SynGEC: Syntax-Enhanced Grammatical Error Correction with a Tailored GEC-Oriented Parser

Anonymous ACL submission

Abstract

Despite their tremendous success, current cutting-edge grammatical error correction (GEC) models make little use of syntactic knowledge, which plays an important role when humans try to understand and fix ungrammatical sentences. This work proposes a syntax-enhanced GEC approach (SynGEC), to incorporate syntactic information into the encoder part of GEC models. The key challenge for this idea is that the performance of off-the-shelf parsers dramatically drops since they are usually trained on clean grammatical sentences. To confront this challenge, we propose to build a tailored GEC-oriented parser (GOPar) using parallel GEC training data as a pivot. First, we present an extended annotation scheme that allows us to represent both grammatical errors and syntactic structure under a unified tree structure. Then, we obtain parse trees of the source incorrect sentences by projecting trees of the target correct sentences and using them for training GOPar. We employ graph convolution networks to encode tree structures produced by GOPar. Experiments on three English/Chinese benchmark datasets show that our proposed SynGEC approach consistently and substantially outperforms the strong baselines and achieves new single-model state-of-the-art performance on all datasets.

1 Introduction

Given a potentially ungrammatical sentence, the grammatical error correction (GEC) task aims to produce a grammatical target sentence (Grundkiewicz et al., 2020; Wang et al., 2021). Recent mainstream approaches treat GEC as a monolingual machine translation (MT) task (Yuan and Briscoe, 2016; Junczys-Dowmunt et al., 2018). Standard encoder-decoder based MT models, e.g., Transformer (Vaswani et al., 2017), have emerged as a dominant paradigm and achieved state-of-the-art (SOTA) results on various GEC datasets (Rothe et al., 2021; Stahlberg and Kumar, 2021; Sun et al., 2022; Zhang et al., 2022). Despite their impressive achievements, most works take the input sentence as a sequence of tokens, and totally ignore the internal syntactic or semantic structure.

Compared with MT, GEC has two peculiarities that directly motivate this work. First, the training data for GEC models is much less abundant, and the data sparseness problem may be alleviated by incorporating linguistic structure knowledge like syntax. As shown in Table 1 and Table 4, the English and Chinese GEC tasks only have about 126K and 157K high-quality labeled source/target sentence pairs for training, if not considering the highly noisy crowd-annotated Lang8 data (Mita et al., 2020). Second, many errors in ungrammatical sentences are intrinsically correlated with syntactic parsing. For example, some errors, such as inconsistency in tense or singular-vs-plural forms, can be better detected and corrected with the help of long-range syntactic dependencies.

In this paper, we propose SynGEC, an approach for injecting the syntactic structure of the input sentence into the encoder part of GEC models. The critical challenge for this idea is the unsatisfactory parsing performance on ungrammatical sentences. Typically, an off-the-shelf syntactic parser is trained upon clean treebanks, while the input sentences for GEC usually contain various grammatical errors. Such a big gap inevitably leads to many ill-formed parse trees (Hashemi and Hwa, 2016). As far as we know, two recent works try to incorporate syntactic knowledge into GEC, but both fail to directly handle this challenge (Wan and Wan, 2021; Li et al., 2022) (see Section 6 for more detailed discussion).

Meanwhile, there has been a line of works that try to adapt syntactic parsing for ungrammatical texts by extending the annotation guidelines and manually annotating data (Dickinson and Ragheb, 2009; Berzak et al., 2016; Nagata and Sakaguchi, 2018).
2016). However, manual annotation is extremely expensive, and thus existing annotated datasets are of a very small scale.

To confront the challenge of unreliable performance of off-the-shelf parsers on ungrammatical sentences, we propose to train a tailored GEC-oriented parser (GOPar). The basic idea is utilizing parallel source/target sentence pairs in the GEC training data. First, we parse the target correct sentences using a vanilla off-the-shelf parser. Then, we construct the tree for the incorrect sentences via tree projection. To accommodate grammatical errors, we propose an extended annotation scheme based on several straightforward rules, which allows us to represent both grammatical errors and syntactic structure in a unified tree. Finally, we train GOPar directly on the automatically constructed trees of the source sentences in the GEC training data.

To incorporate syntactic information, we cascade several graph convolutional network (GCN) layers (Kipf and Welling, 2017) above the encoder of our baseline Transformer-based GEC model. We conduct experiments on two widely-used English GEC evaluation datasets, i.e., CoNLL-14 (Ng et al., 2014) and BEA-19 (Bryant et al., 2019), and the recently proposed Chinese MuCGEC dataset (Zhang et al., 2022). Results show that our SynGEC approach achieves consistent and substantial improvement on all datasets, even when the baseline model is enhanced with BART (Lewis et al., 2020). SynGEC achieves new single-model SOTA results: 69.6 F0.5 score on CoNLL-14, 72.9 F0.5 score on BEA-19, and 46.8 F0.5 score on MuCGEC, outperforming previous SOTA under comparable settings by large margins (+3.5/+0.5/+7.3). We will release our code at http://github.com.

2 Our GEC-Oriented Parser

This section describes our tailored GOPar, which is more competent in parsing ungrammatical sentences than off-the-shelf parsers.

2.1 An Extended Annotation Scheme

The standard annotation scheme for representing syntax is designed for grammatical sentences, and thus does not cover many structures in grammatically erroneous sentences. Therefore, to obtain a tailored parser, our first task is to extend the annotation scheme and, more specifically, to design a complementary set of annotation guidelines to handle different grammatical mistakes. With this scheme, we can use unified tree structure to represent both grammatical errors and syntactic information.

As shown in Figure 1, we propose a simple extended annotation scheme based on several straightforward rules, corresponding to the three types of grammatical errors, i.e., substituted, redundant and missing (Bryant et al., 2017). Correspondingly, we add three labels into the original syntactic label set, i.e., “S”, “R” and “M”, to capture three kinds of errors. Since such categorization is also adopted in the grammatical error detection (GED) task (Yuan et al., 2021), we refer to them as GED labels.

Substituted errors (S) include spelling errors, tense errors, singular/plural inconsistency errors, etc. For simplicity, we do not consider such fine-grained categories, and use a single “S” label to indicate that the word should be replaced by another one, as shown in Figure 1(b).

Redundant errors (R) mean that some words should be deleted. For each redundant word, we let it depend on its right-side adjacent word, with a label “R”, even if the right-side adjacent word is redundant also, as shown in Figure 1(c).

Missing errors (M) mean that some words should be inserted. For each missing word, we assign a label “M” to the incoming arc of its right-side adjacent word, unless the missing word is at the end of the sentence, as shown in Figure 1(d). If several consecutive words are missing, the annotation remains the same with when a single word is missing. Moreover, since a missing word may have children in the tree of the correct sentence, we let them depend on the head word of the missing

Figure 1: Illustration of our extended annotation scheme for all three kinds of grammatical errors.
word, without changing their syntactic labels.

Based on the above simple rules, we aim to perform minimal adjustments and make the resulting tree as consistent as possible with the syntactic tree of the correct sentence. Meanwhile, please kindly notice that our annotation scheme may encounter problems when different types of errors occur in consecutive positions of a sentence. Taking “But was no buyers” as an example, we need to replace “was” with “were” and then insert “there” before “were” at the same time. Therefore, according to our rules, the label of the incoming arc of “was” can be either “S” or “M”, leading to a label conflict. To decide a unique label, we simply define a priority order: “S”>“R”>“M”.

2.2 Building GOPar

Based on the above annotation scheme, we propose to train our tailored GOPar using the parallel GEC training data \( D = \{ (x_i, y_i) \} \) as a pivot. The major goal is to automatically generate high-quality parse trees for large-scale sentences with realistic grammatical errors, and use them to train a parser suitable for parsing ungrammatical sentences. Figure 2 illustrates the workflow, consisting of the following four steps.

First, we use an off-the-shelf parser to parse the target correct sentences (e.g., \( y_i \)) of the GEC training data. Since such sentences are free from grammatical errors, the parser is more likely to produce reliable parse trees.

Second, we use ERRANT (Bryant et al., 2017)\(^1\) to extract all grammatical errors in the source incorrect sentence (e.g., \( x_i \)) according to the alignments between \( x_i \) and \( y_i \).

Third, we construct the parse tree of \( x_i \) by projecting the tree of \( y_i \). For words that are not related to any errors, dependencies (arcs) and labels are directly copied; for those related to errors, dependencies and labels are assigned according to the rules introduced in Section 2.1.

Fourth, with constructed parse trees for all source-side sentences in \( D \), we then use them as a treebank to train our tailored GOPar.

An alternative way to build a tailored GOPar is directly utilizing manually labeled treebanks. Since existing treebanks only contain grammatical sentences, we can inject synthetic errors based on rules or back-translation models (Foster et al., 2008; Cahill, 2015). Then, we can produce parse trees for ungrammatical sentences analogously through the above second and third steps. However, our preliminary experiments show that GOPar built in this way is much inferior and can only slightly improves our baseline GEC model. We suspect that the reasons are two-fold. On the one hand, there is a considerable gap between synthetic and real grammatical errors; on the other hand, the generated data is not enough to train GOPar adequately due to the limited scale of existing treebanks.

3 The DepGCN-based GEC Model

In this section, we describe our DepGCN-based GEC Model. Inspired by Wan and Wan (2021), we first adopt GCN (Kipf and Welling, 2017) to encode the dependency syntax trees of the source sentence. Then, we feed the encoded syntactic information into a Transformer-based GEC model.

3.1 Transformer Backbone

We model GEC as a sequence-to-sequence task and employ the commonly used Transformer model (Vaswani et al., 2017) as the backbone. The Transformer is composed of an encoder and a decoder. The encoder utilizes the multi-head self-attention mechanism to get the contextualized representations of each token in the source sentence. The decoder has a similar architecture while additionally containing a masked multi-head self-attention module to model the generated token information.

\(^1\)https://github.com/chrisjbryant/errant

Figure 2: The workflow for obtaining our tailored GOPar.
During training, the objective function is minimizing the teacher forcing negative log-likelihood loss (Williams and Zipser, 1989), formally:

$$\mathcal{L}(\theta) = -\log(P(y|x; \theta))$$

$$= -\log \sum_{i=1}^{n} (P(y_i|y_{<i}; x; \theta))$$  \hspace{1cm} (1)

where $\theta$ is trainable model parameters, $x$ is the source sentence, $y = \{y_1, y_2, ..., y_n\}$ is the ground-truth target sentence with $n$ tokens, and $y_{<i} = \{y_1, y_2, ..., y_{i-1}\}$ is the visible tokens in $i$-th training time step.

During inference, we utilize the beam search decoding (Wiseman and Rush, 2016) to find an optimal sequence $o$ by maximizing the conditional probability $P(o|x; \theta)$.

Previous works show that PLMs, e.g., BART (Lewis et al., 2020) and T5 (Raffel et al., 2020), can improve GEC performance by a large margin (Rothe et al., 2021; Sun et al., 2022). We further use BART to build stronger baseline, since it shares the same model architecture with our Transformer backbone. Specifically, we use the BART parameters to initialize our Transformer backbone and then continue training on GEC training data. More details are discussed in Section 4.1 and Section 5.

### 3.2 Dependency GCN (DepGCN)

We employ GCN to encode the dependency syntax information, named DepGCN. The DepGCN module stacks several identical blocks, and each block has a GCN sub-layer and a feed-forward sub-layer.

For the GCN sub-layer, we introduce the information of the dependency arcs and dependency labels simultaneously. We compute the output $h^{(l)}_i$ of $l$-th GCN at the $i$-th token as:

$$h^{(l)}_i = \text{ReLU}(\sum_{j=1}^{n} A_{ij} W^{(l)}_1 h^{(l-1)}_j + \ell^{(l)}_i)$$  \hspace{1cm} (2)

where $A \in \mathbb{R}^{n \times n}$ denotes the adjacency matrix, $e_{ij} \in \mathbb{R}^{d}$ is the embedding of the dependency label between word $w_i$ and word $w_j$, $W \in \mathbb{R}^{d \times 2d}$ and $b \in \mathbb{R}^{d}$ are model parameters. ReLU (Nair and Hinton, 2010) is the activation function, and $\oplus$ refers to the vector concatenation.

To reduce error propagation of GOPar, follow Zhang et al. (2020a), we use the arc probability matrix derived from GOPar as the adjacency matrix $A$, which can provide all latent syntactic structures.

We then feed the outputs of the GCN sub-layer to the feed-forward (FF) layer that contains two linear transformations with a ReLU activation function in between, as shown below:

$$\text{FF}(h) = \text{ReLU}(W_1 h + b_1)W_2 + b_2$$  \hspace{1cm} (3)

### 3.3 Integrating Syntactic Information

We put DepGCN at the junction between Transformer encoder and decoder, and build a DepGCN-based GEC model. As depicted in Figure 3, given an input sentence $x$ and corresponding syntactic information $x^{\text{syn}}$ (including dependency arcs and labels) from GOPar, we first obtain the contextualized representations $H^c$ of $x$ from Transformer encoder. Then, we feed $H^c$ and $x^{\text{syn}}$ into DepGCN and get the syntax-enhanced representations $H^{\text{syn}}$. Finally, we use the weighted-sum operation to derive the fusion representations $H^f$ as the input of Transformer decoder:

$$H^f = \beta H^c + (1 - \beta) H^{\text{syn}}$$  \hspace{1cm} (4)

where $\beta \in (0, 1)$ is a hyper-parameter. The training and inference processes are consistent with the Transformer baseline.
## 4 Experiments on English GEC

### 4.1 Settings

#### Datasets and evaluation. We first pre-train our model on the cleaned version of the Lang8 dataset (CLang8)\(^2\) released by Rothe et al. (2021). Then, we use the FCE dataset (Yannakoudakis et al., 2011), the NUCLE dataset (Dahlmeier et al., 2013) and the W&I+LOCNESS train-set (Bryant et al., 2019) for model fine-tuning following previous studies. Like Omelianchuk et al. (2020), we decompose the fine-tuning procedure into two stages: 1) fine-tuning on FCE+NUCLE+W&I+LOCNESS; 2) further fine-tuning only on the small-scale but high-quality W&I+LOCNESS. For evaluation, we report average results over three distinct runs on the CoNLL-14 test-set\(^3\) (Ng et al., 2014) evaluated by M2Scorer (Dahlmeier and Ng, 2012) and BEA-19 test-set (Bryant et al., 2019) evaluated by ERRANT (Bryant et al., 2017). The BEA-19 dev-set serves as validation data during the whole training. The statistics of above datasets are shown in Table 1.

### GEC model details. We adopt Fairseq\(^4\) (Ott et al., 2019) to build our Transformer baseline and DepGCN-based model. For the DepGCN-based model, we stack 3 DepGCN blocks and set aggregation factor \(\beta\) in Equation 4 to 0.5. We apply BPE (Sennrich et al., 2016) to generate a 32K shared subword vocabulary. We re-implement the Dropout-Src mechanism proposed by Junczys-Dowmunt et al. (2018) and perform it on the source word embedding matrix to alleviate over-fitting. More model details are discussed in Appendix A.

### GOPar details. We employ the biaffine parser (Dozat and Manning, 2017) as the model architecture of GOPar. For implementation, we adopt the parsing toolkit Supar\(^5\) (Zhang et al., 2020b) and incorporate BART into our DepGCN-based model. Firstly, we fine-tune the BART-initialized Transformer backbone until it converges. Secondly, we add an auxiliary DepGCN module into the converged Transformer backbone and only tune the DepGCN parameters on the same training data.

### 4.2 Main Results

The main results are listed in Table 2. In the top group of results without PLMs, our SynGEC approach achieves 65.2/68.4 \(F_{0.5}\) scores on CoNLL-14 and BEA-19 test-sets, respectively, outperforming all other systems utilizing syntax. The performance of SynGEC is only lower than Stahlberg and Kumar (2021) on both test-sets as they use extra huge synthetic corpus with 540M sentences while we only use 2.4M. The incorporation of syntactic information provided by GOPar leads to 3.5/4.2 \(F_{0.5}\) improvements over our baseline, which demonstrates that the syntactic knowledge from GOPar is quite helpful for GEC and our DepGCN-based GEC model can effectively capture it.

In the bottom group, our SynGEC approach augmented with BART achieves 69.6/72.9 \(F_{0.5}\) scores, continuously outperforming other PLM-augmented models under similar sizes\(^7\). After removing syntax, the \(F_{0.5}\) scores decline by 0.9 on both datasets, setting hyper-parameters following their paper. After compare several popular PLMs, we use ELECTRA (Clark et al., 2020) to provide the contextual token representations\(^6\). The GOPar training data is generated from CLang8 GEC dataset (Rothe et al., 2021) by the data construction approach described in detail in Section 2. For using GOPar in GEC, since the inputs of GEC models are subword sequences while the basic token unit in syntax trees is the word, we add arcs from all subwords of \(w\) to all subwords of \(v\) if there exits an arc from word \(w\) to word \(v\) in the word-level tree before feeding this tree into GEC models.

### Incorporating BART. To verify that the syntactic knowledge is still useful after incorporating a powerful PLM, we use BART (Lewis et al., 2020) to initialize the Transformer backbone of our models. It is noteworthy that we adopt a two-stage training procedure to keep the training stable when incorporating BART into our DepGCN-based model.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Sentences</th>
<th>%Error</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLang8</td>
<td>2,372,119</td>
<td>57.8</td>
<td>Pre-training</td>
</tr>
<tr>
<td>FCE</td>
<td>34,490</td>
<td>62.6</td>
<td>Fine-tuning I</td>
</tr>
<tr>
<td>NUCLE</td>
<td>57,151</td>
<td>38.2</td>
<td>Fine-tuning I</td>
</tr>
<tr>
<td>W&amp;I+LOCNESS</td>
<td>34,308</td>
<td>66.3</td>
<td>Fine-tuning I&amp;II</td>
</tr>
<tr>
<td>BEA-19-Dev</td>
<td>4,384</td>
<td>65.2</td>
<td>Validation</td>
</tr>
<tr>
<td>CoNLL-14-Test</td>
<td>1,312</td>
<td>72.7</td>
<td>Testing</td>
</tr>
<tr>
<td>BEA-19-Test</td>
<td>4,477</td>
<td>-</td>
<td>Testing</td>
</tr>
</tbody>
</table>

Table 1: Statistics of English GEC datasets.
which reveals that the contribution from adaptive syntax and PLMs does not fully overlap.

### 4.3 Analysis and Discussion

**Effectiveness of GOPar.** In Table 2, we present the results of using an off-the-shelf parser\(^8\) to provide syntactic knowledge (GOPar → Off-the-shelf Parser). After changing the parser, the impact of syntax becomes marginal under all settings. This observation implies that the performance gains contributed from syntax are highly contingent on the quality of parses. We look further into the parses and find that GOPar is more robust when facing grammatical errors and can further identify such errors, while the off-the-shelf parser is vulnerable and tends to provide incorrect parses. So we can draw a conclusion that the task adaptation of parsers is essential when applying syntax to the GEC task.

**Decomposition of syntactic information.** To gain more insights on how adaptive syntactic information works, we decompose it into three parts: 1) the arc information, which means only using the topological structure of the syntax tree; 2) the GED label information, which refers to the special labels “S”, “R” and “M” for marking erroneous tokens; and 3) the syntax label information, such as “subj” and “iobj” for different syntactic relations. We conduct an ablation study to explore the effect of each kind of information for GEC\(^9\), as shown in Table 3. First, removing GED labels or syntax labels reduce the performance of the SynGEC model to the similar extent, which indicates that they are equally important to GEC. Second, when we only use the arc information, i.e., removing all labels, the recall drops sharply while the precision increases notably compared with the baseline. We speculate that the contribution from arc information is mainly on preventing GEC models from being misled by inappropriate context. Third, the full SynGEC model utilizing all three information achieves the best performance, which implies that they have intrinsic complementary strengths.

**Error type performance.** Figure 4 shows more fine-grained evaluation results on different error types on BEA-19-dev. The results support that syntactic information from GOPar is beneficial for most error types. Specifically, syntactic knowledge

---

\(^8\)We use *biaffine-dep-roberta-en* model provided by Supar.

\(^9\)For “w/o GED Labels”, we force the parser to skip GED labels and select the syntax label with the highest probability when predicting. For “w/o Syntax Labels”, we replace all syntax labels with “O” in the results. For “w/o All Labels”, we do not feed the label embeddings into the DepGCN module.

**Table 2: Single-model results on English GEC test-sets.** Our results are averaged over three runs with different random seeds. **Layer, Hidden** and **FFN** denote the depth, hidden size and feed-forward network size of Transformer. “w/ PLM” means using pre-trained language models. “w/ syntax” means using syntactic knowledge.

<table>
<thead>
<tr>
<th>System</th>
<th>Extra Data Size</th>
<th>Transformer Layer, Hidden, FFN</th>
<th>CoNLL-14-test</th>
<th>BEA-19-test</th>
</tr>
</thead>
<tbody>
<tr>
<td>w/o syntax</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kiyono et al. (2019)(^9)</td>
<td>70M</td>
<td>12+12,1024,4096</td>
<td>67.9</td>
<td>44.1</td>
</tr>
<tr>
<td>Lichtarge et al. (2020)(^1)</td>
<td>340M</td>
<td>12+12,1024,4096</td>
<td>69.4</td>
<td>43.9</td>
</tr>
<tr>
<td>Stahlberg and Kumar (2021)(^1)</td>
<td>540M</td>
<td>12+12,1024,4096</td>
<td>72.8</td>
<td>49.5</td>
</tr>
<tr>
<td>Our Baseline(^2)</td>
<td>2.4M</td>
<td>64+6,152,2048</td>
<td>69.6</td>
<td>42.4</td>
</tr>
</tbody>
</table>

| w/o PLM                        |                 |                                |               |            |
| w/ syntax                      |                 |                                |               |            |
| Wan and Wan (2021)\(^9\)      | 10M             | 64+6,512,2048                 | 74.4          | 39.5       | 63.2       | 74.5       | 48.6       | 67.3       |
| Li et al. (2022)\(^1\)        | 30M             | 12+12,1024,4096               | 66.7          | 38.3       | 58.1       | -           | -           | -           |
| SynGEC\(^2\)                  | 2.4M            | 64+6,152,2048                 | 71.6          | 47.9       | **65.2**   | 70.9       | 59.9       | **68.4**   |
| GOPar→Off-the-shelf Parser    | 2.4M            | 64+6,152,2048                 | 70.5          | 42.6       | 62.4       | 67.3       | 55.4       | 64.5       |

| w/ PLM                         |                 |                                |               |            |
| w/ syntax                      |                 |                                |               |            |
| Kaneko et al. (2020)\(^9\)    | 70M             | 12+12,1024,4096               | 69.2          | 45.6       | 62.6       | 67.1       | 60.1       | 65.6       |
| Katsumata and Komachi (2020)   | -               | 12+12,1024,4096               | 69.3          | 45.0       | 62.6       | 68.3       | 57.1       | 65.6       |
| Omelianchuk et al. (2020)\(^1\) | 9M              | 12+0,768,3072                 | 77.5          | 40.1       | 65.3       | 79.2       | 53.9       | **72.4**   |
| Rothe et al. (2021)\(^2\)     | 2.4M            | 12+12,1024,4096               | -             | 66.1       | -           | -           | 72.1       | **72.4**   |
| Our Baseline\(^2\)            | 2.4M            | 12+12,1024,4096               | 75.6          | 50.4       | **68.7**   | 74.0       | 64.9       | 72.0       |

| w/ PLM                         |                 |                                |               |            |
| w/ syntax                      |                 |                                |               |            |
| Li et al. (2022)\(^1\)        | 30M             | 12+12,1024,4096               | 68.1          | 44.1       | 61.4       | -           | -           | -           |
| SynGEC\(^2\)                  | 2.4M            | 12+12,1024,4096               | 76.6          | 51.1       | **69.6**   | 75.1       | 65.5       | **72.9**   |
| GOPar→Off-the-shelf Parser    | 2.4M            | 12+12,1024,4096               | 76.1          | 50.0       | 68.9       | 74.6       | 64.1       | 72.3       |
Figure 4: Performance on a selection of error types from ERRANT (Bryant et al., 2017) on BEA-19-dev. Numbers in parentheses represent the percentages of error types. We exclude error types that account for less than 1%.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Sentences</th>
<th>%Error</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lang8</td>
<td>1,220,906</td>
<td>89.5</td>
<td>Training</td>
</tr>
<tr>
<td>HSK</td>
<td>15,6870</td>
<td>60.8</td>
<td>Training</td>
</tr>
<tr>
<td>MuCGEC-Dev</td>
<td>1,125</td>
<td>95.1</td>
<td>Validation</td>
</tr>
<tr>
<td>MuCGEC-Test</td>
<td>5,938</td>
<td>92.2</td>
<td>Testing</td>
</tr>
</tbody>
</table>

Table 4: Statistics of Chinese GEC datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>PLM</th>
<th>Syntax</th>
<th>P</th>
<th>R</th>
<th>F0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zhang et al. (2022)</td>
<td>✓</td>
<td>✓</td>
<td>43.81</td>
<td>28.56</td>
<td>39.58</td>
</tr>
<tr>
<td>Baseline</td>
<td>✗</td>
<td>✓</td>
<td>43.79</td>
<td>25.93</td>
<td>38.49</td>
</tr>
<tr>
<td>SynGEC</td>
<td>✓</td>
<td>✓</td>
<td>46.88</td>
<td>27.68</td>
<td>40.58</td>
</tr>
<tr>
<td>Baseline</td>
<td>✓</td>
<td>✓</td>
<td>54.21</td>
<td>28.51</td>
<td>45.93</td>
</tr>
<tr>
<td>SynGEC</td>
<td>✓</td>
<td>✓</td>
<td>55.59</td>
<td>28.72</td>
<td>46.83</td>
</tr>
</tbody>
</table>

Table 5: Single-model results on MuCGEC-test.

significantly improves the GEC model’s ability to correct context-aware errors, such as DET, PREP, PUNCT, VERB:SVA, and VERB:TENSE. Correcting such errors requires long-distance information, which syntax can effectively provide. The syntax also helps solve word-ordering (WO) errors, which need structural information to correct. Besides, the performance on PRON, OTHER, VERB is also substantially improved. Meanwhile, we note that a small subset of types are negatively affected, like ADJ, MORPH, SPELL, and NOUN. After more careful observation, we find that their corrections mainly depend on local information, where syntactic knowledge may not help much.

5 Experiments on Chinese GEC

Datasets and evaluation. We directly follow the recent work of Zhang et al. (2022), who propose a multi-reference multi-source Chinese GEC evaluation dataset named MuCGEC. We report P/R/F0.5 values on MuCGEC-test using the official char-based evaluation tool. MuCGEC-dev is used for hyper-parameter tuning. For training data, we use the Chinese Lang8 dataset (Zhao et al., 2018) and HSK dataset (Zhang, 2009). The statistics of above-mentioned datasets are shown in Table 4.

Char-based GOPar. Current Chinese GEC models usually treat the input sentence as a character sequence and do not perform word segmentation. In contrast, syntactic parsers typically treat the input sentence as a word sequence. To handle this mismatch, we follow Yan et al. (2020) and build a char-based GOPar. The basic idea is to convert a word-based tree into a char-based one by letting each char depends on its right-hand one inside multi-char words.

Use of BART. We employ the recently proposed Chinese BART (Shao et al., 2021), which is originally implemented with the HuggingFace Transformers toolkit10 (Wolf et al., 2020). We manage to wrap their code and use it on our Fairseq implementation, which is faster than the original implementation. More importantly, we find many common characters are missing in its vocabulary and thus add 3,866 Chinese characters and punctuation marks from Chinese Gigaword and Wikipedia corpora, leading to a substantial performance boost according to our preliminary experiments.

Results are presented in Table 5. When not

10https://huggingface.co/
using BART, our SynGEC outperforms the Transformer baseline by 2.09 F\(_{0.5}\) score on MuCGEC-test. When using BART, our baseline already outperforms the previous SOTA system (Zhang et al., 2022), and SynGEC further improves the F\(_{0.5}\) score by 0.90. These results indicate that our proposed SynGEC approach can be steadily effective for different languages.

6 Related Works

Grammatical error correction. Recent works mainly formulate GEC as a monolingual translation task and handle it with encoder-decoder-based MT models (Yuan and Briscoe, 2016; Junczys-Dowmunt et al., 2018), among which Transformer (Vaswani et al., 2017) has become a dominant paradigm. With the help of synthetic training data (Lichtarge et al., 2019; Yasunaga et al., 2021) and large PLMs (Kaneko et al., 2020; Katsumata and Komachi, 2020), Transformer-based GEC models have achieved SOTA performance on various benchmark datasets (Rothe et al., 2021; Stahlberg and Kumar, 2021).

Meanwhile, the sequence-to-edit (Seq2Edit) approach emerges as a competitive alternative, which predicts a sequence of edit operations to achieve correction (Gu et al., 2019; Awasthi et al., 2019; Omelianchuk et al., 2020). Although this work adopts the Transformer-based GEC models as the baseline, our SynGEC approach can also be applied to Seq2Edit models straightforwardly, which we leave to the future work.

Parsing ungrammatical sentences. Despite the success of syntactic parsing on clean sentences (Dozat and Manning, 2017; Zhang et al., 2020b), the performance of off-the-shelf parsers dramatically degrades when confronting ungrammatical sentences (Hashemi and Hwa, 2016). Previous studies mainly tackle this problem by annotating small-scale ungrammatical sentences and fine-tuning an off-the-shelf parser on them (Dickinson and Ragheb, 2009; Petrov and McDonald, 2012; Cahill, 2015; Berzak et al., 2016). In contrast, we propose to train a tailored parser on automatically generated syntax trees from parallel GEC data, which avoids the laborious manual annotation.

Syntax-enhanced GEC. Many previous works have demonstrated the effectiveness of utilizing syntactic information for various NLP tasks, such as machine translation (Bastings et al., 2017; Zhang et al., 2019), opinion role labeling (Zhang et al., 2020a), and semantic role labeling (Xia et al., 2019; Sachan et al., 2021). Meanwhile, we have found two recent works on syntax-enhanced GEC (Wan and Wan, 2021; Li et al., 2022). Both works directly produce the dependency tree of the input sentence using an off-the-shelf parser, without tailoring parsers for ungrammatical sentences. They both use graph attention networks (GAT) for tree encoding (Velickovic et al., 2018). Besides the dependency tree, Li et al. (2022) exploits the constituent tree of the input sentence as well.

Compared with the above two works, the major contribution of our work is directly dealing with the severe performance drop issue via our tailored GOPar. We adopt GCN for tree encoding because our preliminary experiments show that it achieves similar performance but is faster. Moreover, our baseline GEC models achieve much higher performance than theirs, as shown in Table 2.

7 Conclusions

This paper presents a SynGEC approach that effectively incorporates syntactic knowledge into GEC models. The key idea is adjusting vanilla parsers to accommodate ungrammatical sentences. To achieve this goal, we first extend the standard syntactic annotation scheme to use a unified tree structure to encode both grammatical errors and syntactic structure. Then we obtain high-quality parse trees of ungrammatical sentences by projecting target-sentence trees into the source side in GEC training data, which are used for training our tailored GOPar. To incorporate parse trees produced by GOPar, we present a DepGCN-based GEC model based on Transformer. Experiments on three English/Chinese datasets show that our proposed SynGEC approach is effective and achieves new SOTA results under comparable settings.

Limitations and future work. We observe two possible limitations of our present work. First, off-the-shelf parsers may still produce noisy parse trees for the target correct sentences in the GEC training data, which further leads to noise in our projected trees for the source sentences. Second, so far, we only use three coarse-grained labels to distinguish grammatical errors in our annotation scheme, while fine-grained categories may further benefit GEC. As discussed above, both limitations may be strengthened by integrating ideas and resources achieved by previous works on manually annotating syntactic trees for ungrammatical sentences.
References


Christopher Bryant, Mariano Felice, Øistein A Andersen, and Ted Briscoe. 2019. The bea-2019 shared task on grammatical error correction. In Proceedings of BEA@ACL, pages 52–75.


Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: pretraining text encoders as discriminators rather than generators. In Proceedings of ICLR.


Zachao Li, Kevin Parnow, and Hai Zhao. 2022. Incorporating rich syntax information in grammatical error correction. *Information Processing and Management*.


Slav Petrov and Ryan McDonald. 2012. Overview of the 2012 shared task on parsing the web.


Table 6: Binary GED performance.

A Hyper-parameters

The main hyper-parameters used in our experiments are presented in Table 7. When not using PLMs, the total training time is about 3 hours. When using PLMs, the training costs about 7 hours.

B Download links of PLMs

The download links of PLMs used in our experiments are listed below. We employ ELECTRA (Clark et al., 2020; Cui et al., 2020) to build GOPar and BART (Lewis et al., 2020; Shao et al., 2021) to enhance our GEC model.

- ELECTRA-English-Large.
- ELECTRA-Chinese-Large.
- BART-English-Large.
- BART-Chinese-Large.

C The GED ability of GOPar

We evaluate the binary Grammatical Error Detection (GED) performance of GOPar on two mainstream GED dataset, i.e., BEA-19-dev (Bryant et al., 2019) and FCE-test (Yannakoudakis et al., 2011). We follow Rei and Yannakoudakis (2016) and report token-level P/R/F values for detecting incorrect labels. Table 6 shows the performance of GOPar and other leading GED models. When using the same training data, GOPar has a superior ability to detect grammatical errors. This phenomenon is very interesting and worthy of more in-depth study.

D Experiments on JFLEG

JFLEG (Napoles et al., 2017) is an English GEC evaluation dataset which focuses on fluency and uses the GLUE score (Napoles et al., 2015) as the evaluation metric. We evaluate the baseline and the SynGEC approach in Table 2 on JFLEG. Since JFLEG’s scale is relatively small (only 747 sentences), we choose to present the results in the appendix. From Table 8, we can see that the syntactic knowledge still continuously improves the GEC performance over baselines with/without PLMs.

E Performance Regarding the Input Sentence Length

Figure 5 shows performance on CoNLL-14-test in terms of input lengths. The baselines heavily degenerate when encountered with long sentences.
After introducing syntactic knowledge, it is clear that larger improvements are obtained for longer sentences in both scenarios with or without the PLM, as syntax helps to solve errors related to remote information, which are common in long sentences.