A Neural Grammatical Error Correction System Built On Better Pre-training and Sequential Transfer Learning

Yo Joong Choe[*]
Kakao
yj.c@kakaocorp.com

Jiyeon Ham[*]
Kakao Brain
jiyeon.ham@kakaobrain.com

Kyubyong Park[*]
Kakao Brain
kyubyong.park@kakaobrain.com

Yeoil Yoon[*]
Kakao Brain
yeoil.yoon@kakaobrain.com

Abstract

Grammatical error correction can be viewed as a low-resource sequence-to-sequence task, because publicly available parallel corpora are limited. To tackle this challenge, we first generate erroneous versions of large unannotated corpora using a realistic noising function. The resulting parallel corpora are subsequently used to pre-train Transformer models. Then, by sequentially applying transfer learning, we adapt these models to the domain and style of the test set. Combined with a context-aware neural spellchecker, our system achieves competitive results in both restricted and low resource tracks in ACL 2019 BEA Shared Task. We release all of our code and materials for reproducibility. [1]

1 Introduction

Grammatical error correction (GEC) is the task of correcting various grammatical errors in text, as illustrated by the following example:

[Travel → **Travelling**] by bus is [exspensive → **expensive**], [bored → **boring**] and annoying.

While the dominant approach following the CoNLL-2014 Shared Task (Ng et al., 2014) has been different adaptations of phrase-based and statistical machine translation (PBSMT) models (Junczys-Dowmunt and Grundkiewicz, 2016), more recent work on GEC increasingly adopted partial (Grundkiewicz and Junczys-Dowmunt, 2018) or exclusive (Junczys-Dowmunt et al., 2018; Chollampatt and Ng, 2018a) use of deep sequence-to-sequence (seq2seq) architectures (Sutskever et al., 2014; Cho et al., 2014), which showed immense success in neural machine translation (NMT) (Bahdanau et al., 2015; Gehring et al., 2017; Vaswani et al., 2017).

In GEC, unlike NMT between major languages, there are not enough publicly available corpora (GEC's hundreds of thousands to NMT's tens of millions). This motivates the use of pre-training and transfer learning, which has shown to be highly effective in many natural language processing (NLP) scenarios in which there is not enough annotated data, notably in low-resource machine translation (MT) (Lample et al., 2018b; Ruder, 2019). As a result, recent GEC systems also include pre-training on various auxiliary tasks, such as language modeling (LM) (Junczys-Dowmunt et al., 2018), text revision (Lichtarge et al., 2018), and denoising (Zhao et al., 2019).

In this paper, we introduce a neural GEC system that combines the power of pre-training and transfer learning. Our contributions are summarized as follows:

- We pre-train our model for the denoising task using a novel noising function, which gives us a parallel corpus that includes realistic grammatical errors;
- We leverage the idea of sequential transfer learning (Ruder, 2019), thereby effectively adapting our pre-trained model to the domain as well as the writing and annotation styles suitable for our final task.
- We introduce a context-aware neural spellchecker, which improves upon an off-the-shelf spellchecker by incorporating context into spellchecking using a pre-trained neural language model (LM).

2 Background

2.1 Transformers

Transformers (Vaswani et al., 2017) are powerful deep seq2seq architectures that rely heavily on the

[*] Equal contribution. Listed alphabetically.
[1] `https://github.com/kakaobrain/helo_word`

attention mechanism (Bahdanau et al., 2015; Luong et al., 2015). Both the encoder and the decoder of a Transformer are stacks of Transformer blocks, each of which consists of a multi-head self-attention layer followed by a position-wise feed-forward layer, along with residual connection (He et al., 2016) and layer normalization (Ba et al., 2016). Each decoder block also attends (Luong et al., 2015) to the encoder outputs, in between its self-attention and feed-forward layers. Each input token embedding in a Transformer is combined with a positional embedding that encodes where the token appeared in the input sequence.

2.2 Copy-Augmented Transformers

Copy-augmented Transformers (Zhao et al., 2019) are a class of Transformers that also incorporate an attention-based copying mechanism (Gu et al., 2016; See et al., 2017; Jia and Liang, 2016) in the decoder. For each output token y_t at output position t, the output probability distribution of a copy-augmented Transformer is a mixture of the decoder's generative distribution p^{gen} and a copy distribution p^{copy}, which is defined as an encoder-decoder attention layer that assigns a distribution over tokens appearing in the source sentence. By defining a mixture weight parameter α_t^{copy} per each decoding step, the output distribution can be compactly represented as follows:

$$p(y_t) = (1 - \alpha_t^{\text{copy}}) \cdot p^{\text{gen}}(y_t) + \alpha_t^{\text{copy}} \cdot p^{\text{copy}}(y_t) \tag{1}$$

The mixture weight balances between how likely it is for the model to simply copy a source token, rather than generating a possibly different token.

2.3 Denoising Autoencoders

Denoising autoencoders (DAEs) (Vincent et al., 2008) are a class of neural networks that learns to reconstruct the original input given its noisy version. Given an input $\mathbf{x}$ and a (stochastic) noising function $\mathbf{x} \mapsto \tilde{\mathbf{x}}$, the encoder-decoder model of a DAE minimizes the reconstruction loss:

$$L(\mathbf{x}, \text{dec}(\text{enc}(\tilde{\mathbf{x}}))) \tag{2}$$

where L is some loss function.

Within the NLP domain, DAEs have been for pre-training in seq2seq tasks that can be cast as a denoising task. For example, in GEC, pre-trained DAEs have been used for correcting erroneous sentences (Xie et al., 2018; Zhao et al., 2019). Another example is low-resource machine translation (MT) (Lample et al., 2018b), pre-trained DAEs were used to convert word-by-word translations into natural sentences.

3 Related Work

Many recent neural GEC models (Junczys-Dowmunt et al., 2018; Lichtarge et al., 2018; Zhao et al., 2019) made use of the Transformer (Vaswani et al., 2017) architecture and saw results nearly as good as or better than convolutional (Chollampatt and Ng, 2018a,b) and recurrent (Grundkiewicz and Junczys-Dowmunt, 2018; Ge et al., 2018a) architectures. Recently, Zhao et al. (2019) further incorporated a copying mechanism (Gu et al., 2016; See et al., 2017; Jia and Liang, 2016) to the Transformer, highlighting the fact that most (from 83% in Lang-8 to 97% in CoNLL-2013) of the target tokens are exact copies of the corresponding source tokens.

Several prior results, both early (Brockett et al., 2006; Felice and Yuan, 2014) and recent (Ge et al., 2018a; Xie et al., 2018; Zhao et al., 2019), introduced different strategies for generating erroneous text that can in turn be used for model (pre-)training. One major direction is to introduce an additional "back-translation" model (Ge et al., 2018a; Xie et al., 2018), inspired by its success in NMT (Sennrich et al., 2016a), and let this model learn to generate erroneous sentences from correct ones. While these back-translation models can learn naturally occurring grammatical errors from the parallel corpora in reverse, they also require relatively large amounts of parallel corpora, which are not readily available in low resource scenarios. The other direction, which can avoid these issues, is to incorporate a pre-defined noising function, which can generate pre-training data for a denoising task (Zhao et al., 2019). Compared to (Zhao et al., 2019), our work introduces a noising function that generates more realistic grammatical errors.

4 Pre-training a Denoising Autoencoder on Realistic Grammatical Errors

Given the relative lack of parallel corpora for the GEC task, it is important to define a realistic pre-training task, from which the learned knowledge can transfer to an improved performance.

When pre-training a seq2seq model on an auxiliary denoising task, the choice of the noising function is important. For instance, in low-resource MT, Lample et al. (2018a,b) made use of a noising function that randomly insert/replace/remove tokens or mix up nearby words at uniform probabilities. They showed that this approach is effective in translating naive word-by-word translations into correct ones, both because the coverage of word-to-word dictionaries can be limited and because word order is frequently swapped between languages (e.g., going from SVO to SOV).

In GEC, Zhao et al. (2019) used a similar noising function to generate a pre-training dataset. However, we find that this noising function is less realistic in GEC than in low-resource MT. For example, randomly mixing up nearby words can be less effective for GEC than for low-resource MT, because word order errors occur less frequently than other major error categories, such as missing punctuations and noun numbers. Also, replacing a word to any random word in the vocabulary is a less realistic scenario than only replacing it with its associated common error categories, such as prepositions, noun numbers and verb tenses.

To generate realistic pre-training data, we introduce a novel noising function that captures in-domain grammatical errors commonly made by human writers.

4.1 Constructing Noising Scenarios

We introduce two kinds of noising scenarios, using a token-based approach and a type-based approach.

In the token-based approach, we make use of extracted human edits from annotated GEC corpora, using automated error annotation toolkits such as ERRANT (Bryant et al., 2017). We first take a subset of the training set, preferably one that contains in-domain sentences with high-quality annotations, and using an error annotation toolkit, we collect all edits that occurred in the parallel corpus as well as how often each edit was made. We then take edits that occur in for at least k times, where k is a pre-defined threshold (we fix $k = 4$ in our experiments), in order to prevent overfitting to this (possibly small) subset. These extracted edits include errors commonly made by human writers, including missing punctuations (e.g., adding a comma), preposition errors (e.g., *of* $\rightarrow$ *at*), and verb tenses (e.g., *has* $\rightarrow$ *have*). As a result, we obtain an automatically constructed dictionary of common edits made by human annotators on the in-domain training set. Then, we can define a realistic noising scenario by randomly applying these human edits, in reverse, to a grammatically correct sentence.

In the type-based approach, we also make use of *a priori* knowledge and construct a noising scenario based on token types, including prepositions, nouns, and verbs. For each token type, we define a noising scenario based on commonly made errors associated with that token type, but without changing the type of the original token. In particular, we replace prepositions with other prepositions, nouns with their singular/plural version, and verbs with one of their inflected versions. This introduces another set of realistic noising scenarios, thereby increasing the coverage of the resulting noising function.

4.2 Generating Pre-training Data

Our goal is to come up with an error function that introduces grammatical errors that are commonly made by human writers in a specific setting (in this case, personal essays written by English students). Given sets of realistic noising scenarios, we can generate large amounts of erroneous sentences from high-quality English corpora, such as the Project Gutenberg corpus (Lahiri, 2014) and Wikipedia (Merity et al., 2016).

We first check if a token exists in the dictionary of token edits. If it does, a token-based error is generated with the probability of 0.9. Specifically, the token is replaced by one of the associated edits with the probabilities proportional to the frequency of each edit. For example, the token *for* may be replaced with *during*, *in*, *four*, and also *for* (coming from a *noop* edit).

If a token is not processed through the token-based scenario, we then examine if it belongs to one of the pre-defined token types: in our case, we use prepositions, nouns, and verbs. If the token belongs to one such type, we then apply the corresponding noising scenario.

5 Sequential Transfer Learning

5.1 Transferring Pre-trained DAE Weights

As discussed in (Zhao et al., 2019), an important benefit of pre-training a DAE is that it provides good initial values for both the encoder and the decoder weights in the seq2seq model. Given a pre-trained DAE, we initialize our seq2seq GEC model using the learned weights of the DAE and train on all available parallel training corpora with smaller learning rates. This model transfer approach (Wang and Zheng, 2015) can be viewed as a (relatively simple) version of sequential transfer learning (Ruder, 2019).

5.2 Adaptation by Fine-tuning

As noted in (Junczys-Dowmunt et al., 2018), the distribution of grammatical errors occurring in text can differ across the domain and content of text. For example, a Wikipedia article introducing a historical event may involve more rare words than a personal essay would. The distribution can also be affected significantly by the writer's style and proficiency, as well as the annotator's preferred style of writing (e.g., British vs. American styles, synonymous word choices, and Oxford commas).

In this work, given that the primary source of evaluation are personal essays at various levels of English proficiency – in particular the W&I+LOCNESS dataset (Yannakoudakis et al., 2018) – we adapt our trained models to such characteristics of the test set by fine-tuning the model only on the training portion of W&I, which largely matches the domain of the development and test sets.[2] Similar to our training step in §5.1, we use (even) smaller learning rates. Overall, this sequential transfer learning framework can also be viewed as an alternative to oversampling in-domain data sources, as proposed in (Junczys-Dowmunt et al., 2018).

6 A Context-Aware Neural Spellchecker

Many recent GEC systems include an off-the-shelf spellchecker, such as the open-source package `enchant` (Sakaguchi et al., 2017; Junczys-Dowmunt et al., 2018) and Microsoft's Bing spellchecker (Ge et al., 2018a,b). While the idea of incorporating context into spellchecking has been repeatedly discussed in the literature (Flor and Futagi, 2012; Chollampatt and Ng, 2017), popular open-sourced spellcheckers such as `hunspell` primarily operate at the word level. This fundamentally limits their capacity, because it is often difficult to find which word is intended for without context. For example, given the input sentence *This is an esay about my favorite sport.*, `hunspell` invariably suggests *easy* as its top candidate for *esay*, which should actually be corrected as *essay*.

Our spellchecker incorporates context to `hunspell` using a pre-trained neural language model (LM). Specifically, we re-rank the top candidates suggested by `hunspell` through feeding each, along with the context, to the neural LM and scoring them.

[2] This is analogous to the NUCLE dataset matching "perfectly" with the CoNLL dataset, as noted in (Junczys-Dowmunt et al., 2018).

Source	Public?	# Sent.	# Annot.
Gutenberg	Yes	11.6M	n/a
Tatoeba	Yes	1.17M	n/a
WikiText-103	Yes	3.93M	n/a
FCE	Yes	33.2K	1
Lang-8	Yes	1.04M	1-8
NUCLE	Yes	57.2K	1
W&I-Train	Yes	34.3K	1
W&I+L-Dev	Yes	4.38K	1
W&I+L-Test	Yes	4.48K	5

Table 1: Summary of datasets. The first three datasets are unannotated English corpora, from which we generate parallel data for pre-training using a pre-defined noising function.

7 Experiments

Throughout our experiments, we use `fairseq`[3] (Ott et al., 2019), a publicly available sequence-to-sequence modeling toolkit based on PyTorch (Paszke et al., 2017). Specifically, we take `fairseq-0.6.1` and add our own implementations of a copy-augmented transformer model as well as several GEC-specific auxiliary losses.

7.1 Datasets & Setups

In Table 1, we summarize all relevant data sources, their sizes, whether they are public, and the number of annotators.

For pre-training, we use the Gutenberg dataset (Lahiri, 2014), the Tatoeba[4] dataset, and the WikiText-103 dataset (Merity et al., 2016). We learned through initial experiments that the quality of pre-training data is crucial to the final model's performance, because our DAE model assumes §4 that these unannotated corpora contain little grammatical errors. Our choice of corpora is based on both the quality and diversity of text: Guten-

[3] https://github.com/pytorch/fairseq

[4] https://tatoeba.org/eng/downloads

	Restricted (§7.5)	Low Resource (§7.6)	CoNLL-2014 (§7.7)
Error Extraction	W&I Train	W&I+L Dev-3K	NUCLE
Pre-training	Gutenberg, Tatoeba, WikiText-103		
Training	FCE, Lang-8, NUCLE, W&I Train	W&I+L Dev-3K	FCE, Lang-8, NUCLE
Fine-tuning	W&I Train	n/a	NUCLE
Validation	W&I+L Dev	W&I+L Dev-1K	CoNLL-2013
Test	W&I+L Test	W&I+L Test	CoNLL-2014

Table 2: Datasets used for each set of results. For the W&I+L development set, Dev-3K and Dev-1K respectively indicate a 3:1 train-test random split of the development set, such that the original proportions of English proficiency (A, B, C, N) are kept the same in each split. See Table 1 for more information about each dataset.

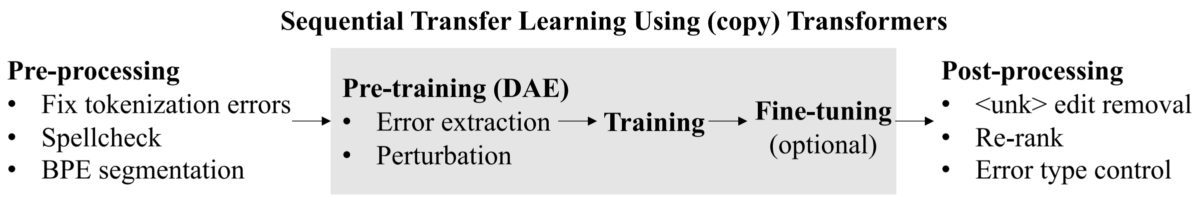

Figure 1: Overall pipeline for our approach. Gray shaded box includes the training steps for a seq2seq model.

berg contains clean novel writings with minimal grammatical errors, Tatoeba contains colloquial sentences used as sample sentences in dictionaries, and WikiText-103 contains "Good" and "Featured" articles from Wikipedia. Our final pre-training data is a collection of 45M (perturbed, correct) sentence pairs based on these datasets, with our noising approach (§4) applied multiple times to each dataset to approximately balance data from each source (1x Gutenberg, 12x Tatoeba, and 5x WikiText-103).

Our default setup is the "Restricted Track" scenario (§7.5) for the BEA 2019 Shared Task, where we use four data sources: the FCE dataset (Bryant et al., 2019), the Lang-8 dataset[5] (Mizumoto et al., 2011; Tajiri et al., 2012), the NUCLE (v3.3) dataset (Dahlmeier et al., 2013), and the newly released Write & Improve and LOCNESS (W&I+L) datasets (Yannakoudakis et al., 2018).[6] For the "Low Resource Track" (§7.6), we use a 3:1 train-test random split of the W&I+L development set, keeping the proportions of proficiency levels the same. In both tracks, we report our final results on the W&I+L test set, which contains 5 annotations. Further, because the W&I+L dataset is relatively new, we also include results on the CoNLL-2014 (Ng et al., 2014) dataset, with and without using the W&I+L dataset during training (§7.7). In Table 2, we summarize which datasets were used in each setup.

7.2 Pre-processing

As part of pre-processing, we first fix minor tokenization issues in the dataset using regular expressions. We use spaCy v1.9 (Honnibal and Montani, 2017) to make tokenization consistent with the final evaluation module (ERRANT).

This tokenized input is then fed to our context-aware neural spellchecker (§6). For the neural LM, we use a gated convolutional neural network language model (Dauphin et al., 2017) pre-trained on WikiText-103 (Merity et al., 2016).

During spellchecking, we also found it beneficial to fix casing errors within our context-aware spellchecking process. To fix case errors, we extract a list of words used in the capital form much more than their lower-case version (more than 99 times) in WikiText-103 (Merity et al., 2016). We then include a capitalized version of the word as a candidate in the LM re-scoring process if it appears in its capitalized form is in the extracted list of common capital words.

Before feeding spellchecked text into our seq2seq model, we apply byte-pair encoding

[5]As in previous results, we remove all duplicates but take multiple annotations (if available) the Lang-8 dataset, leaving only 575K parallel examples.

[6]See Appendix B for an exploratory data analysis.

(BPE) (Sennrich et al., 2016b) using SentencePiece (Kudo and Richardson, 2018). We first train a SentencePiece model with 32K vocabulary size on the original Gutenberg corpus, and apply this model to all input text to the model. This allows us to avoid `<unk>` tokens in most training and validation sets, including the W&I+L development set.

7.3 Model & Training Details

Throughout our experiments, we use two variants of the Transformer model: the "vanilla" Transformer (Vaswani et al., 2017) and the copy-augmented Transformer (Zhao et al., 2019). We use two configurations for the vanilla Transformer: a *base* model with 6 blocks of 512-2048 units with 8 attention heads, and a *large* model with 6 blocks of 1024-4096 units with 16 attention heads and pre-attention layer normalization. We only use the large model for Restricted Track (§7.5) and for the CoNLL-2014 comparison (§7.7). For the *copy*-augmented Transformer, we follow the default configuration from (Zhao et al., 2019): 6 blocks of 512-4096 units with 8 attention heads, along with an 8-head copy attention layer. For each model configuration, we train two independent models using different seeds.

Our model training is a three-stage process, as illustrated in Figure 1: DAE pre-training, training, and fine-tuning, except in Low Resource Track where there is no fine-tuning data (see Table 2). At each step, we train a model until its ERRANT score on the development set reaches convergence, and use the learned weights as initial values for the next step. In all training steps, we used the Adam (Kingma and Ba, 2015) optimizer.

Our final model is an ensemble among the different model configurations and seeds. Among the six (four for Low Resource Track) best models, we greedily search for the best combination, starting with the best-performing single model.

7.4 Post-processing

Our post-processing phase involves three steps. First, we find any `<unk>` tokens found in the original input text, and using ERRANT, we remove any edits associated with the token. Next, since many of the model's corrections can still be unnatural, if not incorrect, we re-rank candidate corrections within each sentence using a pre-trained neural LM (Dauphin et al., 2017). Specifically, we remove any combination of up to 7 edits per sentence, and choose the combination that yields the highest LM score. Finally, we noticed that, as in many previous results, our neural system performs well on some error categories (e.g., M:PUNCT) but poorly on others (e.g., R:OTHER). Because ERRANT provides a fine-grained analysis of model performance based on error types, we found it beneficial to remove edits belonging to certain categories in which the model performs too poorly. Given our final model, we randomly remove all edits from a subset of (at most N) categories for repeated steps, and choose to remove the subset of error categories that gave the highest score on the development set.

7.5 Restricted Track Results

In Table 3, we summarize our results on Restricted Track. The results illustrate that each step in our approach substantially improves upon the previous model, both on the W&I+L development and test sets. We highlight that our pre-training step with realistic human errors already gets us at a 54.82 $F_{0.5}$ score on span-based correction in ERRANT for the test set, even though we only indirectly used the W&I training set for error extraction and no other parallel corpora. This suggests that pre-training on a denoising task with realistic and common errors can already lead to a decent GEC system.

Our final ensemble model is a combination of five independent models – one base model, two large models, and two copy-augmented models – achieving 69.06 $F_{0.5}$ score on the test set.

7.6 Low Resource Track Results

In Table 4, we summarize our results on Low Resource Track. Similar to Restricted Track, each step in our approach improves upon the previous model significantly, and despite the lack of parallel data (3K for training, 1K for validation), our pre-training step already gets us at 51.71 $F_{0.5}$ score on the test set. Compared to Restricted Track, the only difference in pre-training is that the reverse dictionary for the noising function was constructed using much fewer parallel data (3K), but we see that this amount of parallel data is already enough to get within 3 points of our pre-trained model in Restricted Track.

Our final model is an ensemble of two independent models – one base model and one copy model – achieving 61.47 $F_{0.5}$ score on the test set.

Steps	W&I+L Dev				W&I+L Test			
	P	R	F0.5	Δ	P	R	F0.5	Δ
Spellcheck	59.28	5.27	**19.43**	n/a	68.77	10.55	**32.69**	n/a
+ DAE Pre-train	48.58	24.92	**40.82**	+21.39	58.33	44.20	**54.82**	+22.13
+ Train	54.30	28.67	**46.07**	+ 5.25	66.05	50.72	**62.28**	+ 7.46
+ Fine-tune	54.34	32.15	**47.75**	+ 1.68	66.02	53.41	**63.05**	+ 0.77
+ Ensemble (5)	63.54	31.48	**52.79**	+ 5.04	76.19	50.25	**69.06**	+ 6.01

Table 3: ACL 2019 BEA Workshop **Restricted Track** results. For each training step, we only list results from the model configuration that achieved the best $F_{0.5}$ test set score. All evaluation is done using ERRANT's span-based correction scorer. Pre-processing and post-processing are included in the first step and last steps, respectively.

Steps	W&I+L Dev-1K				W&I+L Test			
	P	R	F0.5	Δ	P	R	F0.5	Δ
Spellcheck	61.88	5.29	**19.72**	n/a	68.77	10.55	**32.69**	n/a
+ DAE Pre-train	46.26	19.84	**36.53**	+16.81	57.14	37.46	**51.71**	+19.02
+ Train	47.97	30.91	**43.20**	+ 6.67	58.60	47.47	**55.98**	+ 4.27
+ Ensemble (4)	58.89	26.68	**47.02**	+ 5.75	69.69	41.76	**61.47**	+ 5.49

Table 4: ACL 2019 BEA Workshop **Low Resource Track** results. For each training step, we only list results from the model configuration that achieved the best $F_{0.5}$ test set score. All evaluation is done using ERRANT's span-based correction scorer. Note that 3K examples from the W&I+Locness development set ("W&I+L Dev-3K") were used for the training step and is excluded during evaluation. Pre-processing and post-processing are included in the first step and last steps, respectively.

7.7 CoNLL-2014 Results

In Table 5, we show the performance of our approach on the CoNLL-2014 (Ng et al., 2014) dataset, with and without the newly released W&I+L dataset.[7] We also list some of the state-of-the-art[8] results prior to the shared task: copy-augmented Transformers pre-trained on random error denoising (Zhao et al., 2019), Transformers pre-trained on Wikipedia revisions and round-trip translations (Lichtarge et al., 2019), hybrid statistical and neural machine translation systems (Junczys-Dowmunt et al., 2018), and convolutional seq2seq models with quality estimation (Chollampatt and Ng, 2018b).

The results show that our approach is competitive with some of the recent state-of-the-art results that achieve around 56 MaxMatch (M^2) scores and further achieves 60+ M^2 score when the W&I+L dataset is used. This illustrates that our approach can also achieve a "near human-level performance" (Grundkiewicz and Junczys-Dowmunt, 2018). We also note that the 60.33 M^2 score was obtained by the final ensemble model from §7.5, which includes a fine-tuning step to the W&I model. This suggests that "overfitting" to the W&I dataset does not necessarily imply a reduced performance on an external dataset such as CoNLL-2014.

[7]See Appendix F for a step-by-step training progress.
[8]http://nlpprogress.com/english/grammatical_error_correction.html.

8 Analysis & Discussion

8.1 Error Analysis

Here, we give an analysis of our model's performance on some of the major ERRANT error categories on the W&I test set. Detailed information is available in Tabel 10. We observe that our model performs well on syntax relevant error types, i.e., subject-verb agreement (VERB:SVA) (84.09 $F_{0.5}$), noun numbers (NOUN:NUM) (72.19), and prepositions (PREP) (64.27), all of which are included as part of our type-based error generation in the pre-training data (§4.2). Our model also achieves 77.26 on spelling errors (SPELL) and 75.83 on orthographic errors (ORTH), both of which are improvements made mostly by our context-aware neural spellchecker. Our model also achieves 77.86 on punctuation errors (PUNCT), which happen to be the most common error category in the

Models	Pre-training	W&I+L	CoNLL-2014		
			P	R	F0.5
Our Models					
Transformers (Vanilla + Copy-Aug.)	DAE with Realistic Errors	No	71.11	32.56	**57.50**
		Yes	74.76	34.05	**60.33**
Previous Results					
Copy-Aug. Transformers	DAE with Random Errors	No	71.57	38.65	**61.15**
Transformers	Revisions + Round-Trip Translations	No	66.70	43.90	60.40
ConvS2S + QE	LM (Decoder Pre-training)	No	n/a	n/a	56.52
SMT + BiGRU	LM (Ensemble Decoding)	No	66.77	34.49	56.25

Table 5: Results on CoNLL-2014 as point of comparison. "W&I+L" indicates whether the approach made use of the (newly released) W&I+L dataset. Evaluation is done using the MaxMatch (M^2) scorer, rather than ERRANT. Pre-processing & post-processing are included before the first step and after the last step, respectively. See §7.7 for details and references.

Step	Ours	Random	Δ
DAE	**54.82**	32.25	+22.57
+ Train	**62.28**	57.00	+ 5.28
+ Fine-tune	**63.05**	60.22	+ 2.83

Table 6: Comparison of realistic and random error generation on Restricted Track. Δ means the difference between **Ours** and **Random**.

Step	Ours	Random	Δ
DAE	**51.71**	32.01	+19.70
+ Train	**55.98**	35.44	+20.54

Table 7: Comparison of realistic and random error generation on Low Resource Track. Δ means the difference between **Ours** and **Random**.

W&I+L dataset. This may be due to both our use of extracted errors from the W&I dataset during pre-training and our fine-tuning step. Finally, we find it challenging to match human annotators' "naturalness" edits, such as VERB (26.76), NOUN (41.67), and OTHER (36.53). This is possibly due to the variability in annotation styles and a lack of large training data with multiple human annotations.

8.2 Effect of Realistic Error Generation

To see how effective our realistic error based pre-training is, we compare it with (Zhao et al., 2019)'s method. According to them, random insertion, deletion, and substitution occur with the probability of 0.1 at every word, and words are reordered with a certain probability. As seen in Table 6 and 7, our pre-training method outperforms the random based one in both Restricted and Low Resource Tracks by 22.57 and 19.70, respectively. And it remains true for each step of the following transfer learning. The performance gap, however, decreases to 5.3 after training and to 3.2 after fine-tuning in Restricted Track. On the other hand, the gap in Low Resource Track slightly increases to 20.54 after training. This leads to the conclusion that our pre-training functions as proxy for training, for our generated errors resemble the human errors in the training data more than the random errors do.

8.3 Effect of Context-Aware Spellchecking

We further investigate the effects of incorporating context and fixing casing errors to the off-the-shelf `hunspell`, which we consider as a baseline. We test three spellchecker variants: `hunspell`, `hunspell` using a neural LM, and our final spellchecker model.

On the original W&I+L test set, our LM-based approach improves upon the ERRANT F0.5 score by 5.07 points, and fixing casing issues further improves this score by 4.02 points. As a result, we obtain 32.69 F0.5 score just by applying our context-aware spellchecker model.

9 Conclusion & Future Work

We introduced a neural GEC system that leverages pre-training using realistic errors, sequential transfer learning, and context-aware spellchecking with a neural LM. Our system achieved competitive results on the newly released W&I+L dataset in both standard and low-resource settings.

Spellchecker	W&I+L Test			
	P	**R**	**F0.5**	**Δ**
Hunspell	53.59	7.29	**23.60**	n/a
Hunspell + LM	65.14	8.85	**28.67**	+5.07
Ours	68.77	10.55	**32.69**	+4.02

Table 8: Effect of incorporating context into a standard spellchecker.

There are several interesting future directions following our work. One is to extend sentence-level GEC systems to multi-sentence contexts, for example by including the previous sentence, to better cope with complex semantic errors such as collocation. Because the W&I+L dataset is also a collection of (multi-)paragraph essays, adding multi-sentence contexts can improve these GEC systems. Also, to better understand the role of several components existing in modern GEC systems, it is important to examine which components are more necessary than others.

References

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. arXiv preprint arXiv:1607.06450.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In ICLR.

Chris Brockett, William B Dolan, and Michael Gamon. 2006. Correcting esl errors using phrasal smt techniques. In Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics, pages 249–256. Association for Computational Linguistics.

Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe. 2019. The BEA-2019 Shared Task on Grammatical Error Correction. In Proceedings of the 14th Workshop on Innovative Use of NLP for Building Educational Applications. Association for Computational Linguistics.

Christopher Bryant, Mariano Felice, and Edward John Briscoe. 2017. Automatic annotation and evaluation of error types for grammatical error correction. In ACL.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In EMNLP.

Shamil Chollampatt and Hwee Tou Ng. 2017. Connecting the dots: Towards human-level grammatical error correction. In Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications, pages 327–333.

Shamil Chollampatt and Hwee Tou Ng. 2018a. A multilayer convolutional encoder-decoder neural network for grammatical error correction. In Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018, pages 5755–5762.

Shamil Chollampatt and Hwee Tou Ng. 2018b. Neural quality estimation of grammatical error correction. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 2528–2539.

Daniel Dahlmeier, Hwee Tou Ng, and Siew Mei Wu. 2013. 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, pages 22–31.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 933–941, International Convention Centre, Sydney, Australia. PMLR.

Mariano Felice and Zheng Yuan. 2014. Generating artificial errors for grammatical error correction. In Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics, pages 116–126, Gothenburg, Sweden. Association for Computational Linguistics.

Michael Flor and Yoko Futagi. 2012. On using context for automatic correction of non-word misspellings in student essays. In Proceedings of the seventh workshop on building educational applications Using NLP, pages 105–115. Association for Computational Linguistics.

Tao Ge, Furu Wei, and Ming Zhou. 2018a. Fluency boost learning and inference for neural grammatical error correction. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), volume 1, pages 1055–1065.

Tao Ge, Furu Wei, and Ming Zhou. 2018b. Reaching human-level performance in automatic grammatical error correction: An empirical study. arXiv preprint arXiv:1807.01270.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. 2017. Convolutional sequence to sequence learning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1243–1252. JMLR. org.

Roman Grundkiewicz and Marcin Junczys-Dowmunt. 2018. Near human-level performance in grammatical error correction with hybrid machine translation. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers), volume 2, pages 284–290.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1631–1640, Berlin, Germany. Association for Computational Linguistics.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Identity mappings in deep residual networks. In European conference on computer vision, pages 630–645. Springer.

Matthew Honnibal and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Robin Jia and Percy Liang. 2016. Data recombination for neural semantic parsing. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 12–22, Berlin, Germany. Association for Computational Linguistics.

Marcin Junczys-Dowmunt and Roman Grundkiewicz. 2016. Phrase-based machine translation is state-of-the-art for automatic grammatical error correction. In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1546–1556, Austin, Texas. Association for Computational Linguistics.

Marcin Junczys-Dowmunt, Roman Grundkiewicz, Shubha Guha, and Kenneth Heafield. 2018. 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, Volume 1 (Long Papers), volume 1, pages 595–606.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. ICLR.

Taku Kudo and John Richardson. 2018. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. arXiv preprint arXiv:1808.06226.

Shibamouli Lahiri. 2014. Complexity of Word Collocation Networks: A Preliminary Structural Analysis. In Proceedings of the Student Research Workshop at the 14th Conference of the European Chapter of the Association for Computational Linguistics, pages 96–105, Gothenburg, Sweden. Association for Computational Linguistics.

Guillaume Lample, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018a. Unsupervised machine translation using monolingual corpora only. In International Conference on Learning Representations.

Guillaume Lample, Myle Ott, Alexis Conneau, Ludovic Denoyer, and Marc'Aurelio Ranzato. 2018b. Phrase-based & neural unsupervised machine translation. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 5039–5049, Brussels, Belgium. Association for Computational Linguistics.

Jared Lichtarge, Chris Alberti, Shankar Kumar, Noam Shazeer, Niki Parmar, and Simon Tong. 2019. Corpora generation for grammatical error correction. arXiv preprint arXiv:1904.05780.

Jared Lichtarge, Christopher Alberti, Shankar Kumar, Noam Shazeer, and Niki Parmar. 2018. Weakly supervised grammatical error correction using iterative decoding. arXiv preprint arXiv:1811.01710.

Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In EMNLP.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. arXiv preprint arXiv:1609.07843.

Tomoya Mizumoto, Mamoru Komachi, Masaaki Nagata, and Yuji Matsumoto. 2011. 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, pages 147–155.

Hwee Tou Ng, Siew Mei Wu, Ted Briscoe, Christian Hadiwinoto, Raymond Hendy Susanto, and Christopher Bryant. 2014. The conll-2014 shared task on grammatical error correction. In Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task, pages 1–14.

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. arXiv preprint arXiv:1904.01038.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. In NIPS-W.

Sebastian Ruder. 2019. Neural Transfer Learning for Natural Language Processing. Ph.D. thesis, NATIONAL UNIVERSITY OF IRELAND, GALWAY.

Keisuke Sakaguchi, Matt Post, and Benjamin Van Durme. 2017. Grammatical error correction with neural reinforcement learning. In Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 2: Short Papers), volume 2, pages 366–372.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1073–1083, Vancouver, Canada. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016a. Improving neural machine translation models with monolingual data. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 86–96, Berlin, Germany. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016b. Neural machine translation of rare words with subword units. In ACL.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In NIPS.

Toshikazu Tajiri, Mamoru Komachi, and Yuji Matsumoto. 2012. Tense and aspect error correction for esl learners using global context. In Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2, pages 198–202. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Llion Jones, Jakob Uszkoreit, Aidan N Gomez, and Ł ukasz Kaiser. 2017. Attention is all you need. In NIPS.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. 2008. Extracting and composing robust features with denoising autoencoders. In Proceedings of the 25th international conference on Machine learning, pages 1096–1103. ACM.

Dong Wang and Thomas Fang Zheng. 2015. Transfer learning for speech and language processing. In 2015 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA), pages 1225–1237. IEEE.

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

Helen Yannakoudakis, Øistein E Andersen, Ardeshir Geranpayeh, Ted Briscoe, and Diane Nicholls. 2018. Developing an automated writing placement system for esl learners. Applied Measurement in Education, 31(3):251–267.

Wei Zhao, Liang Wang, Kewei Shen, Ruoyu Jia, and Jingming Liu. 2019. Improving grammatical error correction via pre-training a copy-augmented architecture with unlabeled data. In NAACL.

A Copy-Augmented Transformers: Formal Derivation

Copy-augmented Transformers (Zhao et al., 2019) incorporate an attention-based copying mechanism (Gu et al., 2016; See et al., 2017; Jia and Liang, 2016) in the decoder of Transformers. For each output token y_t at output position t, given source token sequence $\mathbf{x} = (x_1, \ldots, x_{T'})$, the output probability distribution over token vocabulary V is defined as:

$$\mathbf{H}^{\text{enc}} = \text{enc}(\mathbf{x}) \tag{3}$$

$$\mathbf{h}_t^{\text{dec}} = \text{dec}\left(y_{1:t-1}; \mathbf{H}^{\text{enc}}\right) \tag{4}$$

$$p^{\text{gen}}(y_t \mid y_{1:t-1}; \mathbf{x}) = \text{softmax}\left(\mathbf{W}^{\text{gen}}\mathbf{h}_t^{\text{dec}}\right) \tag{5}$$

where enc denotes the encoder that maps the source token sequence $\mathbf{x}$ to a sequence of hidden vectors $\mathbf{H}^{\text{enc}} \in \mathbb{R}^{d \times T'}$, dec denotes the decoder that takes output tokens at previous time steps along with encoded embeddings and produces a hidden vector $\mathbf{h}_t^{\text{dec}} \in \mathbb{R}^d$, and $\mathbf{W}^{\text{gen}} \in \mathrm{R}^{|V| \times d}$ is a learnable linear output layer that maps the hidden vector to a pre-softmax output probabilities ("logits"). We denote the resulting distribution as the (token) generative distribution, denoted as p^{gen}.

A copy attention layer can be defined as an additional (possibly multi-head) attention layer between the encoder outputs and the final-layer hidden vector at the current decoding step. The attention layer yields two outcomes, the layer output $\mathbf{o}_t$ and the corresponding attention scores $\mathbf{s}_t$:

$$\mathbf{s}_t = \text{softmax}\left(\frac{(\mathbf{h}_t^{\text{dec}})^T \mathbf{H}^{\text{enc}}}{\sqrt{d}}\right) \tag{6}$$

$$\mathbf{o}_t = \mathbf{H}^{\text{enc}} \mathbf{s}_t \tag{7}$$

The copy distribution is then defined as the attention scores in (6) themselves[9]:

$$p^{\text{copy}}(y_t \mid y_{1:t-1}; \mathbf{x}) = \mathbf{s}_t \tag{8}$$

The final output of a copy-augmented Transformer as a mixture of both generative and copy distributions. The mixture weight[10] α_t^{copy} is defined at each decoding step as follows:

$$\alpha_t^{\text{copy}} = \text{sigmoid}\left((\mathbf{w}^{\text{alpha}})^T \mathbf{o}_t\right) \tag{9}$$

$$p(y_t) = (1 - \alpha_t^{\text{copy}}) \cdot p^{\text{gen}}(y_t) + \alpha_t^{\text{copy}} \cdot p^{\text{copy}}(y_t) \tag{10}$$

where $\mathbf{w}^{\text{alpha}} \in \mathbb{R}^d$ is a learnable linear output layer. (For simplicity, we omit the dependencies of all probabilities in (10) on both $y_{1:t-1}$ and $\mathbf{x}$.) The mixture weight balances between how likely it is for the model to simply copy a source token, rather than generating a possibly different token.

B Exploratory Data Analysis

B.1 Data Sizes

Figure 2 illustrates the number of available parallel corpora (counting multiple annotations) across data sources. Note that the vertical axis is capped at 100K for a better visual comparison among other sources.

For the Lang-8 dataset, we count all available (ranging from 1 to 8) annotations for each of 1.04M original sentences. Also note that we only use the subset of Lang-8 whose source and target sentences are different, leaving only 575K sentences instead of 1.11M.

B.2 Sentence Length vs. Number of Edits

Figure 3 illustrates the distribution of sentence lengths and the number of edits per sentence across different data sources.

Table 9 includes our permutation test[11] results on the number of edits per sentence, normalized by sentence length (i.e., number of word-level tokens), between training data sources. Using an approximate permutation test with 10k simulations and a significant level of $\alpha = 0.05$, we find that there is a statistical difference in the normalized edit count per sentence between the W&I training set and each of FCE, NUCLE, and Lang-8. This serves as a preliminary experiment showing how the distribution of grammatical errors can be significantly different across different sources – even when they belong to a roughly similar domain.

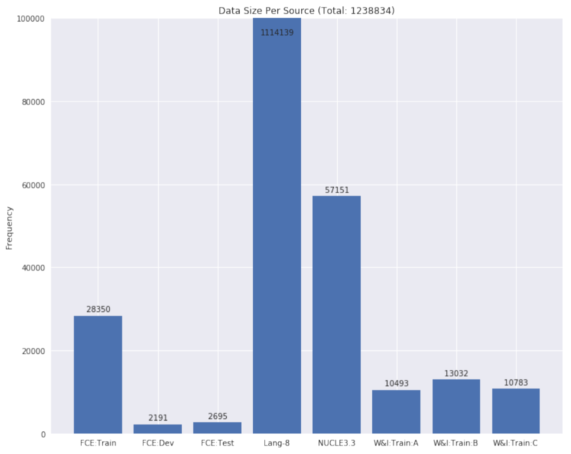

Figure 2: Data size per source for all Restricted Track training data. Number includes multiple annotations for Lang-8. Vertical axis is capped at 100K for a better visual comparison among the smaller sources. The three FCE splits (train, dev, test) are collectively used for training, and the three W&I+L splits correspond to three English proficiency levels ("A", "B", "C"). After duplicate removal, only 575K of the Lang-8 parallel corpus are actually used for training.

C Full Noising Algorithm

Algorithms 1 and 2 detail our noising scenarios.

D Results on error categories

Table 10 shows the result on error categories.

E CoNLL-2014 Full Results (Without Using W&I+L)

In Table 11, we include the training progress for our result for CoNLL-2014.

A noticeable difference between this result and our results for Restricted Track and Low Resource Track is that adaptation via fine-tuning is not necessarily effective here. We hypothesize that this is mostly due to the fact that the training subset to which we fine-tune our model (NUCLE) comes

[9]In practice, this involves adding up the copy scores defined for each source token into a $|V|$-dimensional vector, using commands such as `scatter_add()` in PyTorch.

[10]When computing the mixture weight α_t^{copy}, Zhao et al. (2019) applies a linear layer to $\mathbf{H}^{\text{enc}}\tilde{\mathbf{s}}_t$, where $\tilde{\mathbf{s}}_t$ are the attention scores in (6) *before* taking softmax. Our formulation gives essentially the same copying mechanism, while being more compatible to standard Transformer implementations.

[11]We used the off-the-shelf `mlxtend` package to run permutation tests. See `http://rasbt.github.io/mlxtend/user_guide/evaluate/permutation_test/`.

Algorithm 1 Pseudocode for constructing noise dictionary

```
function CONSTRUCTNOISEDICTIONARY(ParallelCorpus, min_count)
    Initialize a dictionary dict
    for (CorToken, OriToken) in ParallelCorpus do
        dict[CorToken] += OriToken
    end for
    for CorToken, OriTokenList in dict do
        for OriToken in OriTokenList do
            if count(OriToken) $<$ min_count then
                delete OriToken from dict[CorToken]
            end if
        end for
        if length(OriTokenList)==1 and CorToken==OriTokenList[0] then
            delete OriToken from dict
        end if
    end for
return dict
end function
```

Algorithm 2 Pseudocode for generating noisy sentences

```
function CHANGE_TYPE(word, prob)
    preposition_set = [$\emptyset$, for, to, at, $\cdots$ ]
    if random[0, 1] $>$ prob then return word
    else
        if word in preposition_set then
            random_choose_one_from(preposition_set)
        else if word is Noun then change_number(word)
        else if word is Verb then change_form(word)
        end if
    end if
return word
end function
function MAKE_NOISE(sentence, prob)
    dict = ConstructNoiseDictionary(ParallelCorpus, min_count)
    noised = []
    for word in sentence do
        if word in dict and random[0, 1] $>$ prob then
            candidates = dict[word]
            noise = random_choose_one_from(candidates)
        else
            noise = change_type(word)
        end if
        noised += noise
    end forreturn noised
end function
```

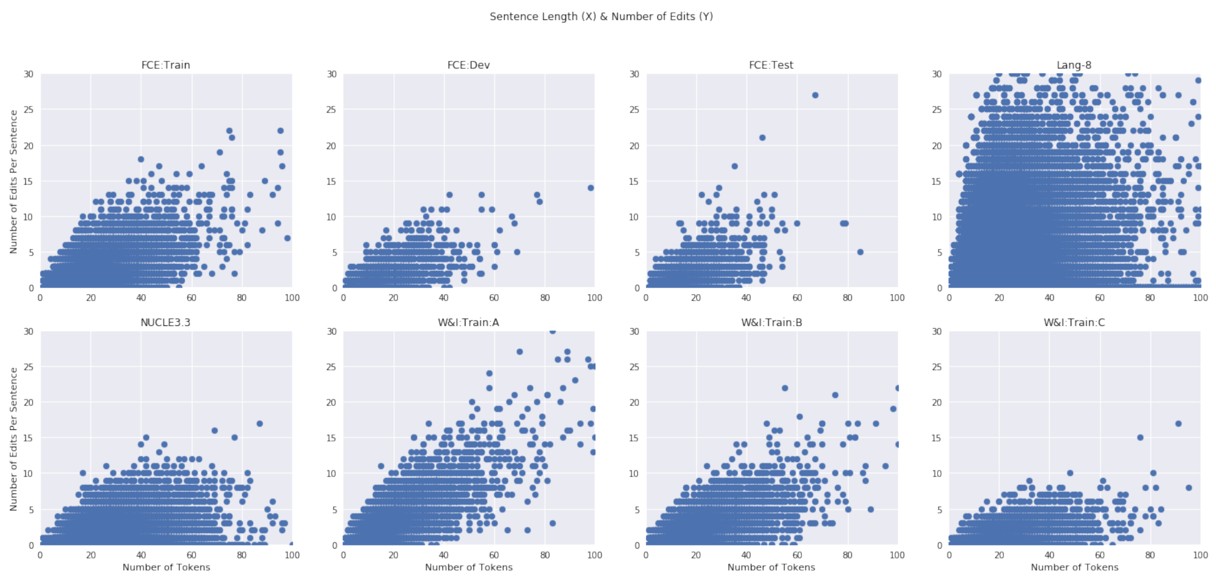

Figure 3: Sentence length versus the number of edits made in each sentence, across all training data sources for the Restricted Track. The horizontal axis is capped at 100 words (less than 0.02% of all sentences contain more than 100 words). The vertical axis is capped at 40 edits (less than 0.02% of all sentences contain more than 30 edits).

Source	**# Sent.**	**# Edits / Length**	**Perm. Test vs. W&I**
W&I-Train	34.3K	0.0922	n/a
FCE	33.2K	0.0898	$p = .002$
NUCLE	57.2K	0.0318	$p < .001$
Lang-8	1.11M	0.1357	$p < .001$

Table 9: Comparing the average number of edits per sentence, normalized by sentence length, between the W&I training set and other available training data sources for the Restricted Track. "vs. W&I" refers to the result of an approximate permutation test (10k rounds) against that in the W&I training set. Under the significance level of $\alpha = 0.05$, the number for FCE, NUCLE, and Lang-8 are all significantly different from that for the W&I training set.

from a different source than the actual test set (CoNLL-2014) – despite the fact that both datasets have similar domains (personal essays from English students), they can still have many other different characteristics, including the writer's English proficiency and annotation styles.

F Training Details

Our model training is a three-stage process: DAE pre-training, training, and fine-tuning, except in Low Resource Track where there is no fine-tuning data. At each step, we train a model until its ERRANT score on the development set reaches convergence, and use the learned weights as ini-

Error types	**P**	**R**	**F0.5**
ADJ	71.43	28.57	54.95
ADJ:FORM	100.00	40.00	76.92
ADV	70.59	22.22	49.18
CONJ	100.00	4.76	20.00
CONTR	100.00	91.67	98.21
DET	78.95	47.04	69.52
MORPH	81.18	49.29	71.88
NOUN	64.52	17.24	41.67
NOUN:INFL	100.00	41.18	77.78
NOUN:POSS	81.82	48.21	71.81
ORTH	87.38	49.60	75.83
OTHER	55.93	15.30	36.53
PART	76.19	55.17	70.80
PREP	69.69	49.01	64.27
PRON	78.67	43.70	67.82
PUNCT	79.95	70.48	77.86
SPELL	76.07	82.41	77.26
VERB	66.67	7.88	26.76
VERB:FORM	72.45	73.20	72.60
VERB:INFL	100.00	85.71	96.77
VERB:SVA	83.77	85.43	84.09
VERB:TENSE	71.43	45.64	64.18
WO	67.74	25.61	50.97

Table 10: Results on error types.

Steps	CoNLL-2014		
	P	R	F0.5
Spellcheck	54.75	5.75	**20.25**
+ Pre-train (b)	54.76	15.09	**35.89**
+ Train (b)	60.43	34.22	**52.40**
+ Fine-tune (b)	60.81	33.32	**52.20**
+ Pre-train (c)	65.81	24.17	**48.95**
+ Train (c)	61.38	30.97	**51.30**
+ Fine-tune (c)	60.82	32.50	**51.79**
+ Ensemble (b+c)	71.11	32.56	**57.50**

Table 11: Training progress on CoNLL-2014. *No W&I+Locness datasets were used in these results.* 'b' and 'c' refer to the base and copy configurations of the Transformer, respectively. Evaluation is done using the MaxMatch (M^2) scorer. Pre-processing & post-processing are included before the first step and after the last step, respectively.

tial values for the next step. For pre-training, we used a learning rate of $5 \cdot 10^{-4}$ for the base and copy-augmented Transformers and 10^{-3} for the large Transformer. For training, we reset the optimizer and set the learning rate to 10^{-4}. For fine-tuning (if available), we again reset the optimizer and set the learning rate to $5 \cdot 10^{-5}$. In all training steps, we used the Adam (Kingma and Ba, 2015) optimizer with the inverse square-root schedule and a warmup learning rate of 10^{-7}, along with a dropout rate of 0.3.

G Further Analysis

G.1 Effect of Copying Mechanisms & Ensembles

One of our contributions is to highlight the benefit of ensembling multiple models with diverse characteristics. As shown in Table 3, the final ensemble step involving different types of models was crucial for our model's performance, improving the test score by over 6 $F_{0.5}$ points. We first noticed that the copy-augmented Transformer learns to be more conservative – i.e., higher precision but lower recall given similar overall scores – in its edits than the vanilla Transformer, presumably because the model includes an inductive bias that favors copying (i.e., not editing) the input token via its copy attention scores. Table 12 shows this phenomenon for Restricted Track.

Given multiple models with diverse characteristics, the choice of models for ensemble can translate to controlling how conservative we want our final model to be. For example, combining one vanilla model with multiple independent copy-augmented models will result in a more conservative model. This could serve as an alternative to other methods that control the precision-recall ratio, such as the edit-weighted loss (Junczys-Dowmunt et al., 2018).

Model (Config.)	W&I+L Test		
	P	R	F0.5
Vanilla (Large)	63.66	56.82	62.17
Copy (Copy)	66.02	53.41	63.05
Δ	+2.36	-3.41	+0.88

Table 12: Single-model ERRANT scores for Restricted Track, using a large Transformer and a copy-augmented Transformer.