EdiT5: Semi-Autoregressive Text Editing with T5 Warm-Start

Jonathan Mallinson **Jakub Adamek** **Eric Malmi** **Aliaksei Severyn**
Google Research
{jonmall,enkait,emalmi,severyn}@google.com

Abstract

We present EdiT5[1] – a novel semi-autoregressive text-editing model designed to combine the strengths of non-autoregressive text-editing and autoregressive decoding. EdiT5 is faster during inference than conventional sequence-to-sequence (seq2seq) models, while being capable of modeling flexible input-output transformations.

This is achieved by decomposing the generation process into three sub-tasks: (1) *tagging* to decide on the subset of input tokens to be preserved in the output, (2) *re-ordering* to define their order in the output text, and (3) *insertion* to infill the missing tokens that are not present in the input. The *tagging* and *re-ordering* steps, which are responsible for generating the largest portion of the output, are non-autoregressive, while the *insertion* step uses an autoregressive decoder.

Depending on the task, EdiT5 on average requires significantly fewer autoregressive steps, demonstrating speedups of up to 25x when compared to seq2seq models. Quality-wise, EdiT5 is initialized with a pre-trained T5 checkpoint yielding comparable performance to T5 in high-resource settings when evaluated on three NLG tasks: Sentence Fusion, Grammatical Error Correction, and Decontextualization while clearly outperforming T5 in low-resource settings.

1 Introduction

Pre-trained seq2seq models such as T5 (Raffel et al., 2020), BART (Lewis et al., 2020a), and MASS (Song et al., 2019) have established strong baselines for the majority of text-to-text transduction tasks. A recent trend to massively scale up model sizes, e.g., all the way up to 540B params (Chowdhery et al., 2022), as well as the sizes of pretraining corpora, has further pushed the state-of-the-art without signs of reaching a plateau. From a practical point of view, running inference with such models is prohibitively expensive for most applications, which motivates the work on finding efficient recipes for model distillation, e.g., (Kim and Rush, 2016) and choosing a model architecture that can provide a better trade-off between performance on a given task and inference speed. A typical choice is to distill a large language model into a smaller seq2seq model, e.g., Transformer (Vaswani et al., 2017). In this paper we propose a novel model architecture EdiT5which blends ideas from a seq2seq T5 (Raffel et al., 2020) and text-editing to provide faster inference without sacrificing on task performance.

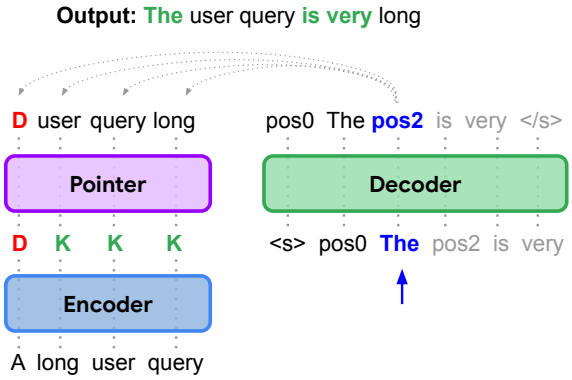

Figure 1: EdiT5 transforms the input text *A long user query* into the output *The user query is very long* by first generating a sequence of edit tags `D K K K` (where `K` stands for keeping and `D` for deleting the input token), re-ordering the input tokens with the pointer network, and infilling missing tokens into the source sequence with an autoregressive decoder which jointly predicts the text spans (`The` and `is very`) and the position where to insert them (`pos0` and `pos2`). The blue arrow shows how the token `pos2` is predicted conditioned on the prefix `<s> pos0 The` generated thus far. The dotted arrow lines depict the encoder-decoder cross attention over the re-ordered input tokens and edit tags.

Seq2seq-based models output text token-by-

[1]Code and pre-trained models https://edit5.page.link/code

Findings of the Association for Computational Linguistics: EMNLP 2022, pages 2126–2138
December 7-11, 2022

token from scratch, allowing them to model any kind of input-output relationship. However, for many real-world tasks this degree of generality is unnecessary, especially for monolingual tasks where the input and output texts have relatively high degrees of overlap. In such cases a natural approach is to cast conditional text generation as a text-editing task, where the model learns to construct target texts by applying a set of edit operations to the inputs (Malmi et al., 2022). Typically the set of edit operations is defined ahead of time (Omelianchuk et al., 2020; Malmi et al., 2019; Awasthi et al., 2019), which on the one hand limits the flexibility of the model to reconstruct arbitrary output texts from the inputs, but on the other, leads to latency improvements as the limited set of allowed operations significantly reduces the output vocabulary of the decoder. In this paper, we propose an approach which is both fast at inference time and flexible, able to model arbitrary rewrites.

Faster inference. A common method for achieving low latency in serving models is to reduce their size, thus reducing their computational cost. Doing so naively, however, often leads to inferior model quality, and much work has gone into finding better methods for model size reduction, such as distillation (Kim and Rush, 2016).

Regardless of model size, one of the major contributors to the total inference time for seq2seq models is the decoder, which generates the output sequence step-by-step. EDIT5 also relies on an autoregressive decoder, but generates the majority of the output sequence with its tagging and pointing networks, and as such the decoder makes far fewer steps.

Flexible text-editing. Recent text-editing approaches, e.g., (Awasthi et al., 2019; Malmi et al., 2019), are not as powerful as general purpose seq2seq approaches when it comes to modeling arbitrary input-output text transductions. EDIT5 supports open-vocabulary generation by relying on an autoregressive decoder. In the extreme case, where there is no overlap between the source and the target texts, it reduces to a vanilla seq2seq model generating the entire output from scratch. However, when the input and output overlap, it can benefit from the *tagging* and *pointer* networks to reconstruct the bulk of the output text that is further infilled (refined) by the autoregressive decoder.

Warm start. Training a high-precision text generation model typically requires large amounts of high-quality supervised data. Self-supervised techniques based on text in-filling (Rothe et al., 2020a; Lewis et al., 2020b; Raffel et al., 2020) have been shown to provide a crucial advantage over non-pre-trained models especially in low-resource settings. Hence, we design EDIT5 to be able to benefit from already existing pre-trained language models (specifically T5), where the final model is directly fine-tuned on the downstream task.

EDIT5 decomposes the generation task into three steps: *tagging*, *pointing* and *insertion* (see Fig. 1). The tagger and pointer networks decide which source tokens to preserve and in which order they should appear in the output, thus allowing for arbitrary word dropping and reordering. The tagger is implemented using a non-autoregressive feedforward network, and pointing is implemented using a novel non-autoregressive pointing mechanism (Vinyals et al., 2015) combined with sinkhorn layers (Mena et al., 2018). The insertion network inserts/infills words which are present in the target sequence but do not appear in the source sequence. The network is implemented using an autoregressive transformer decoder, which attends to the tagged, reordered source sequence. The decoder predicts both the locations of where the token spans should be infilled, as well as the spans themselves.

We evaluate EDIT5 on three distinct text generation tasks: Sentence Fusion, Grammatical Error Correction (GEC), and Decontextualization, comparing to recent text-editing approaches and T5. Each task is unique in the editing operations required and the amount of training data available, which helps to better quantify the value of modeling decisions we have integrated into EDIT5.

Additionally, we explore the impact of training data size and model size on EDIT5. Finally we quantify the latency of EDIT5, providing a detailed analysis and comparison to T5.

2 Model description

The model architecture of EDIT5 resembles a vanilla Transformer (Vaswani et al., 2017) composed of an **encoder** and a **decoder**. EDIT5 decomposes the generation of a text $\mathbf{y}$ from an input $\mathbf{x}$ into three parts: predicting a sequence of edit tags $\mathbf{y}^t$ (indicating whether a token from the input

should be copied to the output), a permutation of the input tokens π (indicating the order that copied tokens should appear in in the output), and a sequence of tokens $\mathbf{y}^d$ (indicating additional tokens that should be in the output, and where in the permuted input they should be inserted). $\mathbf{y}^t$ and π are modeled by the **encoder**, and $\mathbf{y}^d$ by the **decoder**.

There are multiple ways to choose the triple ($\mathbf{y}^t$, π, $\mathbf{y}^d$) for a given ($\mathbf{x}$, $\mathbf{y}$) pair. During dataset creation we choose a single such triple for each training pair (see section 2.1 for details), in which case the probability of $\mathbf{y}$ can be expressed as:

$$P(\mathbf{y}|\mathbf{x}) := \left(\prod_{i}^{|\mathbf{y}^d|} P(\mathbf{y}_i^d|\mathbf{y}_{<i}^d, \mathbf{y}^t, \pi, \mathbf{x})\right) * P(\pi|\mathbf{y}^t, \mathbf{x}) * P(\mathbf{y}^t|\mathbf{x}) \quad (1)$$

During inference, we first greedily set $\mathbf{y}^t$ to maximize the third term, then π to maximize the second term, and finally $\mathbf{y}^d$ to maximize the first term. The output text $\mathbf{y}$ is realized by applying the tags $\mathbf{y}^t$ and permutation π to the input sequence $\mathbf{x}$ and then inserting the tokens $\mathbf{y}^d$.

2.1 Text-editing encoder

The EDIT5 encoder consists of three steps: encoding, tagging, and pointing.

Encoder. The source sentence $\mathbf{x}$ is first encoded using N transformer layers into the hidden representations $\mathbf{h}$.

Tagging. The tag sequence $\mathbf{y}^t$ is constructed as follows: source tokens that must be copied are assigned the KEEP tag, tokens not present in the output are marked by the DELETE tag. Tags are predicted by applying a single transformer layer followed by a classification layer to the output of the encoder $\mathbf{h}$, which is trained using cross-entropy:

$$\mathcal{L}_{tagging} = -\sum_{j}^{|\mathbf{x}|} \log P(y_j^t|f_t(\mathbf{h})_j) \quad (2)$$

where $\mathbf{y}^t$ are the gold tags, j is the index of the source token, and f_t is a transformer layer followed by a classification layer. During inference we use *argmax* to determine the tags, whereas during training we use the gold tags. The encoder hidden state is then updated to take these tags into account:

$$\mathbf{h}_j^t = f_{te}([\mathbf{h}_j; TE(\mathbf{y}_j^t)]) \quad (3)$$

Where TE is a tag embedding layer, whose output is concatenated to the original hidden representation of the source sequence, before a feed-forward layer f_{te} is applied.

Pointing. In many tasks it is helpful for the model to be able to rearrange the kept input tokens. For example, we can grammatically correct the sentence *Who you are?* to *Who are you?* purely by reordering tokens from the input. In EDIT5 this is made possible thanks to its pointing mechanism. In contrast, in text editing approaches such as Malmi et al. (2019); Dong et al. (2019), correcting this sentence involves first deleting the words *you are* and then recreating them in the right order.

Given a sequence $\mathbf{x}$ and the predicted tags $\mathbf{y}^t$, the re-ordering model generates a permutation π. Our implementation is based on a pointer network (Vinyals et al., 2015), where an attention mechanism points to the next token. We follow Mallinson et al. (2020) which, unlike previous approaches where a decoder state attends over an encoder sequence, applies intra-attention, where source tokens attend to all other source tokens. As such the output of this model is a series of predicted pointers, where each source token predicts the token that comes after it. π can easily be constructed by daisy-chaining these predicted pointers together, as seen in Fig. 2. We calculate attention using key-query attention, where we include an additional transformer layer prior to the key network:

$$\alpha_{m,j} = f^q(\mathbf{h}^t)_m \cdot f^k(\mathbf{h}^t)_j \quad (4)$$

Where $\alpha_{m,j}$ is the unnormalized attention, f^q is the query network, a single feed-forward layer, and f^k is the key network, a transformer layer followed by a single feedfoward layer.

Unlike Mallinson et al. (2020), we ensure a valid permutation is formed, i.e. no token is pointed to twice, by using sinkhorn layers (Mena et al., 2018), which normalizes over both the rows and the columns of the intra-pointer attention α. Sinkhorn layers are defined as:

$$S^0 = \exp(\alpha) \quad (5)$$

$$S^i = T_c(T_r(S^{i-1}(\alpha)) \quad (6)$$

where $T_c^{j,m}(X) = \frac{X_{j,m}}{\sum_l X_{l,m}}$ is the column normalization operator and $T_r^{j,m}(X) = \frac{X_{j,m}}{\sum_l X_{j,l}}$ is the row normalization operator.

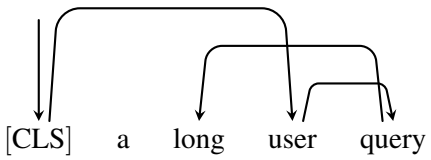

Figure 2: Pointing mechanism to transform *"a long user query"* into *"user query long"*.

The loss for the pointing network is defined as:

$$\mathcal{L}_{pointing} = CE(\pi|S(\alpha)) \quad (7)$$

Where CE is the cross-entropy loss. During inference we use argmax to determine π.

We use additional positional embeddings to update the hidden states with their new position (offset from 0). For example if *Who you are?* was reordered into *Who are you?*, the position information would be updated as $_0$Who $_2$you $_1$are $_3$?.

$$\mathbf{h}^p_j = (\mathbf{h}^t_j + \mathbf{PE}(\pi_j)) \quad (8)$$

where PE are learnt absolute positional embeddings (Devlin et al., 2019). These additional positional embeddings are masked out for those source words which do not appear in the target sequence. Finally we apply a transformer encoder layer to $\mathbf{h}^p$ forming the final encoded representation of the sequence $\mathbf{h}^f$. $\mathbf{h}^f$ captures the edits as well as the original sequence $\mathbf{x}$, and the decoder attends to this representation.

Decoder. We use a standard transformer decoder, which is tasked with inserting tokens which are in the output sequence but don't appear within the input sequence. EDIT5 takes advantage of the pre-training of a T5 model, where T5 was pre-trained to infill missing spans. When pre-training T5 uses special tokens *⟨pos_i⟩* to indicate where missing spans should be inserted, as demonstrated in Figure 3. EDIT5 re-purposes these special tokens, using them to indicate at which position new tokens should be infilled. I.e. *⟨pos_1⟩*, indicates that the tokens should be inserted after the first token. As such the decoder first decodes a special position token and then decodes the inserted tokens which should appear after this token. For example to insert *the cat* after the first token, the decoder generates: *⟨pos_1⟩ the cat*. The decoder is trained

Source/Target:	`a long user query .`
T5 Input:	`a` [X] `user query` [Y]
T5 decoder:	[X] `long` [Y] `.`
EdiT5 Input:	`user a query the`
EdiT5 tagger:	`K K K D`
EdiT5 pointer:	`a user query`
EdiT5 decoder:	[0] `long` [2] `.`

Figure 3: Example pre-training noise for T5 and EDIT5. `K` and `D` indicate keep and delete tags resspectivly, and [0] indicates *pos0*.

with a standard cross-entropy loss:

$$\mathcal{L}_{insertion} = -\sum_{i}^{|\mathbf{y}^d|} \log P(y^d_i|\mathbf{y}^d_{<i}, h^f) \quad (9)$$

Where i is the decoder index, and h^f is the encoder output. The loss for the entire model is defined as the sum of the three individual losses:

$$\mathcal{L} = \lambda_1\mathcal{L}_{tagging} + \lambda_2\mathcal{L}_{pointing} + \lambda_3\mathcal{L}_{insertion} \quad (10)$$

where λ_1, λ_2 and λ_3 are hyper-parameters determining the relative importance of tagging, pointing and insertion losses in the final loss.

Pre-training. While we initialize EDIT5 from T5 base, T5 was pre-trained with 12 decoder layers, and for EDIT5 we use a single decoder layer. To account for this change in the decoder layers, we perform additional pre-training. We use a pre-training objective which combines a T5 style span insertion task, with a generic text-editing denoising task, as used in BART (Lewis et al., 2020b). A source sentence is corrupted by dropping, swapping and adding spans (an example can be seen in Figure 3), and we task our model to reconstruct the original sentence. By introducing noise we are able to train the tagger to detect incorrect spans, and the pointer to reorder the sentence. The decoder then behaves like the T5 pre-training objective inserting the content of missing spans. Unlike BART's pre-training, our approach is computationally cheap, as we do not decode the entire sequence when training, instead just decoding the missing spans.

Dataset construction. When constructing the training dataset, there are many possible combinations of $\mathbf{y}^t$, π and $\mathbf{y}^d$ which could produce $\mathbf{y}$. For instance, all source tokens could be deleted and the decoder could then produce all the target tokens.

However to minimize latency, we wish to make the number of inserted tokens (i.e. the number of decoder steps) as small as possible, and maximize the number of kept tokens.

To produce alignments from a target sequence to a source sequence, we iterate left-to-right through characters in the target sequence, trying to find spans of target characters which appear in the sequence of source tokens, as described in Algorithm 1 (see Appendix A). Each source token can only be aligned to a single target span. Those target spans that can't be aligned are instead inserted after the closest previous aligned source token. In cases where there are multiple possible alignments, e.g. the same token appears multiple times in the source, we align the target character span to produce the longest contiguous span of source tokens aligned with the target, i.e. where source tokens appear one-after-another in the target sequence. To find the longest contiguous span we compare the contiguous overlap between source and target for each possible alignment.

3 Experiments

We evaluate EDIT5 on three distinct text-editing tasks: Sentence Fusion, Grammatical Error Correction, and Decontextualization. In addition to reporting previously published results for each task, we also compare to FELIX (Mallinson et al., 2020), a recent non-autoregressive text-editing model, and a strong pre-trained T5 baseline implemented in the T5X framework (Roberts et al., 2022).

Modeling. For EDIT5 we initialize with a T5 base model with a 12-layer Transformer encoder, and single-layer Transformer decoder. Our code is based on the Tensorflow Model Garden's (Hongkun Yu and Li, 2020) TF2 version of T5. After initializing with the T5 checkpoint, we further pre-train on the denoising objective (see Section 2.1) using the C4 corpus (Raffel et al., 2020), training for 100k steps.

For all experiments EDIT5 is trained using AdamW (Loshchilov and Hutter, 2019), additionally the learning rate was decayed using the validation set, and exact match is used for checkpoint selection. Tokenization is based on T5's SentencePiece vocabulary (Kudo and Richardson, 2018), with a vocabulary size of 32k. We, however, modify the vocabulary, removing tokens which have punctuation as a suffix, and replacing them with additional span insertion special token, giving EDIT5 512 span insertion special token. Unless otherwise stated, we use an input sequence length of 128. We performed minimal hyper-parameter selection, which is discussed in the Appendix.

Task Analysis. The chosen tasks cover a diverse set of edit operations and a wide range of dataset sizes, varying from under 11 thousand data points to over 4.5 million. Table 1 provides dataset statistics including: the size, input sequence length, output sequence length for seq2seq models, the output sequence length for EDIT5, and the translation error rate (TER) (Snover et al., 2006) between the source and target sentences. We use TER to highlight unique properties of each task.

From Table 1 we see that for all tasks EDIT5 requires significantly fewer decoder steps than a seq2seq model, which results in significant latency savings. We also see that decontextualization has the longest input and output sequences, where the maximum input length of decontextualization is 512 tokens. Decontextualization has the highest TER, with the major contribution being deletion, which is due to the input sequence consisting of a paragraph, whereas the output is a single sentence. In contrast GEC, has the shortest input and output sequence, with the majority of the dataset consisting of a single input and a single output sentence. GEC has the lowest TER, however it has the highest insertion TER. Sentence fusion consists of two sentences being rewritten into a single sentence, and has a middling TER and sequence lengths. It also has the fewest substitutions.

3.1 Sentence Fusion

Sentence Fusion is the task of fusing independent sentences into a coherent output sentence(s) (Geva et al., 2019). It requires operations such as inferring the appropriate discourse connective, pronominalization, reordering the text to introduce relative clauses, and changing the order of the input sentences.

Data. We use the "balanced Wikipedia" portion of the DiscoFuse dataset (Geva et al., 2019) and also study the impact of training data size by creating four additional smaller subsets of DiscoFuse consisting of: 450,000 (10%), 45,000 (1%), 4,500 (0.1%) and 450 (0.01%) data points.

Setup. Following Geva et al. (2019), we report *Exact match*, which is the percentage of exactly correctly predicted fusions. In addition to the T5

Dataset	Size	L_{src}	L_{tgt}	E5-Ins	TER	Ins	Del	Sub	Shft
Sentence fusion	4.5M	42.5	41.1	5.8	10.92	2.49	04.91	3.75	0.62
GEC	2.3M	24.3	24.7	4.6	09.72	2.99	01.19	5.05	0.49
Decontextualization	11K	193.9	49.1	7.2	84.80	0.28	90.64	6.43	2.65

Table 1: Statistics across tasks: size of the dataset (Size), source length in tokens (L_{src}), target length in tokens (L_{tgt}), EdiT5 insertion tokens (E5-Ins), and TER scores, including number of insertions (Ins), deletions (Del), substitutions (Sub), and shifts (Shft). Token counts are measured using a sentencepiece tokenizer and averaged over the development set.

	#Params	100%	10%	1%	0.1%	0.01%	latency
LaserTagger	110M	53.80	47.31	38.46	25.74	12.32	-
Felix	220M	61.31	52.85	45.45	36.87	16.96	**1.8**
Seq2Edits	279M	61.71	-	-	-	-	-
EdiT5	141M	64.95	59.26	**52.09**	**43.83**	**28.64**	2.2
- pre-training	141M	65.16	59.27	50.39	34.18	1.90	2.2
T5 base	220M	65.52	**59.75**	50.75	33.84	10.75	52.7
ROBERTA	380M	**66.6**	-	-	-	-	-
AugBERT	157M	65.0	-	-	-	-	-

Table 2: Sentence fusion results (Exact Match, lowercased) under various data conditions, latency (ms), and number of parameters.

baseline and the text-editing baselines LaserTagger (Malmi et al., 2019), Felix (Mallinson et al., 2020), and Seq2Edits (Stahlberg and Kumar, 2020), an autoregressive text-editing model, we also report state-of-the-art seq2seq models ROBERTASHARE (Rothe et al., 2020b), based on ROBERTA large, and AugBERT (Ben-David et al., 2020), based on BERT base. Additionally, we measure the impact of our pre-training (Section 2.1) initializing EdiT5 with a T5 checkpoint, without additional pre-training.

Results. From the top section in Table 2 we first observe that EdiT5 strongly outperforms other text-editing methods. Next it performs comparably to T5 in high-resource settings (100% and 10%), where it's just 0.5 points lower in exact match than T5, whilst achieving a latency that is 25 times faster, and using fewer parameters. The current SOTA, ROBERTASHARE, which outperforms EdiT5 by 1.5 points, is based on the ROBERTA large checkpoint which overall has more parameters and a larger encoder. In low-resource settings, EdiT5 clearly outperforms T5 by up to 18 points (0.01%, i.e. 450 training examples).

The results in Table 2 additionally demonstrate that the significant improvements of EdiT5 over Felix in high/medium-resource settings do not stem from EdiT5 pre-training. With 450 datapoints, pre-training is critical since there's a larger mismatch between EdiT5 and T5 checkpoints than there is between Felix and BERT checkpoints. We additionally ablated the impact of sinkhorn layers, and found that under the 100% data condition there was a modest decrease in performance (0.5 exact match points).

3.2 Decontextualization

Sentence decontextualization task was introduced by Choi et al. (2021). The goal is to rewrite an input sentence to make it stand-alone without the original context.

Data. We use the train, dev and test data from Choi et al. (2021), where sentences were selected from Wikipedia passages. Human annotators were asked to rewrite them, if possible, to be interpretable and grammatical without the context. We compare against T5 base, T5 xxl, Felix, and a copy baseline. All models use a sequence length of 512.

Metrics. Following Choi et al. (2021), we report exact match, exact match when a sentence needs to be rewritten and SARI F1 (deletion and addition) on unigrams (Xu et al., 2016).

Analysis. Results in Table 3 show that EdiT5 achieves a higher exact match scores, and SARI delete score when compared to T5 base, with a significant drop in latency and using fewer parameters. T5 base achieves significantly higher SARI add, suggesting its better at inserting new tokens, which is unsurprising as EdiT5 is primarily focused on copying the source sequence. Both T5 and EdiT5 achieve significantly higher numbers than Felix. EdiT5 and T5 base, however, still achieve a significantly lower score than the T5 xxl, which can be explained by the difference in model size.

3.3 Grammatical Error Correction

GEC requires systems to identify and fix grammatical errors in a given input text.

	#Params	EM	EMc	ADD	DEL	latency
Repeat	-	36	0	0	0	-
T5 xxl	11B	**52**	**32**	**43**	**47**	-
FELIX	220M	32	10	28	32	4
EDITS	141M	48	23	31	41	**3.8**
T5 base*	220M	40	21	36	40	75

Table 3: Decontextualization results, including exact match (*EM*, exact match on those sentences which need rewriting *EMc*, SARI *ADD*, SARI *DEL*ete, latency (ms), and number of parameters. * indicates scores were calculated by running the models provided by Choi et al. (2021) on the test set.

Data. We evaluate on the standard GEC test set BEA (Bryant et al., 2019), and use BEA-DEV for checkpoint selection. For pre-training we use an artificial GEC dataset C4_200M of 200M sentences (Stahlberg and Kumar, 2021). We then fine-tune on cLang-8 (Rothe et al., 2021), a distilled version of the Lang-8 learners corpus (Mizumoto et al., 2011).

Setup. We report *ERRANT* F0.5 scores for BEA. We report additional gT5/gFelix baseline numbers from Rothe et al. (2020b), where T5/Felix models were trained only on cLang-8. For pre-training we sampled 0.2% examples from the training set to use as a development set, and train till convergence as measured on this development set.

We additionally measure the impact that model size has on quality and latency, training T5 and EDITS small, base, and large models. To make the latency comparison fairer, we also train single-decoder-layer variants of the T5 models we call T5 Slim. To further ensure a fair latency comparison between EDITS and T5 we use the same framework for both models. Additionally, we do not perform EDITS specific pre-training.

Results. From Table 4, we see that all models outperform their equivalent gT5/gFelix models, which is not surprising as the latter models were trained on less data. A surprising result is that the T5 slim variants achieve comparable scores to the full T5 models while having significantly lower latency. Comparing EDITS against T5 models, we see up to ∼1 point differences in F0.5 scores between models of the same size (small/base/large), however EDITS produces speed ups between 10x and 25x.

In Figure 4, we study the latency–quality trade-offs of T5, T5 slim, and EDITS models. We omit Felix from this analysis, because Felix achieves a significantly lower score. We focus on the 95 percentile latency, as it is often the case that users require that a model returns a result within a fixed latency budget. We see that EDITS drops less than 0.25 F0.5 points comparing across model sizes, whilst being significantly faster. Additionally for a given latency budget of 5ms, no full T5 model would fit, and only the T5 slim small would fit, whereas both EDITS small and base fit. Comparing EDITS base against T5 slim small, we see that EDITS scores 3 F0.5 points higher, whilst being faster. For any latency budget under 20ms, EDITS is quicker and offer better results than T5 and T5 slim. For latency budgets above 20ms, T5 slim large scores slightly (<0.25 F0.5) higher than EDITS, and if latency is not a factor then gT5 xxl should be used.

Model	#Params	F0.5	Mean	Median	95%	Speed Up
gT5 small	76M	65.01	-	-	-	-
gT5 base	248M	69.39	-	-	-	-
gT5 large	783M	72.06	-	-	-	-
gT5 xxl	11B	**75.88**	-	-	-	-
gFelix base	220M	59.05	-	-	-	-
T5 small	76M	69.79	10.5	9.2	21.0	3.5x
T5 base	248M	72.39	35.5	31.2	74.1	1.0x
T5 large	783M	73.43	92.4	81.3	184.8	0.4x
T5 slim small	55M	68.50	2.6	2.3	5.1	14.5x
T5 slim base	144M	71.78	4.7	4.3	8.7	8.5x
T5 slim large	391M	73.18	11.1	10.1	20.0	3.7x
Felix base	220M	63.50	1.8	1.8	1.8	41.2x
EDITS small	50M	68.40	**0.9**	**0.8**	**1.3**	**57.0x**
EDITS base	141M	71.58	1.8	1.6	2.5	29.6x
EDITS large	391M	72.93	4.1	3.9	6.6	11.2x

Table 4: GEC F0.5 results for gT5, gFelix, T5, T5 slim, Felix, and EDITS; number of parameters; mean, mode and 95 percentile latencies (in milliseconds); we also present *speed up*, the ratio of 95 percentile latency to T5 base.

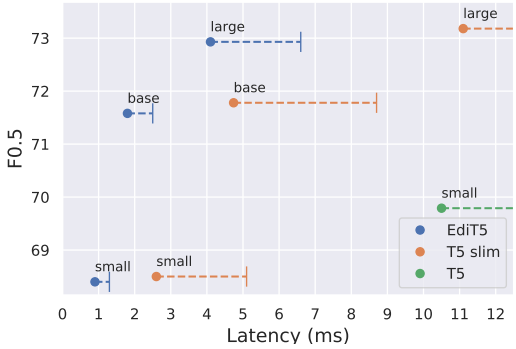

Figure 4: Mean and 95% percentile latency for T5, T5 slim and EDITS across model sizes on BEA.

4 Latency analysis

The tasks on which EDITS outperforms seq2seq models in latency are those that have overlap be-

tween sources and targets, but it's unclear how much overlap is required for EDITT5 to produce latency savings. To answer this question, we split EDITT5 base, T5 base and T5 slim base into components whose latencies we measure separately and compare. Details on how latencies are measured can be found in the Appendix C.

A seq2seq model decomposes into two parts: the encoder (we include the input embedding here, so we refer to this as encoder* below), and the decoder. EDITT5 has both of these parts, but also includes a third part (which we call its overhead), comprising of pointer realization and additional transformer layers. To make our analysis simpler and more task-agnostic, we make two simplifying assumptions. First, we assume the worst-case that no tokens are deleted by EDITT5 and there are no padding tokens in the input[2], in practice this is not the case, and provides significant latency savings for EDITT5. Second, we assume that decoder latency is linear in the number of decoder steps[3]. Both of these assumptions benefit the latency of seq2seq models more than EDITT5.

Results. In Table 5 we present latencies of encoder*, worst-case EDITT5 overhead and the per-step latency of a decoder under various input-length conditions. We see the overhead added by EDITT5 even in the worst-case is small.

From these results we can derive a simple rule for when EDITT5 will provide a net latency benefit. Compared to T5 slim base[4], EDITT5 base must save on average 4 decoder steps with an input length of 128, and 7 steps with an input length of 512.

Finally, collating the results in Table 5 with the number of decoder steps performed by EDITT5 and T5 in Table 1, we see that whereas in T5 the decoder latency dominates the latency of encoder*, in EDITT5 this is no longer the case. For instance for GEC, at 24.7 decoder steps on average required to construct the output, T5 slim spends 3.7x more time in its decoder than in encoder*. EDITT5 however spends less time in its decoder than in encoder*, as such the encoder* is now the latency bottleneck.

Component	Len. 128	Len. 512
Encoder*	0.98	2.65
Worst-case EDITT5 overhead	0.49	1.16
1 layer decoder per-step	0.15	0.17
12 layer decoder per-step	1.26	1.47

Table 5: Mean latencies (in milliseconds, $\pm$ 0.01ms) measured for the components of EDITT5 and T5 models for various input lengths. EDITT5 overhead is normally input dependent, but we estimate worst-case latency.

5 Related work

T5 (Raffel et al., 2020) is a pre-trained, Transformer-based (Vaswani et al., 2017) encoder-decoder model which has become a general-purpose tool for a variety of sequence-transduction tasks, establishing many new state-of-the-art results (Raffel et al., 2020; Rothe et al., 2021). However, two considerable challenges hindering the productionizing of T5-based models are the high latency caused by autoregressive decoding and the need for having a relatively large number of training examples despite the fact that pre-training makes T5 more sample efficient. Recently, it has been found that the sample efficiency problem can be mitigated by performing in-context few-shot learning, but this typically requires scaling up the model size even further (Brown et al., 2020; Chowdhery et al., 2022), increasing the latency.

To reduce latency, a number of non-autoregressive (NAT) seq2seq methods have been proposed for neural machine translation (Gu et al., 2018, 2019; Du et al., 2021) but a quality gap compared to autoregressive methods still exists. To decrease the gap, it is common to run the NAT methods iteratively, which, however, limits the inference speed advantage over autoregressive methods (Lee et al., 2018). In contrast, we show that for tasks where inputs and outputs overlap, we can maintain an order-of-magnitude speed-up without compromising on the model quality by treating the problem as a text-editing task and producing the output in a single pass.

A number of text-editing models have been proposed as a faster and more sample efficient alternative to seq2seq models like T5 (Awasthi et al., 2019; Malmi et al., 2019; Omelianchuk et al., 2020; Mallinson et al., 2020). Another recently proposed approach to speed up the inference time of Transformer models is called *aggressive decoding* (Sun et al., 2021; Ge et al., 2022).

Closest to our work, Mallinson et al. (2020) show that adding pointing mechanism for reordering and a separate insertion model allow their text-editing model, FELIX, to produce an arbitrary output in

[2]The pointer realization runs for exactly input-length steps.

[3]This ignores decoder self-attention, but is justified when the number of decoder steps is small.

[4]The overhead is smaller than 1 step of T5 base.

a flexible manner. FELIX is a non-autoregressive model which first predicts the tokens to keep, their order, and the locations at which to insert new tokens. Then it runs a separate model based on a BERT masked language model for inserting new tokens. In contrast, EDIT5 employs a single, end-to-end model which has an autoregressive insertion component. This enables more accurate insertions, while keeping the latency low, given that most of the tokens can be copied from the source non-autoregressively. Other text-editing models that employ autoregressive insertion include EditNTS (Dong et al., 2019), the text-normalization model by Zhang et al. (2019), Seq2Edits (Stahlberg and Kumar, 2020), ESC (Chen et al., 2020) and LEWIS (Reid and Zhong, 2021). However, unlike EDIT5, these models perform also the edit operation prediction autoregressively, making them potentially slower at inference time.

6 Conclusions

In this paper we have proposed EDIT5 a low latency solution to text generation, that achieves comparable or betters results, across three distinct tasks, to a strong T5 baseline whilst achieving inference latencies that are up to 25x quicker than the baseline model.

In the future we wish to explore the following ideas: 1) The impact of distillation for EDIT5. Distillation has previously been shown to be particularly advantageous to non-autoregressive models. 2) Exploring the impact that quantization has on both latency and quality. 3) Applying EDIT5 to additional languages. EDIT5 makes no language specific assumptions and we plan to apply it to languages other than English.

Limitations

A limitation of EDIT5, and text-editing models in general, is the assumption of overlapping text between the input and output sequences. For instance, in machine translation the overlap between source and target is minimal to none. As such EDIT5 would decode the entire target sequence, thus offering no latency saving.

An additional limitation is that all of our experiments were done on English tasks. It is unclear how EDIT5's pointing mechanism would behave with languages which have a less strict word-order, such as Czech.

Finally, we have measured latency only on V4 TPUs, and thus it is unclear how the performance would behave on different graphics cards or on CPUs. As such to determine if EDIT5 offers a good trade-off between quality and latency, one must measure latency on the target device.

Acknowledgement

We thank Sebastian Krause, Sascha Rothe, and Hongkun Yu for useful discussions, suggestions and feedback. We also thank Shankar Kumar and Felix Stahlberg for providing feedback on an earlier draft of the paper.

References

Abhijeet Awasthi, Sunita Sarawagi, Rasna Goyal, Sabyasachi Ghosh, and Vihari Piratla. 2019. Parallel iterative edit models for local sequence transduction. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4260–4270, Hong Kong, China. Association for Computational Linguistics.

Eyal Ben-David, Orgad Keller, Eric Malmi, Idan Szpektor, and Roi Reichart. 2020. Semantically driven sentence fusion: Modeling and evaluation. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1491–1505, Online. Association for Computational Linguistics.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe. 2019. The BEA-2019 shared task on grammatical error correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 52–75, Florence, Italy. Association for Computational Linguistics.

Mengyun Chen, Tao Ge, Xingxing Zhang, Furu Wei, and Ming Zhou. 2020. Improving the efficiency of grammatical error correction with erroneous span detection and correction. In *Proceedings of the 2020*

Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 7162–7169, Online. Association for Computational Linguistics.

Eunsol Choi, Jennimaria Palomaki, Matthew Lamm, Tom Kwiatkowski, Dipanjan Das, and Michael Collins. 2021. Decontextualization: Making sentences stand-alone. *Transactions of the Association for Computational Linguistics*, 9:447–461.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. PaLM: Scaling Language Modeling with Pathways. *arXiv preprint arXiv:2204.02311*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Yue Dong, Zichao Li, Mehdi Rezagholizadeh, and Jackie Chi Kit Cheung. 2019. EditNTS: An neural programmer-interpreter model for sentence simplification through explicit editing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3393–3402, Florence, Italy. Association for Computational Linguistics.

Cunxiao Du, Zhaopeng Tu, and Jing Jiang. 2021. Order-agnostic cross entropy for non-autoregressive machine translation. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 2849–2859. PMLR.

Tao Ge, Heming Xia, Xin Sun, Si-Qing Chen, and Furu Wei. 2022. Lossless acceleration for seq2seq generation with aggressive decoding. *arXiv preprint arXiv:2205.10350*.

Mor Geva, Eric Malmi, Idan Szpektor, and Jonathan Berant. 2019. DiscoFuse: A large-scale dataset for discourse-based sentence fusion. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3443–3455, Minneapolis, Minnesota. Association for Computational Linguistics.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Jiatao Gu, Changhan Wang, and Junbo Zhao. 2019. Levenshtein transformer. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 11179–11189.

Xianzhi Du Yeqing Li Abdullah Rashwan Le Hou Pengchong Jin Fan Yang Frederick Liu Jaeyoun Kim Hongkun Yu, Chen Chen and Jing Li. 2020. TensorFlow Model Garden. `https://github.com/tensorflow/models`.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, Brussels, Belgium. Association for Computational Linguistics.

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*, pages 1173–1182, Brussels, Belgium. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020a. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020b. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Jonathan Mallinson, Aliaksei Severyn, Eric Malmi, and Guillermo Garrido. 2020. FELIX: Flexible text editing through tagging and insertion. In *Findings of the Association for Computational Linguistics: EMNLP*

2020, pages 1244–1255, Online. Association for Computational Linguistics.

Eric Malmi, Yue Dong, Jonathan Mallinson, Aleksandr Chuklin, Jakub Adamek, Daniil Mirylenka, Felix Stahlberg, Sebastian Krause, Shankar Kumar, and Aliaksei Severyn. 2022. Text generation with text-editing models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Tutorial Abstracts*, pages 1–7.

Eric Malmi, Sebastian Krause, Sascha Rothe, Daniil Mirylenka, and Aliaksei Severyn. 2019. Encode, tag, realize: High-precision text editing. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5054–5065, Hong Kong, China. Association for Computational Linguistics.

Gonzalo E. Mena, David Belanger, Scott W. Linderman, and Jasper Snoek. 2018. Learning latent permutations with gumbel-sinkhorn networks. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

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, Chiang Mai, Thailand. Asian Federation of Natural Language Processing.

Kostiantyn Omelianchuk, Vitaliy Atrasevych, Artem Chernodub, and Oleksandr Skurzhanskyi. 2020. GECToR – grammatical error correction: Tag, not rewrite. In *Proceedings of the Fifteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 163–170, Seattle, WA, USA → Online. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

Machel Reid and Victor Zhong. 2021. LEWIS: Levenshtein editing for unsupervised text style transfer. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 3932–3944, Online. Association for Computational Linguistics.

Adam Roberts, Hyung Won Chung, Anselm Levskaya, Gaurav Mishra, James Bradbury, Daniel Andor, Sharan Narang, Brian Lester, Colin Gaffney, Afroz Mohiuddin, Curtis Hawthorne, Aitor Lewkowycz, Alex Salcianu, Marc van Zee, Jacob Austin, Sebastian Goodman, Livio Baldini Soares, Haitang Hu, Sasha Tsvyashchenko, Aakanksha Chowdhery, Jasmijn Bastings, Jannis Bulian, Xavier Garcia, Jianmo Ni, Andrew Chen, Kathleen Kenealy, Jonathan H. Clark, Stephan Lee, Dan Garrette, James Lee-Thorp, Colin Raffel, Noam Shazeer, Marvin Ritter, Maarten Bosma, Alexandre Passos, Jeremy Maitin-Shepard, Noah Fiedel, Mark Omernick, Brennan Saeta, Ryan Sepassi, Alexander Spiridonov, Joshua Newlan, and Andrea Gesmundo. 2022. Scaling up models and data with `t5x` and `seqio`. *arXiv preprint arXiv:2203.17189*.

Sascha Rothe, Jonathan Mallinson, Eric Malmi, Sebastian Krause, and Aliaksei Severyn. 2021. A simple recipe for multilingual grammatical error correction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 702–707, Online. Association for Computational Linguistics.

Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. 2020a. Leveraging pre-trained checkpoints for sequence generation tasks. *Transactions of the Association for Computational Linguistics*, 8:264–280.

Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. 2020b. Leveraging pre-trained checkpoints for sequence generation tasks. *Transactions of the Association for Computational Linguistics*, 8:264–280.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4603–4611. PMLR.

Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. 2006. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231, Cambridge, Massachusetts, USA. Association for Machine Translation in the Americas.

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. MASS: masked sequence to sequence pre-training for language generation. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 5926–5936. PMLR.

Felix Stahlberg and Shankar Kumar. 2020. Seq2Edits: Sequence transduction using span-level edit operations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5147–5159, Online. Association for Computational Linguistics.

Felix Stahlberg and Shankar Kumar. 2021. Synthetic data generation for grammatical error correction with tagged corruption models. In *Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 37–47, Online. Association for Computational Linguistics.

Xin Sun, Tao Ge, Furu Wei, and Houfeng Wang. 2021. Instantaneous grammatical error correction with shallow aggressive decoding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5937–5947, Online. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2692–2700.

Wei Xu, Courtney Napoles, Ellie Pavlick, Quanze Chen, and Chris Callison-Burch. 2016. Optimizing statistical machine translation for text simplification. *Transactions of the Association for Computational Linguistics*, 4:401–415.

Hao Zhang, Richard Sproat, Axel H. Ng, Felix Stahlberg, Xiaochang Peng, Kyle Gorman, and Brian Roark. 2019. Neural models of text normalization for speech applications. *Computational Linguistics*, 45(2):293–337.

A Alignment Algorithm

Algorithm 1: EDIT5 Alignment

```
Data: source ; // List of tokens
Data: target ; // List of characters
Result: alignments
buffer ← ∅
alignments ← []
i ← 0
while i < len(target) do
    max_length ← 0
    max_index ← 0
    j ← i + 1
    while j < len(target) do
        source_index, overlap_length ←
          contiguous_length(target[i:j],source)
        if overlap_length > max_length then
            max_length ← overlap_length
            max_index ← source_index
        j ← j + 1
    if max_overlap_length > 0 then
        source[max_index] ← ∅
        alignment ← (i,j,max_index,buffer)
        alignments.append(alignment)
        buffer ← ∅
        i ← j + 1
    else
        buffer ← buffer + target[i]
        i ← i + 1
```

B Training Details

All models were trained on 4x4 or 8x8 TPUs, all EDIT5 models completed training (including EDIT5 pre-training) in under a day. T5 large pre-training large took 2 days to complete and was done using a 4x4 TPU.

B.1 Hyper-Parameters Selection

For T5 we compared the T5 1.0 and T5 1.1 version using the base model on the validation sets and found that T5 1.1 performed better, as such used T5 1.1. For EDIT5 we used the BEA dev set, finding that T5 1.0 base performed better than T5 1.1 and selected 1.0 for all experiments.

For T5 we used the recommend fine-tuning settings, including using the adafactor optimizer (Shazeer and Stern, 2018), with a learning rate of 0.001. For EDIT5 we used AdamW with default settings and the default learning rate of 3e-4.

DiscoFuse. For both EDIT5 and T5 we experimented with 3 different batch sizes 128, 256, 1024. For 100% and 10%, there was not a noticeable difference in the DEV set exact match performance, so we chose 1024 as it converged the quickest. For 1% and lower, we found that a batch size of 128 performed the best on the dev set.

Decontextualization. For EDIT5 we experimented with the batch size 128, 256, 1024 and found that 256 offered the best exact match and used this. We also slightly modified the pre-processing code, bracketing the target sequence with [CLS] and [SEP], which helped the alignment code.

GEC. For both EDIT5 and T5 we used the T5 recommended number of tokens per batch of: batch size = 512, maximum sequence length = 128. We however note that T5 used the inverse: batch size = 128, maximum sequence length = 512. For T5 and EDIT5 we disabled learning rate warmup when fine-tuning on cLang-8. Two additional hyperparameters were set for EDIT5, during pre-training on C4_200M, we noted that EDIT5 train set performance was lower than T5, as such we disabled

dropout on the additional EDITT5 specific transformer layers. We additionally used the dev set to set the values of lambda for equation 10. We experimented with tagging/pointing λ being 1, 2, 10, or equal to the number of tokens. Where λ equal to the number of tokens produced the best results.

C Latency measurement

To report latency for a model, we run inference on a Cloud TPU V4 chip with batch size 1 and report the time spent in computations on the device. This approach ignores some practical contributors to latency, such as memory transfers between the host and device, but we found it also reduced noise significantly, while focusing on the main performance differences between EDITT5, T5 and T5 slim (the amount of computation they each perform). To further minimize spurious latency differences, both EDITT5 and the baseline models are based on the same T5 implementation, found in TensorFlow Model Garden (Hongkun Yu and Li, 2020).