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 $\rightleftharpoons$ plays
ADDSUFFIX(d)	argue $\rightleftharpoons$ argued
ADDSUFFIX(es)	express $\rightleftharpoons$ expresses
ADDSUFFIX(ing)	play $\rightleftharpoons$ playing
ADDSUFFIX(ed)	play $\rightleftharpoons$ played
ADDSUFFIX(ly)	nice $\rightleftharpoons$ nicely
ADDSUFFIX(er)	play $\rightleftharpoons$ player
ADDSUFFIX(al)	renew $\rightleftharpoons$ renewal
ADDSUFFIX(n)	rise $\rightleftharpoons$ risen
ADDSUFFIX(y)	health $\rightleftharpoons$ healthy
ADDSUFFIX($ation$)	inform $\rightleftharpoons$ information
CHANGE-e-TO-ing	use $\rightleftharpoons$ using
CHANGE-d-TO-t	spend $\rightleftharpoons$ spent
CHANGE-d-TO-s	compared $\rightleftharpoons$ compares
CHANGE-s-TO-ing	claims $\rightleftharpoons$ claiming
CHANGE-n-TO-ing	deafen $\rightleftharpoons$ deafening
CHANGE-nce-TO-t	insistence $\rightleftharpoons$ insistent
CHANGE-s-TO-ed	visits $\rightleftharpoons$ visited
CHANGE-ing-TO-ed	using $\rightleftharpoons$ used
CHANGE-ing-TO-ion	creating $\rightleftharpoons$ creation
CHANGE-ing-TO-$ation$	adoring $\rightleftharpoons$ adoration
CHANGE-t-TO-ce	reluctant $\rightleftharpoons$ reluctance
CHANGE-y-TO-ic	homeopathy $\rightleftharpoons$ homeopathic
CHANGE-t-TO-s	meant $\rightleftharpoons$ means
CHANGE-e-TO-al	arrive $\rightleftharpoons$ arrival
CHANGE-y-TO-ily	angry $\rightleftharpoons$ angrily
CHANGE-y-TO-ied	copy $\rightleftharpoons$ copied
CHANGE-y-TO-$ical$	biology $\rightleftharpoons$ biological
CHANGE-y-TO-ies	family $\rightleftharpoons$ 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 $\{0 \ldots 4\}$). 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