5}=41.94$) and ADV (adverb; $F_{0.5}=51.65$) errors. As noted by Kochmar (2016), such error types are quite challenging for error detection and correction systems: content words express meaning, and their semantic properties should be taken into account. Unlike errors in function words, content word errors are often less systematic; e.g., [person → people, ambulate → walk, big → wide, speedily → quickly].

## 5 Conclusion

In this paper, we have presented a hybrid approach to error correction that combines a convolutional and a Transformer-based neural system. We have explored different combination techniques involving sequential pipelines, candidate generation and re-ranking. Our best hybrid system submitted to the restricted track of the BEA 2019 shared task yields a span-level correction score of $F_{0.5}=66.75$, placing our system in the fourth place out of 21 participating teams. High results were observed for both span-level and token-level error detection (ranking our system third and second respectively), suggesting that our error correction system can also effectively detect errors.

Detailed analyses show that our system generalises well across different language proficiency levels (CEFR) and native / non-native domains. An error-type analysis showed that our system is particularly good at correcting verb inflection, orthography and subject–verb agreement errors, but less effective at correcting open-class word errors which are less systematic.

## Acknowledgments

We thank the anonymous reviewers for their valuable feedback. Zheng Yuan, Marek Rei and Helen Yannakoudakis were supported by Cambridge Assessment, University of Cambridge. Felix Stahlberg and Bill Byrne acknowledge support for this work by the U.K. Engineering and Physical Sciences Research Council (EPSRC) grant EP/L027623/1 and grant EP/P020259/1 which provided Cambridge Tier-2 resources operated by the University of Cambridge Research Computing.
We thank the NVIDIA Corporation for the donation of the Titan X Pascal GPU used in this research.

References


Ziang Xie, Anand Avati, Naveen Arivazhagan, Dan Jurafsky, and Andrew Y. Ng. 2016. Neural language correction with character-based attention. arXiv, abs/1603.09727.


