Parallel Iterative Edit Models for Local Sequence Transduction

Abhijeet Awasthi* **Sunita Sarawagi** **Rasna Goyal** **Sabyasachi Ghosh** **Vihari Piratla**
Department of Computer Science and Engineering, IIT Bombay

Abstract

We present a Parallel Iterative Edit (PIE) model for the problem of local sequence transduction arising in tasks like Grammatical error correction (GEC). Recent approaches are based on the popular encoder-decoder (ED) model for sequence to sequence learning. The ED model auto-regressively captures full dependency among output tokens but is slow due to sequential decoding. The PIE model does parallel decoding, giving up the advantage of modelling full dependency in the output, yet it achieves accuracy competitive with the ED model for four reasons: 1. predicting edits instead of tokens, 2. labeling sequences instead of generating sequences, 3. iteratively refining predictions to capture dependencies, and 4. factorizing logits over edits and their token argument to harness pre-trained language models like BERT. Experiments on tasks spanning GEC, OCR correction and spell correction demonstrate that the PIE model is an accurate and significantly faster alternative for local sequence transduction. The code and pre-trained models for GEC are available at https://github.com/awasthiabhijeet/PIE.

1 Introduction

In local sequence transduction (LST) an input sequence $x_1, \ldots, x_n$ needs to be mapped to an output sequence $y_1, \ldots, y_m$ where the $\mathbf{x}$ and $\mathbf{y}$ sequences differ only in a few positions, m is close to n, and x_i, y_j come from the same vocabulary Σ. An important application of local sequence transduction that we focus on in this paper is Grammatical error correction (GEC). We contrast local transduction with more general sequence transduction tasks like translation and paraphrasing which might entail different input-output vocabulary and non-local alignments. The general sequence transduction task is cast as sequence to sequence (seq2seq) learning and modeled popularly using an attentional encoder-decoder (ED) model. The ED model auto-regressively produces each token y_t in the output sequence conditioned on all previous tokens $y_1, \ldots, y_{t-1}$. Owing to the remarkable success of this model in challenging tasks like translation, almost all state-of-the-art neural models for GEC use it (Zhao et al., 2019; Lichtarge et al., 2019; Ge et al., 2018b; Chollampatt and Ng, 2018b; Junczys-Dowmunt et al., 2018).

We take a fresh look at local sequence transduction tasks and present a new parallel-iterative-edit (PIE) architecture. Unlike the prevalent ED model that is constrained to sequentially generating the tokens in the output, the PIE model generates the output in parallel, thereby substantially reducing the latency of sequential decoding on long inputs. However, matching the accuracy of existing ED models without the luxury of conditional generation is highly challenging. Recently, parallel models have also been explored in tasks like translation (Stern et al., 2018; Lee et al., 2018; Gu et al., 2018; Kaiser et al., 2018) and speech synthesis (van den Oord et al., 2018), but their accuracy is significantly lower than corresponding ED models. The PIE model incorporates the following four ideas to achieve comparable accuracy on tasks like GEC in spite of parallel decoding.

1. Output edits instead of tokens: First, instead of outputting tokens from a large vocabulary, we output edits such as copy, appends, deletes, replacements, and case-changes which generalize better across tokens and yield a much smaller vocabulary. Suppose in GEC we have an input sentence: `fowler fed dog`. Existing seq2seq learning approaches would need to output the four tokens `Fowler`, `fed`, `the`, `dog` from a word vocabulary whereas we

*Correspondence to: awasthi@cse.iitb.ac.in

would predict the edits {`Capitalize token 1, Append(the) to token 2, Copy token 3`}.

2. Sequence labeling instead of sequence generation: Second we perform in-place edits on the source tokens and formulate local sequence transduction as *labeling* the input tokens with edit commands, rather than solving the much harder whole *sequence generation* task involving a separate decoder and attention module. Since input and output lengths are different in general such formulation is non-trivial, particularly due to edits that insert words. We create special compounds edits that merge token inserts with preceding edits that yield higher accuracy than earlier methods of independently predicting inserts (Ribeiro et al., 2018; Lee et al., 2018).

3. Iterative refinement: Third, we increase the inference capacity of the parallel model by iteratively inputting the model's own output for further refinement. This handles dependencies implicitly, in a way reminiscent of Iterative Conditional Modes (ICM) fitting in graphical model inference (Koller and Friedman, 2009). Lichtarge et al. (2019) and Ge et al. (2018b) also refine iteratively but with ED models.

4. Factorize pre-trained bidirectional LMs: Finally, we adapt recent pre-trained bidirectional models like BERT (Devlin et al., 2018) by factorizing the logit layer over edit commands and their token argument. Existing GEC systems typically rely on conventional forward directional LM to pretrain their decoder, whereas we show how to use a bi-directional LM in the encoder, and that too to predict edits.

Novel contributions of our work are as follows:

- Recognizing GEC as a local sequence transduction (LST) problem, rather than machine translation. We then cast LST as a fast non-autoregressive, sequence labeling model as against existing auto-regressive encoder-decoder model.
- Our method of reducing LST to non-autoregressive sequence labeling has many novel elements: outputting edit operations instead of tokens, append operations instead of insertions in the edit space, and replacements along with custom transformations.
- We show how to effectively harness a pre-trained language model like BERT using our factorized logit architecture with edit-specific attention masks.
- The parallel inference in PIE is 5 to 15 times faster than a competitive ED based GEC model like (Lichtarge et al., 2019) which performs sequential decoding using beam-search. PIE also attains close to state of the art performance on standard GEC datasets. On two other local transduction tasks, viz., OCR and spell corrections the PIE model is fast and accurate w.r.t. other existing models developed specifically for local sequence transduction.

2 Our Method

We assume a fully supervised training setup where we are given a parallel dataset of incorrect, correct sequence pairs: $D = \{(\mathbf{x}^i, \mathbf{y}^i) : i = 1 \dots N\}$, and an optional large corpus of unmatched correct sequences $\mathcal{L} = \{\tilde{\mathbf{y}}^1, \dots, \tilde{\mathbf{y}}^U\}$. In GEC, this could be a corpus of grammatically correct sentences used to pre-train a language model.

Background: existing ED model Existing seq2seq ED models factorize $\Pr(\mathbf{y}|\mathbf{x})$ to capture the full dependency between a y_t and all previous $\mathbf{y}_{<t} = y_1, \dots, y_{t-1}$ as $\prod_{t=1}^{m} \Pr(y_t|\mathbf{y}_{<t}, \mathbf{x})$. An encoder converts input tokens $x_1, \dots, x_n$ to contextual states $\mathbf{h}_1, \dots, \mathbf{h}_n$ and a decoder summarizes $\mathbf{y}_{<t}$ to a state $\mathbf{s}_t$. An attention distribution over contextual states computed from $\mathbf{s}_t$ determines the relevant input context $\mathbf{c}_t$ and the output token distribution is calculated as $\Pr(y_t|\mathbf{y}_{<t}, \mathbf{x}) = \Pr(y_t|\mathbf{c}_t, \mathbf{s}_t)$. Decoding is done sequentially using beam-search. When a correct sequence corpus $\mathcal{L}$ is available, the decoder is pre-trained on a next-token prediction loss and/or a trained LM is used to re-rank the beam-search outputs (Zhao et al., 2019; Chollampatt and Ng, 2018a; Junczys-Dowmunt et al., 2018).

2.1 Overview of the PIE model

We move from generating tokens in the output sequence $\mathbf{y}$ using a separate decoder, to labelling the input sequence $x_1, \dots, x_n$ with edits $e_1, \dots, e_n$. For this we need to design a function Seq2Edits that takes as input an $(\mathbf{x}, \mathbf{y})$ pair in D and outputs a sequence $\mathbf{e}$ of edits from an edit space $\mathcal{E}$ where $\mathbf{e}$ is of the same length as $\mathbf{x}$ in spite of $\mathbf{x}$ and $\mathbf{y}$ being

of different lengths. In Section 2.2 we show how we design such a function.

Training: We invoke Seq2Edits on D and learn the parameters of a probabilistic model $\Pr(\mathbf{e}|\mathbf{x}, \theta)$ to assign a distribution over the edit labels on tokens of the input sequence. In Section 2.3 we describe the PIE architecture in more detail. The correct corpus $\mathcal{L}$, when available, is used to pre-train the encoder to predict an arbitrarily masked token y_t in a sequence $\mathbf{y}$ in $\mathcal{L}$, much like in BERT. Unlike in existing seq2seq systems where $\mathcal{L}$ is used to pre-train the decoder that only captures forward dependencies, in our pre-training the predicted token y_t is dependent on both forward and backward contexts. This is particularly useful for GEC-type tasks where future context $y_{t+1} \dots y_m$ can be approximated by $x_{t+1}, \dots, x_n$.

Inference: Given an input $\mathbf{x}$, the trained model predicts the edit distribution for each input token independent of others, that is $\Pr(\mathbf{e}|\mathbf{x}, \theta) = \prod_{t=1}^{n} \Pr(e_t|\mathbf{x}, t, \theta)$, and thus does not entail the latency of sequential token generation of the ED model. We output the most probable edits $\hat{\mathbf{e}} = \text{argmax}_{\mathbf{e}} \Pr(\mathbf{e}|\mathbf{x}, \theta)$. Edits are designed so that we can easily get the edited sequence $\hat{\mathbf{y}}$ after *applying* $\hat{\mathbf{e}}$ on $\mathbf{x}$. $\hat{\mathbf{y}}$ is further refined by iteratively applying the model on the generated outputs $\hat{\mathbf{y}}$ until we get a sequence identical to one of the previous sequences upto a maximum number of iterations I.

2.2 The Seq2Edits Function

Given a $\mathbf{x} = x_1, \dots, x_n$ and $\mathbf{y} = y_1, \dots, y_m$ where m may not be equal to n, our goal is to obtain a sequence of edit operations $\mathbf{e} = (e_1, \dots, e_n) : e_i \in \mathcal{E}$ such that applying edit e_i on the input token x_i at each position i reconstructs the output sequence $\mathbf{y}$. Invoking an off-the-shelf edit distance algorithm between $\mathbf{x}$ and $\mathbf{y}$ can give us a sequence of copy, delete, replace, and insert operations of arbitrary length. The main difficulty is converting the insert operations into in-place edits at each x_i. Other parallel models (Ribeiro et al., 2018; Lee et al., 2018) have used methods like predicting insertion slots in a pre-processing step, or predicting zero or more tokens in-between any two tokens in $\mathbf{x}$. We will see in Section 3.1.4 that these options do not perform well. Hence we design an alternative edit space $\mathcal{E}$ that merges inserts with preceding edit operations creating compound append or replace operations. Further, we create a dictionary Σ_a of common q-gram insertions or replacements observed in the training data.

Require: $\mathbf{x} = (x_1, \dots, x_n)$, $\mathbf{y} = (y_1, \dots, y_m)$ and $\mathcal{T}$: List of Transformations
  diffs $\leftarrow$ LEVENSHTEIN-DIST$(\mathbf{x}, \mathbf{y})$ with modified cost.
  diffs $\leftarrow$ In diffs break substitutions, merge inserts into q-grams
  $\Sigma_a \leftarrow M$ most common inserts in training data
  $\mathbf{e} \leftarrow$ EMPTYARRAY(n)
  for $t \leftarrow 1$ to LENGTH(diffs) **do**
    if diffs$[t] = (\text{C}, x_i)$ or (D, x_i) **then**
      $e_i \leftarrow$ diffs$[t].op$
    else if diffs$[t] = (\text{I}, x_i, w)$ **then**
      if $e_i = \text{D}$ **then**
        if (x_i, w) match transformation $\text{T} \in \mathcal{T}$ **then**
          $e_i \leftarrow \text{T}$
        else
          $e_i \leftarrow \text{R}(w)$ **if** $w \in \Sigma_a$ else C
      else if $e_i = \text{C}$ **then**
        $e_i \leftarrow \text{A}(w)$ **if** $w \in \Sigma_a$ else C
  return $\mathbf{e}$

Figure 1: The Seq2Edits function used to convert a sequence pair $\mathbf{x}, \mathbf{y}$ into in-place edits.

Our edit space ($\mathcal{E}$) comprises of copy (C) x_i, delete (D) x_i, append (A) a q-gram $w \in \Sigma_a$ after copying x_i, replace (R) x_i with a q-gram $w \in \Sigma_a$. For GEC, we additionally use transformations denoted as $\text{T}_1, \dots, \text{T}_k$ which perform word-inflection (e.g. `arrive` to `arrival`). The space of all edits is thus:

$$\mathcal{E} = \{\text{C}, \text{D}, \text{T}_1, \dots, \text{T}_k\} \cup \{\text{A}(w) : w \in \Sigma_a\} \cup \{\text{R}(w) : w \in \Sigma_a\} \quad (1)$$

	Example 1
$\mathbf{x}$	[Bolt can have run race]
$\mathbf{y}$	[Bolt could have run the race]
diff	(C,[) (C,Bolt) (D,can) (I,can,could) (C,have) (C,run) (I,run,the) (C,race) (C,])
$\mathbf{e}$	C C R(could) C A(the) C C ↑ ↑ ↑ ↑ ↑ ↑ ↑ [Bolt can have run race]
	Example 2
$\mathbf{x}$	[He still won race !]
$\mathbf{y}$	[However , he still won !]
diff	(C,[) (I,[,However,) (D,He) (I,He,he) (C,still) (C,won) (D,-race) (C,!) (C,])
$\mathbf{e}$	A(However,) T_case C C D C C ↑ ↑ ↑ ↑ ↑ ↑↑ [He still won race !]

Table 1: Two example sentence pairs converted into respective edit sequences.

We present our algorithm for converting a sequence $\mathbf{x}$ and $\mathbf{y}$ into in-place edits on $\mathbf{x}$ using the above edit space in Figure 1. Table 1 gives examples of converting ($\mathbf{x}$, $\mathbf{y}$) pairs to edit sequences.

We first invoke the Levenshtein distance algorithm (Levenshtein, 1966) to obtain `diff` between $\mathbf{x}$ and $\mathbf{y}$ with delete and insert cost as 1 as usual, but with a modified substitution cost to favor matching of related words. We detail this modified cost in the Appendix and show an example of how this modification leads to more sensible edits. The `diff` is post-processed to convert substitutions into deletes followed by inserts, and consecutive inserts are merged into a q-gram. We then create a dictionary Σ_a of the M most frequent q-gram inserts in the training set. Thereafter, we scan the `diff` left to right: a copy at x_i makes e_i = C, a delete at x_i makes e_i = D, an insert w at x_i and a e_i = C flips the e_i into a e_i = A(w) if w is in Σ_a, else it is dropped, an insert w at x_i and a e_i = D flips the e_i into a e_i = T(w) if a match found, else a replace R(w) if w is in Σ_a, else it is dropped.

The above algorithm does not guarantee that when $\mathbf{e}$ is applied on $\mathbf{x}$ we will recover $\mathbf{y}$ for all sequences in the training data. This is because we limit Σ_a to include only the M most frequently inserted q-grams. For local sequence transduction tasks, we expect a long chain of consecutive inserts to be rare, hence our experiments were performed with $q = 2$. For example, in NUCLE dataset (Ng et al., 2014) which has roughly 57.1K sentences, less than 3% sentences have three or more consecutive inserts.

2.3 The Parallel Edit Prediction Model

We next describe our model for predicting edits $\mathbf{e} : e_1, \ldots, e_n$ on an input sequence $\mathbf{x} : x_1, \ldots, x_n$. We use a bidirectional encoder to provide a contextual encoding of each x_i. This can either be multiple layers of bidirectional RNNs, CNNs or deep bidirectional transformers. We adopt the deep bidirectional transformer architecture since it encodes the input in parallel. We pre-train the model using $\mathcal{L}$ much like in BERT pre-training recently proposed for language modeling(Devlin et al., 2018). We first give an overview of BERT and then describe our model.

Background: BERT The input is token embedding $\mathbf{x}_i$ and positional embedding $\mathbf{p}_i$ for each token x_i in the sequence $\mathbf{x}$. Call these jointly as $\mathbf{h}_i^0 = [\mathbf{x}_i, \mathbf{p}_i]$. Each layer ℓ produces a $\mathbf{h}_i^\ell$ at each position i as a function of $\mathbf{h}_i^{\ell-1}$ and self-attention over all $\mathbf{h}_j^{\ell-1}$, $j \in [1, n]$. The BERT model is pre-trained by masking out a small fraction of the input tokens by a special MASK, and predicting the masked word from its bidirectional context captured as the last layer output: $\mathbf{h}_1, \ldots, \mathbf{h}_n$.

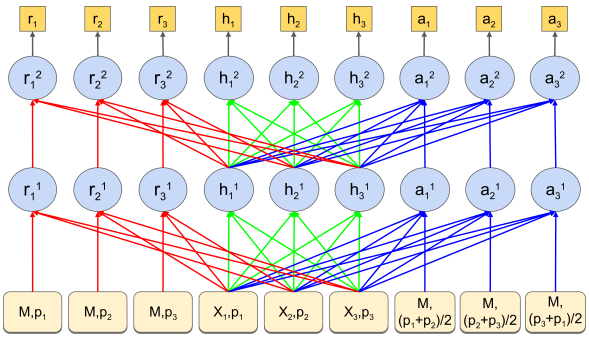

Figure 2: A Parallel Edit Architecture based on a 2-layer bidirectional transformer. Input sequence length is 3. Arrows indicate attention mask for computation of $\mathbf{h}_i^l, \mathbf{r}_i^l, \mathbf{a}_i^l$ at position i for layer l.

2.3.1 A Default use of BERT

Since we have cast our task as a sequence labeling task, a default output layer would be to compute $\Pr(e_i|\mathbf{x})$ as a softmax over the space of edits $\mathcal{E}$ from each $\mathbf{h}_i$. If W_e denotes the softmax parameter for edit e, we get:

$$\Pr(e_i = e|\mathbf{x}) = \text{softmax}(W_e^\intercal \mathbf{h}_i) \qquad (2)$$

The softmax parameters, W_e in Equation 2, have to be trained from scratch. We propose a method to exploit token embeddings of the pre-trained language model to warm-start the training of edits like appends and replaces which are associated with a token argument. Furthermore, for appends and replaces, we provide a new method of computing the hidden layer output via alternative input positional embeddings and self-attention. We do so without introducing any new parameters in the hidden layers of BERT.

2.3.2 An Edit-factorized BERT Architecture

We adapt a pre-trained BERT-like bidirectional language model to learn to predict edits as follows. For suitably capturing the contexts for replace edits, for each position i we create an additional input comprising of $\mathbf{r}_i^0 = [\text{M}, \mathbf{p}_i]$ where M is the embedding for MASK token in the LM. Likewise for a potential insert between i and $i+1$ we create an input $\mathbf{a}_i^0 = [\text{M}, \frac{\mathbf{p}_i + \mathbf{p}_{i+1}}{2}]$ where the second component is the average of the position embedding of the ith and $i+1$th position. As shown in Figure 2, at each layer ℓ we compute self-attention for the ith replace unit $\mathbf{r}_i^\ell$ over $\mathbf{h}_j^\ell$ for all $j \neq i$ and itself. Likewise, for the append unit $\mathbf{a}_i^\ell$ the self-attention

is over all $\mathbf{h}_j^\ell$ for all js and itself. At the last layer we have $\mathbf{h}_1, \ldots, \mathbf{h}_n, \mathbf{r}_1, \ldots, \mathbf{r}_n, \mathbf{a}_1, \ldots, \mathbf{a}_n$. Using these we compute logits factorized over edits and their token argument. For an edit $e \in \mathcal{E}$ at position i, let w denote the argument of the edit (if any). As mentioned earlier, w can be a q-gram for append and replace edits. Embedding of w, represented by $\phi(w)$ is obtained by summing up individual output embeddings of tokens in w. Additionally, in the outer layer we allocate edit-specific parameters $\boldsymbol{\theta}$ corresponding to each distinct command in $\mathcal{E}$. Using these, various edit logits are computed as follows:

$$\Pr(e_i|\mathbf{x}) = \text{softmax}(\text{logit}(e_i|\mathbf{x})) \quad \text{where}$$
$$\text{logit}(e_i|\mathbf{x}) = \tag{3}$$
$$\begin{cases} \theta_{\text{C}}^\intercal \mathbf{h}_i + \phi(x_i)^\intercal \mathbf{h}_i + 0 & \text{if } e_i = \text{C} \\ \theta_{\text{A}(w)}^\intercal \mathbf{h}_i + \phi(x_i)^\intercal \mathbf{h}_i + \phi(w)^\intercal \mathbf{a}_i & \text{if } e_i = \text{A}(w) \\ \theta_{\text{R}(w)}^\intercal \mathbf{h}_i + 0 + (\phi(w) - \phi(x_i))^\intercal \mathbf{r}_i & \text{if } e_i = \text{R}(w) \\ \theta_{\text{D}}^\intercal \mathbf{h}_i + 0 + 0 & \text{if } e_i = \text{D} \\ \theta_{T_k}^\intercal \mathbf{h}_i + \phi(x_i)^\intercal \mathbf{h}_i + 0 & \text{if } e_i = T_k \end{cases}$$

The first term in the RHS of above equations captures edit specific score. The second term captures the score for copying the current word x_i to the output. The third term models the influence of a new incoming token in the output obtained by a replace or append edit. For replace edits, score of the replaced word is subtracted from score of the incoming word. For transformation we add the copy score because they typically modify only the word forms, hence we do not expect meaning of the transformed word to change significantly.

The above equation provides insights on why predicting independent edits is easier than predicting independent tokens. Consider the append edit ($\text{A}(w)$). Instead of independently predicting x_i at i and w at $i+1$, we jointly predict the tokens in these two slots and contrast it with not inserting any new w after x_i in a single softmax. We will show empirically (Sec 3.1.4) that such selective joint prediction is key to obtaining high accuracy in spite of parallel decoding.

Finally, loss for a training example $(\mathbf{e}, \mathbf{x})$ is obtained by summing up the cross-entropy associated with predicting edit e_i at each token x_i.

$$\mathcal{L}(\mathbf{e}, \mathbf{x}) = -\Sigma_i \log(\Pr(e_i|\mathbf{x})) \tag{4}$$

3 Experiments

We compare our parallel iterative edit (PIE) model with state-of-the-art GEC models that are all based on attentional encoder-decoder architectures. In Section 3.2 we show that the PIE model is also effective on two other local sequence transduction tasks: spell and OCR corrections. Hyperparameters for all the experiments are provided in Table 12, 13 of the Appendix.

3.1 Grammatical error correction (GEC)

We use Lang-8 (Mizumoto et al., 2011), NUCLE (Ng et al., 2014) and FCE (Yannakoudakis et al., 2011) corpora, which jointly comprise 1.2 million sentence pairs in English. The validation dataset comprises of 1381 sentence pairs from CoNLL-13 (Ng et al., 2013) test set. We initialize our GEC model with the publicly available BERT-LARGE[1] model that was pre-trained on Wikipedia (2500M words) and Book corpus (Zhu et al., 2015) (800M words) to predict 15% randomly masked words using its deep bidirectional context. Next, we perform 2 epochs of training on a synthetically perturbed version of the One-Billion-word corpus (Chelba et al., 2013). We refer to this as synthetic training. Details of how we create the synthetic corpus appear in Section A.3 of the appendix. Finally we fine-tune on the real GEC training corpus for 2 epochs. We use a batch size of 64 and learning rate 2e-5.

The edit space consists of copy, delete, 1000 appends, 1000 replaces and 29 transformations and their inverse. Arguments of Append and Replace operations mostly comprise punctuations, articles, pronouns, prepositions, conjunctions and verbs. Transformations perform inflections like add suffix `s`, `d`, `es`, `ing`, `ed` or replace suffix `s to ing`, `d to s`, etc. These transformations were chosen out of common replaces in the training data such that many replace edits map to only a few transformation edits, in order to help the model better generalize replaces across different words. In Section 3.1.4 we see that transformations increase the model's recall. The complete list of transformations appears in Table 10 of the Appendix. We evaluate on $F_{0.5}$ score over span-level corrections from the MaxMatch (M2) scorer (Dahlmeier and Ng, 2012) on CONLL-2014-test. Like most existing GEC systems, we invoke a spell-checker on the test sentences before applying our model. We also report GLEU$^+$ (Napoles et al., 2016) scores on JFLEG corpus (Napoles

[1] https://github.com/google-research/bert

et al., 2017) to evaluate fluency.

Work	CONLL-14			JFLEG
	P	R	$F_{0.5}$	GLEU$^+$
Zhao et al. (2019)	67.7	40.6	59.8	-
Lichtarge et al. (2019)	65.5	37.1	56.8	61.6
Chollampatt and Ng (2018a)	69.9	23.7	46.4	51.3
PIE (This work)	66.1	43.0	59.7	60.3

Table 2: Comparison of non-ensemble model numbers for various GEC models. Best non-ensemble results are reported for each model.

3.1.1 Overall Results

Table 3 compares ensemble model results of PIE and other state of the art models, which all happen to be seq2seq ED models and also use ensemble decoding. For PIE, we simply average the probability distribution over edits from 5 independent ensembles. In Table-2 we compare non-ensemble numbers of PIE with the best available non-ensemble numbers of competing methods. On CoNLL-14 test-set our results are very close to the highest reported by Zhao et al. (2019). These results show that our parallel prediction model is competitive without incurring the overheads of beam-search and slow decoding of sequential models. GLEU$^+$ score, that rewards fluency, is somewhat lower for our model on the JFLEG test set because of parallel predictions. We do not finetune our model on the JFLEG dev set. We expect these to improve with re-ranking using a LM. All subsequent ablations and timing measurements are reported for non-ensemble models.

3.1.2 Running Time Comparison

Parallel decoding enables PIE models to be considerably faster than ED models. In Figure 3 we compare wall-clock decoding time of PIE with 24 encoder layers (PIE-LARGE, $F_{0.5} = 59.7$), PIE with 12 encoder layers (PIE-BASE, $F_{0.5} = 56.6$) and competitive ED architecture by Lichtarge et al. (2019) with 6 encoder and 6 decoder layers (T2T, $F_{0.5} = 56.8$) on CoNLL-14 test set. All decoding experiments were run and measured on a n1-standard-2[2] VM instance with a single TPU shard (v-2.8). We observe that even PIE-LARGE is between a factor of 5 to 15 faster than an equivalent transformer-based ED model

[2] https://cloud.google.com/compute/docs/machine-types#standard_machine_types

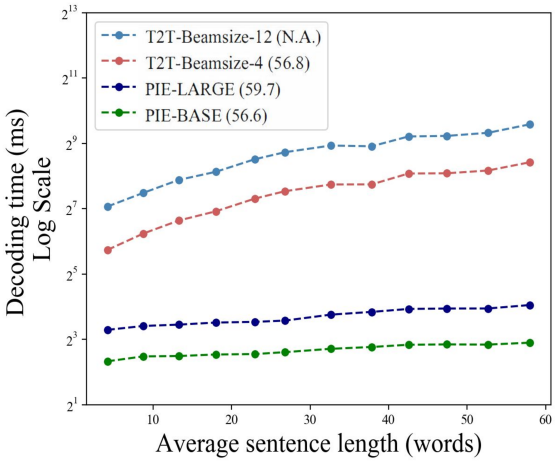

Figure 3: Comparing average decoding time in milliseconds (log scale) of PIE Vs transformer based ED model with two beam widths. Numbers in legend denote M2 score.

(T2T) with beam-size 4. The running time of PIE-LARGE increases sub-linearly with sentence length whereas the ED model's decoding time increases linearly.

3.1.3 Impact of Iterative Refinement

We next evaluate the impact of iterative refinements on accuracy in Table 4. Out of 1312 sentences in the test set, only 832 sentences changed in the first round which were then fed to the second round where only 156 sentences changed, etc. The average number of refinement rounds per example was 2.7. In contrast, a sequential model on this dataset would require 23.2 steps corresponding to the average number of tokens in a sentence. The $F_{0.5}$ score increases from 57.9 to 59.5 at the end of the second iteration.

Table 5 presents some sentences corrected by PIE. We see that PIE makes multiple parallel edits in a round if needed. Also, we see how refinement over successive iterations captures output space dependency. For example, in the second sentence `interact` gets converted to `interacted` followed by insertion of `have` in the next round.

3.1.4 Ablation study on the PIE Architecture

In this section we perform ablation studies to understand the importance of individual features of the PIE model.

Synthetic Training We evaluate the impact of training on the artifically generated GEC corpus in row 2 of Table 6. We find that without it the $F_{0.5}$ score is 3.4 points lower.

Work	Method	CONLL-14			JFLEG
		P	R	$F_{0.5}$	GLEU$^+$
Zhao et al. (2019)	Transformer + Pre-training + LM + Spellcheck + Ensemble Decoding (4)	71.6	38.7	61.2	61.0
Lichtarge et al. (2019)	Transformer + Pre-training + Iterative refinement + Ensemble Decoding (8)	66.7	43.9	60.4	63.3
Grundkiewicz and Junczys-Dowmunt (2018)	SMT + NMT (Bi-GRU) + LM + Ensemble Decoding (4) + Spellcheck	66.8	34.5	56.3	61.5
Chollampatt and Ng (2018a)	CNN + LM + Ensemble Decoding (4) + Spellcheck	65.5	33.1	54.8	57.5
PIE (This work)	Transformer + LM + Pre-training + Spellcheck + Iterative refinement + Ensembles (5)	68.3	43.2	61.2	61.0

Table 3: Comparison of recent GEC models trained using publicly available corpus. All the methods here except ours perform sequential decoding. Precision and Recall are represented by P and R respectively.

	R1	R2	R3	R4
#edited	832	156	20	5
P	64.5	65.9	66.1	66.1
R	41.0	42.9	43.0	43.0
$F_{0.5}$	57.9	59.5	59.7	59.7

Table 4: Statistics on successive rounds of iterative refinement. First row denotes number changed sentences (out of 1312) with each round on CONLL-14 (test).

x	I started invoving into Facebook one years ago .
PIE1	I started ***involving in*** Facebook one ***year*** ago .
PIE2	I started ***involved*** in Facebook one year ago .
PIE3	I started ***getting*** involved in Facebook one year ago .
x	Since ancient times , human interact with others face by face .
PIE1	Since ancient times , humans ***interacted*** with others face ***to*** face .
PIE2	Since ancient times , humans ***have*** interacted with others face to face .
x	However , there are two sides of stories always .
PIE1	However , there are ***always*** two sides ***to*** stories ~~always~~ .
PIE2	However , there are always two sides to ***the*** stories .
PIE3	However , there are always two sides to the ***story*** .

Table 5: Parallel and Iterative edits done by PIE.

Factorized Logits We evaluate the gains due to our edit-factorized BERT model (Section 2.3.2) over the default BERT model (Section 2.3.1). In Table 6 (row 3) we show that compared to the factorized model (row 2) we get a 1.2 point drop in $F_{0.5}$ score in absence of factorization.

Inserts as Appends on the preceding word was another important design choice. The alternative of predicting `insert` independently at each gap with a null token added to Σ_a performs 2.7 $F_{0.5}$ points poorly (Table 6 row 4 vs row 2).

#	Methods	P	R	$F_{0.5}$
1	PIE	66.1	43.0	59.7
2	− Synthetic training	67.2	34.2	56.3
3	−Factorized-logits	66.4	32.8	55.1
4	−Append +Inserts	57.4	42.5	53.6
5	−Transformations	63.6	27.9	50.6
6	−LM Pre-init	48.8	18.3	36.6
7	PIE on BERT-Base	67.8	34.0	56.6

Table 6: Ablation study on the PIE model.

Transformation edits are significant as we observe a 6.3 drop in recall without them (row 5).

Impact of Language Model We evaluate the benefit of starting from BERT's pre-trained LM by reporting accuracy from an un-initialized network (row 6). We observe a 20 points drop in $F_{0.5}$ establishing the importance of LMs in GEC.

Impact of Network Size We train the BERT-Base model with one-third fewer parameters than BERT-LARGE. From Table 6 (row 7 vs 1) we see once again that size matters in deep learning!

3.2 More Sequence Transduction Tasks

We demonstrate the effectiveness of PIE model on two additional local sequence transduction tasks recently used in (Ribeiro et al., 2018).

Spell Correction We use the twitter spell correction dataset (Aramaki, 2010) which consists of 39172 pairs of original and corrected words obtained from twitter. We use the same train-dev-valid split as (Ribeiro et al., 2018) (31172/4000/4000). We tokenize on characters, and our vocabulary Σ and Σ_a comprises the 26

Methods	Spell	OCR
Seq2Seq (Soft, lstm)	46.3	79.9
Seq2Seq (Hard,lstm)	52.2	58.4
Seq2Seq (Soft-T2T)	67.6	84.5
Ribeiro2018 (best model)	54.1	81.8
PIE	67.0	87.6

Table 7: Comparing PIE with Seq2Seq models (top-part) and Ribeiro's parallel model on two other local sequence transduction tasks.

	PIE	Soft-T2T
Spell	80.43	36.62
OCR	65.44	43.02

Table 8: Wall-clock decoding speed in words/second of PIE and a comparable Seq2Seq T2T model.

lower cased letters of English.

Correcting OCR errors We use the Finnish OCR data set[3] by (Silfverberg et al., 2016) comprising words extracted from Early Modern Finnish corpus of OCR processed newspaper text. We use the same train-dev-test splits as provided by (Silfverberg et al., 2016). We tokenize on characters in the word. For a particular split, our vocabulary Σ and Σ_a comprises of all the characters seen in the training data of the split.

Architecture For all the tasks in this section, PIE is a 4 layer self-attention transformer with 200 hidden units, 400 intermediate units and 4 attention heads. No $\mathcal{L}$ pre-initialization is done. Also, number of iterations of refinements is set to 1.

Results Table 7 presents whole-word 0/1 accuracy for these tasks on PIE and the following methods: Ribeiro et al. (2018)'s local transduction model (described in Section 4), and LSTM based ED models with hard monotonic attention (Aharoni and Goldberg, 2017) and soft-attention (Bahdanau et al., 2015) as reported in (Ribeiro et al., 2018). In addition, for a fair decoding time comparison, we also train a transformer-based ED model referred as Soft-T2T with 2 encoder, 2 decoder layers for spell-correction and 2 encoder, 1 decoder layer for OCR correction. We observe that PIE's accuracy is comparable with ED models in both the tasks. Table 8 compares decoding speed of PIE with Soft-T2T in words/second. Since more than 90% of words have fewer than 9 tokens and the token vocabulary Σ is small, decoding speed-ups of PIE over ED model on these tasks is modest compared to GEC.

[3] https://github.com/mpsilfve/ocrpp

4 Related Work

Grammar Error Correction (GEC) is an extensively researched area in NLP. See Ng et al. (2013) and Ng et al. (2014) for past shared tasks on GEC, and this[4] website for current progress. Approaches attempted so far include rules (Felice et al., 2014), classifiers (Rozovskaya and Roth, 2016), statistical machine translation (SMT) (Junczys-Dowmunt and Grundkiewicz, 2016), neural ED models (Chollampatt and Ng, 2018a; Junczys-Dowmunt et al., 2018; Ge et al., 2018a), and hybrids (Grundkiewicz and Junczys-Dowmunt, 2018). All recent neural approaches are sequential ED models that predict either word sequences (Zhao et al., 2019; Lichtarge et al., 2019) or character sequences (Xie et al., 2016) using either multi-layer RNNs (Ji et al., 2017; Grundkiewicz and Junczys-Dowmunt, 2018) or CNNs(Chollampatt and Ng, 2018a; Ge et al., 2018a) or Transformers (Junczys-Dowmunt et al., 2018; Lichtarge et al., 2019). Our sequence labeling formulation is similar to (Yannakoudakis et al., 2017) and (Kaili et al., 2018) but the former uses it to only detect errors and the latter only corrects five error-types using separate classifiers. Edits have been exploited in earlier GEC systems too but very unlike our method of re-architecting the core model to label input sequence with edits. Schmaltz et al. (2017) interleave edit tags in target tokens but use seq2seq learning to predict the output sequence. Chollampatt and Ng (2018a) use edits as features for rescoring seq2seq predictions. Junczys-Dowmunt et al. (2018) use an edit-weighted MLE objective to emphasise corrective edits during seq2seq learning. Stahlberg et al. (2019) use finite state transducers, whose state transitions denote possible edits, built from an unlabeled corpus to constrain the output of a neural beam decoder to a small GEC-feasible space.

Parallel decoding in neural machine translation Kaiser et al. (2018) achieve partial parallelism by first generating latent variables sequentially to model dependency. Stern et al. (2018) use a parallel generate-and-test method with modest speed-up. Gu et al. (2018) generate all tokens

[4] https://nlpprogress.com/english/grammatical_error_correction.html

in parallel but initialize decoder states using latent fertility variables to determine number of replicas of an encoder state. We achieve the effect of fertility using delete and append edits. Lee et al. (2018) generate target sequences iteratively but require the target sequence length to be predicted at start. In contrast our in-place edit model allows target sequence length to change with appends.

Local Sequence Transduction is handled in Ribeiro et al. (2018) by first predicting insert slots in $\mathbf{x}$ using learned insertion patterns and then using a sequence labeling task to output tokens in $\mathbf{x}$ or a special token `delete`. Instead, we output edit operations including word transformations. Their pattern-based insert pre-slotting is unlikely to work for more challenging tasks like GEC. Koide et al. (2018) design a special edit-invariant neural network for being robust to small edit changes in input biological sequences. This is a different task than ours of edit prediction. Yin et al. (2019) is about neural representation of edits specifically for structured objects like source code. This is again a different problem than ours.

5 Conclusion

We presented a parallel iterative edit (PIE) model for local sequence transduction with a focus on the GEC task. Compared to the popular encoder-decoder models that perform sequential decoding, parallel decoding in the PIE model yields a factor of 5 to 15 reduction in decoding time. The PIE model employs a number of ideas to match the accuracy of sequential models in spite of parallel decoding: it predicts in-place edits using a carefully designed edit space, iteratively refines its own predictions, and effectively reuses state-of-the-art pre-trained bidirectional language models. In the future we plan to apply the PIE model to more ambitious transduction tasks like translation.

Acknowledgements This research was partly sponsored by a Google India AI/ML Research Award and Google PhD Fellowship in Machine Learning. We gratefully acknowledge Google's TFRC program for providing us Cloud-TPUs. We thank Varun Patil for helping us improve the speed of pre-processing and synthetic-data generation pipelines.

References

Roee Aharoni and Yoav Goldberg. 2017. Morphological inflection generation with hard monotonic attention. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, pages 2004–2015.

Eiji Aramaki. 2010. Typo corpus. http://luululu.com/tweet/.

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

Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. 2013. One billion word benchmark for measuring progress in statistical language modeling. *arXiv preprint arXiv:1312.3005*.

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*.

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 and Hwee Tou Ng. 2012. Better evaluation for grammatical error correction. In *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 568–572. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Mariano Felice, Zheng Yuan, Øistein E Andersen, Helen Yannakoudakis, and Ekaterina Kochmar. 2014. Grammatical error correction using hybrid systems and type filtering. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 15–24.

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*.

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)*, pages 284–290.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O.K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *International Conference on Learning Representations (ICLR)*.

Jianshu Ji, Qinlong Wang, Kristina Toutanova, Yongen Gong, Steven Truong, and Jianfeng Gao. 2017. A nested attention neural hybrid model for grammatical error correction. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 753–762, Vancouver, Canada. 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.

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.

Zhu Kaili, Chuan Wang, Ruobing Li, Yang Liu, Tianlei Hu, and Hui Lin. 2018. A simple but effective classification model for grammatical error correction. *arXiv preprint arXiv:1807.00488*.

Lukasz Kaiser, Samy Bengio, Aurko Roy, Ashish Vaswani, Niki Parmar, Jakob Uszkoreit, and Noam Shazeer. 2018. Fast decoding in sequence models using discrete latent variables. In *Proceedings of the 35th International Conference on Machine Learning*, pages 2390–2399.

Satoshi Koide, Keisuke Kawano, and Takuro Kutsuna. 2018. Neural edit operations for biological sequences. In *Advances in Neural Information Processing Systems 31*.

D. Koller and N. Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press.

Jason Lee, Elman Mansimov, and Kyunghyun Cho. 2018. Deterministic non-autoregressive neural sequence modeling by iterative refinement. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Vladimir I Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710.

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*.

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.

Courtney Napoles, Keisuke Sakaguchi, Matt Post, and Joel Tetreault. 2016. Gleu without tuning. *arXiv preprint arXiv:1605.02592*.

Courtney Napoles, Keisuke Sakaguchi, and Joel Tetreault. 2017. Jfleg: A fluency corpus and benchmark for grammatical error correction. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 229–234, Valencia, Spain. Association for Computational Linguistics.

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.

Hwee Tou Ng, Siew Mei Wu, Yuanbin Wu, Christian Hadiwinoto, and Joel Tetreault. 2013. The conll-2013 shared task on grammatical error correction. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning: Shared Task*, pages 1–12, Sofia, Bulgaria. Association for Computational Linguistics.

Aaron van den Oord, Yazhe Li, Igor Babuschkin, Karen Simonyan, Oriol Vinyals, Koray Kavukcuoglu, George van den Driessche, Edward Lockhart, Luis Cobo, Florian Stimberg, Norman Casagrande, Dominik Grewe, Seb Noury, Sander Dieleman, Erich Elsen, Nal Kalchbrenner, Heiga Zen, Alex Graves, Helen King, Tom Walters, Dan Belov, and Demis Hassabis. 2018. Parallel WaveNet: Fast high-fidelity speech synthesis. In *Proceedings of the 35th International Conference on Machine Learning*, pages 3918–3926.

Joana Ribeiro, Shashi Narayan, Shay B. Cohen, and Xavier Carreras. 2018. Local string transduction as sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics (ACL-COLING)*.

Alla Rozovskaya and Dan Roth. 2016. Grammatical error correction: Machine translation and classifiers. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 2205–2215.

Allen Schmaltz, Yoon Kim, Alexander M. Rush, and Stuart M. Shieber. 2017. Adapting sequence models for sentence correction. In *EMNLP*, pages 2807–2813.

Miikka Silfverberg, Pekka Kauppinen, Krister Lindén, et al. 2016. Data-driven spelling correction using weighted finite-state methods. In *The 54th Annual Meeting of the Association for Computational Linguistics Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*. ACL.

Felix Stahlberg, Christopher Bryant, and Bill Byrne. 2019. Neural grammatical error correction with finite state transducers. *arXiv preprint arXiv:1903.10625*.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. In *Advances in Neural Information Processing Systems 31*.

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

Helen Yannakoudakis, Ted Briscoe, and Ben Medlock. 2011. A new dataset and method for automatically grading esol texts. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 180–189. Association for Computational Linguistics.

Helen Yannakoudakis, Marek Rei, Øistein E Andersen, and Zheng Yuan. 2017. Neural sequence-labelling models for grammatical error correction. In *EMNLP*, pages 2795–2806.

Pengcheng Yin, Graham Neubig, Miltiadis Allamanis, Marc Brockschmidt, and Alexander L. Gaunt. 2019. Learning to represent edits. In *International Conference on Learning Representations*.

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. *arXiv preprint arXiv:1903.00138*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.

A Appendix

A.1 Modified Cost in Levenshtein Distance Algorithm

We keep delete and insert cost as 1 as usual, but for substitutions, we use $1 + \epsilon d$, where d is the absolute difference between the number of characters of replaced and substituted word. We set ϵ to 0.001. Table 9 shows two minimum edit diffs if the substitution penalty has no such offset. In Diff-1, {`.`} substitutes {`,`} and `Then` substitutes `then`, followed by insertion of {`,`}. In Diff-2, {`.`} is inserted after `sat` followed by `Then` substituting {`,`} , followed by {`,`} substituting `then` . In absence of offset in substitution penalty, both the diffs have edit distance of 3. In presence of offset, Diff-1 has an edit-distance of 3, while Diff-2 has an edit-distance of 3.006, this allows Diff-1 to be preferred over Diff-2. As we observe, offset helps in selection of well aligned minimum edit diffs among multiple minimum edit diffs.

x	[He sat , then he ran]
y	[He sat . Then , he ran]
op-1	[C C S(, , .) S(`then` , `Then`) I(,) , C C]
Diff-1	[He sat -, +. -then +Then +, he ran]
op-2	[C C I(.) S(, , `Then`) S(`then` , ,) , C C]
Diff-2	[He sat +. -, +Then -then +, he ran]

Table 9: Two diffs having same edit distance in the absence of offset in substitution penalty

A.2 Suffix transformations

Transformation	Example
ADDSUFFIX(s)	play ⇌ plays
ADDSUFFIX(d)	argue ⇌ argued
ADDSUFFIX(es)	express ⇌ expresses
ADDSUFFIX(ing)	play ⇌ playing
ADDSUFFIX(ed)	play ⇌ played
ADDSUFFIX(ly)	nice ⇌ nicely
ADDSUFFIX(er)	play ⇌ player
ADDSUFFIX(al)	renew ⇌ renewal
ADDSUFFIX(n)	rise ⇌ risen
ADDSUFFIX(y)	health ⇌ healthy
ADDSUFFIX($ation$)	inform ⇌ information
CHANGE-e-TO-ing	use ⇌ using
CHANGE-d-TO-t	spend ⇌ spent
CHANGE-d-TO-s	compared ⇌ compares
CHANGE-s-TO-ing	claims ⇌ claiming
CHANGE-n-TO-ing	deafen ⇌ deafening
CHANGE-nce-TO-t	insistence ⇌ insistent
CHANGE-s-TO-ed	visits ⇌ visited
CHANGE-ing-TO-ed	using ⇌ used
CHANGE-ing-TO-ion	creating ⇌ creation
CHANGE-ing-TO-$ation$	adoring ⇌ adoration
CHANGE-t-TO-ce	reluctant ⇌ reluctance
CHANGE-y-TO-ic	homeopathy ⇌ homeopathic
CHANGE-t-TO-s	meant ⇌ means
CHANGE-e-TO-al	arrive ⇌ arrival
CHANGE-y-TO-ily	angry ⇌ angrily
CHANGE-y-TO-ied	copy ⇌ copied
CHANGE-y-TO-$ical$	biology ⇌ biological
CHANGE-y-TO-ies	family ⇌ families

Table 10: 29 suffix transformations and their corresponding inverse make total 58 suffix transformations.

A.3 Artificial Error Generation

Figure 4 shows the algorithm used to introduce artificial errors in clean dataset. Given a sentence, first the number of errors in that sentence is determined by sampling from a multinoulli (over $\{0 \dots 4\}$). Similarly, an error is chosen independently from another multinoulli (over

Input: U: dataset of clean sentences
$AppendError \leftarrow 0$
$VerbError \leftarrow 1$
$ReplaceError \leftarrow 2$
$DeleteError \leftarrow 3$
for sentence in U **do**
  $errorCount \leftarrow multinoulli(0.05, 0.07, 0.25, 0.35, 0.28)$
  for $i \in 1 \ldots errorCount$ **do**
    $errorType \leftarrow multinoulli(0.30, 0.25, 0.25, 0.20)$
    introduce error of type $errorType$
return

Figure 4: Algorithm to introduce errors in clean dataset

$\{AppendError, VerbError, ReplaceError, DeleteError\}$). The distribution of the number of errors in a sentence and probability of each kind of error was obtained based on the available parallel corpus. For append, replace and delete errors, a position is randomly chosen for the error occurrence. For append error the word in that position is dropped. For delete error a spurious word from a commonly deleted words dictionary is added to that position. For replace error, both the actions are done. For a verb error, a verb is chosen at random from the sentence and is replaced by a random verb form of the same word. Commonly deleted words are also obtained from the parallel corpus.

A.4 Wall-clock Decoding Times

Average sentence length (words)	4.30	8.76	13.30	18.00	22.96	27.79	32.64	37.85	42.54	47.41	52.6	58.8
T2T-bs-4 (56.8)	53.5	75.5	99.7	121.1	158.5	185.5	214.0	214.5	270.1	271.4	287.8	343.1
T2T-bs-12 (N.A.)	134.2	179.1	236.0	279.9	365.6	424.3	488.5	481.3	592.6	599.7	640.2	767.2
PIE-BASE (56.6)	5.0	5.5	5.6	5.8	5.8	6.1	6.5	6.8	7.1	7.2	7.1	7.4
PIE-LARGE (59.7)	9.8	10.6	10.9	11.4	11.6	11.9	13.5	14.3	15.2	15.4	15.4	16.6

Table 11: Wall clock decoding time in milliseconds for various GEC models

A.5 Hyperparameters

Hyperparameters	PIE-BASE GEC	PIE-LARGE GEC	PIE OCR/SPELL Correction
attention_probs_dropout_prob	0.1	0.1	0.1
directionality	bi-directional	bi-directional	bi-directinoal
hidden_act	gelu	gelu	gelu
hidden_dropout_prob	0.1	0.1	0.1
hidden_size	768	1024	200
initializer_range	0.02	0.02	0.02
intermediate_size	3072	4096	400
max_position_embeddings	512	512	40
num_attention_heads	12	16	4
num_hidden_layers	12	24	4
type_vocab_size	2	2	2
vocab_size	28996	28996	110/26
copy_weight	0.4	0.4	1

Table 12: Hyperparameters used in PIE Model for GEC, OCR Correction and Spell Correction. In GEC, copy weight of 0.4 is used (based on validation set) to scale down the loss corresponding to copy label for handling class imbalance

Hyperparameters	T2T GEC	T2T OCR Correction	T2T Spell Correction
T2T hparams set	transformer_clean_big_tpu	transformer_tiny	transformer_tiny
num_encoder_layers	6	2	2
num_decoder_layers	6	1	2
hidden_size	1024	200	200
filter_size	4096	400	400

Table 13: Hyperparameters used in T2T transformer models for GEC, OCR Correction and Spell Correction. tensor2tensor (https://github.com/tensorflow/tensor2tensor) library was used for implementation