Encode, Tag, Realize: High-Precision Text Editing

Eric Malmi
Google Research
emalmi@google.com

Sebastian Krause
Google Research
bastik@google.com

Sascha Rothe
Google Research
rothe@google.com

Daniil Mirylenka
Google Research
dmirylenka@google.com

Aliaksei Severyn
Google Research
severyn@google.com

Abstract

We propose LASERTAGGER—a sequence tagging approach that casts text generation as a text editing task. Target texts are reconstructed from the inputs using three main edit operations: *keeping* a token, *deleting* it, and *adding* a phrase before the token. To predict the edit operations, we propose a novel model, which combines a BERT encoder with an autoregressive Transformer decoder. This approach is evaluated on English text on four tasks: sentence fusion, sentence splitting, abstractive summarization, and grammar correction. LASERTAGGER achieves new state-of-the-art results on three of these tasks, performs comparably to a set of strong seq2seq baselines with a large number of training examples, and outperforms them when the number of examples is limited. Furthermore, we show that at inference time tagging can be more than two orders of magnitude faster than comparable seq2seq models, making it more attractive for running in a live environment.

1 Introduction

Neural sequence-to-sequence (seq2seq) models provide a powerful framework for learning to translate source texts into target texts. Since their first application to machine translation (MT) (Sutskever et al., 2014) they have become the de facto approach for virtually every text generation task, including summarization (Tan et al., 2017), image captioning (Xu et al., 2015), text style transfer (Rao and Tetreault, 2018; Nikolov and Hahnloser, 2018; Jin et al., 2019), and grammatical error correction (Chollampatt and Ng, 2018; Grundkiewicz et al., 2019).

We observe that in some text generation tasks, such as the recently introduced *sentence splitting* and *sentence fusion* tasks, output texts highly overlap with inputs. In this setting, learning a seq2seq model to generate the output text from scratch seems intuitively wasteful. Copy mechanisms (Gu et al., 2016; See et al., 2017) allow for choosing between copying source tokens and generating arbitrary tokens, but although such hybrid models help with out-of-vocabulary words, they still require large training sets as they depend on output vocabularies as large as those used by the standard seq2seq approaches.

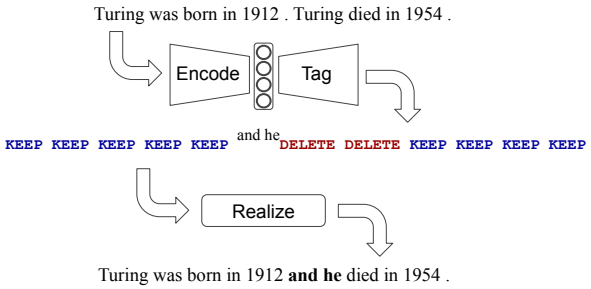

Figure 1: LASERTAGGER applied to sentence fusion.

In contrast, we propose learning a text editing model that applies a set of edit operations on the input sequence to reconstruct the output. We show that it is often enough to use a relatively small set of output tags representing text deletion, rephrasing and word reordering to be able to reproduce a large percentage of the targets in the training data. This results in a learning problem with a much smaller vocabulary size, and the output length fixed to the number of words in the source text. This, in turn, greatly reduces the number training examples required to train accurate models, which is particularly important in applications where only a small number of human-labeled data is available.

Our tagging approach, LASERTAGGER, consists of three steps (Fig. 1): (i) *Encode* builds a representation of the input sequence, (ii) *Tag* assigns edit tags from a pre-computed output vocabulary to the input tokens, and (iii) *Realize* applies a simple set of rules to convert tags into the output text tokens.

An experimental evaluation of LASERTAGGER

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 5054–5065,
Hong Kong, China, November 3–7, 2019. ©2019 Association for Computational Linguistics

on four different text generation tasks shows that it yields comparable results to seq2seq models when we have tens of thousands of training examples and clearly outperforms them when the number of examples is smaller.

Our contributions are the following:

1) We demonstrate that many text generation tasks with overlapping inputs and outputs can be effectively treated as text editing tasks.

2) We propose LASERTAGGER—a sequence tagging-based model for text editing, together with a method for generating the tag vocabulary from the training data.

3) We describe two versions of the tagging model: (i) LASERTAGGER$_{\text{FF}}$—a tagger based on BERT (Devlin et al., 2019) and (ii) LASERTAGGER$_{\text{AR}}$—a novel tagging model combining the BERT encoder with an autoregressive Transformer decoder, which further improves the results over the BERT tagger.

4) We evaluate LASERTAGGER against strong seq2seq baseline models based on the BERT architecture. Our baseline models outperform previously reported state-of-the-art results on two tasks.

5) We demonstrate that a) LASERTAGGER$_{\text{AR}}$ achieves state-of-the-art or comparable results on 3 out of 4 examined tasks, b) LASERTAGGER$_{\text{FF}}$ is up to *100x faster at inference time* with performance comparable to the state-of-the-art seq2seq models. Furthermore, both models: c) require much *less training data* compared to the seq2seq models, d) are more *controllable and interpretable* than seq2seq models due to the small vocabulary of edit operations, e) are *less prone* to typical seq2seq model errors, such as hallucination.

The code will be available at:
`lasertagger.page.link/code`

2 Related Work

Recent work discusses some of the difficulties of learning neural decoders for text generation (Wiseman et al., 2018; Prabhakaran et al., 2018). Conventional seq2seq approaches require large amounts of training data, are hard to control and to constrain to desirable outputs. At the same time, many NLP tasks that appear to be full-fledged text generation tasks are natural testbeds for simpler methods. In this section we briefly review some of these tasks.

Text Simplification is a paraphrasing task that is known to benefit from modeling edit operations. A simple instance of this type are sentence compression systems that apply a drop operation at the token/phrase level (Filippova and Strube, 2008; Filippova et al., 2015), while more intricate systems also apply splitting, reordering, and lexical substitution (Zhu et al., 2010). Simplification has also been attempted with systems developed for phrase-based MT (Xu et al., 2016a), as well as with neural encoder-decoder models (Zhang and Lapata, 2017).

Independent of this work, Dong et al. (2019) recently proposed a text-editing model, similar to ours, for text simplification. The main differences to our work are: (i) They introduce an interpreter module which acts as a language model for the so-far-realized text, and (ii) they generate added tokens one-by-one from a full vocabulary rather than from an optimized set of frequently added phrases. The latter allows their model to generate more diverse output, but it may negatively effect the inference time, precision, and the data efficiency of their model. Another recent model similar to ours is called Levenshtein Transformer Gu et al. (2019), which does text editing by performing a sequence of deletion and insertion actions.

Single-document summarization is a task that requires systems to shorten texts in a meaning-preserving way. It has been approached with deletion-based methods on the token level (Filippova et al., 2015) and the sentence level (Narayan et al., 2018; Liu, 2019). Other papers have used neural encoder-decoder methods (Tan et al., 2017; Rush et al., 2015; Paulus et al., 2017) to do *abstractive* summarization, which allows edits beyond mere deletion. This can be motivated by the work of Jing and McKeown (2000), who identified a small number of fundamental high-level editing operations that are useful for producing summaries (reduction, combination, syntactic transformation, lexical paraphrasing, generalization/specification, and reordering). See et al. (2017) extended a neural encoder-decoder model with a copy mechanism to allow the model to more easily reproduce input tokens during generation.

Out of available summarization datasets (Dernoncourt et al., 2018), we find the one by Toutanova et al. (2016) particularly interesting because (1) it specifically targets abstractive summarization systems, (2) the lengths of texts in this dataset (short paragraphs) seem well-suited for text editing, and (3) an analysis showed that the dataset covers many different summarization operations.

In **Grammatical Error Correction** (Ng et al.,

2013, 2014) a system is presented with input texts written usually by a language learner, and is tasked with detecting and fixing grammatical (and other) mistakes. Approaches to this task often incorporate task-specific knowledge, e.g., by designing classifiers for specific error types (Knight and Chander, 1994; Rozovskaya et al., 2014) that can be trained without manually labeled data, or by adapting statistical machine-translation methods (Junczys-Dowmunt and Grundkiewicz, 2014). Methods for the sub-problem of error *detection* are similar in spirit to sentence compression systems, in that they are implemented as word-based neural sequence labelers (Rei, 2017; Rei et al., 2017). Neural encoder-decoder methods are also commonly applied to the error correction task (Ge et al., 2018; Chollampatt and Ng, 2018; Zhao et al., 2019), but suffer from a lack of training data, which is why task-specific tricks need to be applied (Kasewa et al., 2018; Junczys-Dowmunt et al., 2018).

3 Text Editing as a Tagging Problem

Our approach to text editing is to cast it into a tagging problem. Here we describe its main components: (1) the tagging operations, (2) how to convert plain-text training targets into a tagging format, as well as (3) the realization step to convert tags into the final output text.

3.1 Tagging Operations

Our tagger assigns a tag to each input token. A tag is composed of two parts: a *base tag* and an *added phrase*. The base tag is either `KEEP` or `DELETE`, which indicates whether to retain the token in the output. The added phrase P, which can be empty, enforces that P is added before the corresponding token. P belongs to a vocabulary V that defines a set of words and phrases that can be inserted into the input sequence to transform it into the output.

The combination of the base tag B and the added phrase P is treated as a single tag and denoted by ${}^{P}B$. The total number of unique tags is equal to the number of base tags times the size of the phrase vocabulary, hence there are $\approx 2|V|$ unique tags.

Additional task-specific tags can be employed too. For sentence fusion (Section 5.1), the input consists of two sentences, which sometimes need to be swapped. Therefore, we introduce a custom tag, `SWAP`, which can only be applied to the last period of the first sentence (see Fig. 2). This tag instructs the Realize step to swap the order of the input sentences before realizing the rest of the tags.

For other tasks, different supplementary tags may be useful. E.g., to allow for replacing entity mentions with the appropriate pronouns, we could introduce a `PRONOMINALIZE` tag. Given an access to a knowledge base that includes entity gender information, we could then look up the correct pronoun during the realization step, instead of having to rely on the model predicting the correct tag (${}^{\text{she}}$`DELETE`, ${}^{\text{he}}$`DELETE`, ${}^{\text{they}}$`DELETE`, etc.).

3.2 Optimizing Phrase Vocabulary

The phrase vocabulary consists of phrases that can be added between the source words. On the one hand, we wish to minimize the number of phrases to keep the output tag vocabulary small. On the other hand, we would like to maximize the percentage of target texts that can be reconstructed from the source using the available tagging operations. This leads to the following combinatorial optimization problem.

Problem 1 *Given a collection of phrase sets $A_1, A_2, \ldots A_m$, where $A_i \subseteq P$ and P is the set of all candidate phrases, select a phrase vocabulary $V \subset P$ of at most ℓ phrases (i.e. $|V| \leq \ell$) so that the number of covered phrase sets is maximized. A phrase set A_i is covered if and only if $A_i \subseteq V$.*

This problem is closely related to the minimum k-union problem which is NP-hard (Vinterbo, 2002). The latter problem asks for a set of k phrase sets such that the cardinality of their union is the minimum. If we were able to solve Problem 1 in polynomial time, we could solve also the minimum k-union problem in polynomial time simply by finding the smallest phrase vocabulary size ℓ such that the number of covered phrase sets is at least k. This reduction from the minimum k-union problem gives us the following result:

Theorem 1 *Problem 1 is NP-hard.*

To identify candidate phrases to be included in the vocabulary, we first align each source text s from the training data with its target text t. This is achieved by computing the *longest common subsequence* (LCS) between the two word sequences, which can be done using dynamic programming in time $\mathcal{O}(|s| \times |t|)$. The n-grams in the target text that are not part of the LCS are the phrases that would need to be included in the phrase vocabulary to be able to construct t from s.

Source:	Dylan	won	Nobel	prize	.	Dylan	is		an	American	musician	.	
Tags:	DELETE	KEEP	KEEP	KEEP	SWAP	KEEP	^{comma}DELETE		KEEP	KEEP	KEEP	^{comma}DELETE	
Realization:	Dylan , an American musician , won Nobel prize .												

Figure 2: An example sentence fusion obtained by tagging using the SWAP tag, which swaps the order of the two source sentences.

In practice, the phrase vocabulary is expected to consist of phrases that are frequently added to the target. Thus we adopt the following simple approach to construct the phrase vocabulary: sort the phrases by the number of phrase sets in which they occur and pick ℓ most frequent phrases. This was found to produce meaningful phrase vocabularies based on manual inspection as shown in Section 5. E.g., the top phrases for sentence fusions include many discourse connectives.

We also considered a greedy approach that constructs the vocabulary one phrase at a time, always selecting the phrase that has the largest incremental coverage. This approach is not, however, ideal for our use case, since some frequent phrases, such as " (" and ") ", are strongly coupled. Selecting " (" alone has close to zero incremental coverage, but together with ") ", they can cover many examples.

3.3 Converting Training Targets into Tags

Once the phrase vocabulary is determined, we can convert the target texts in our training data into tag sequences. Given the phrase vocabulary, we do not need to compute the LCS, but can leverage a more efficient approach, which iterates over words in the input and greedily attempts to match them (1) against the words in the target, and in case there is no match, (2) against the phrases in the vocabulary V. This can be done in $\mathcal{O}(|s| \times n_p)$ time, where n_p is the length of the longest phrase in V, as shown in Algorithm 1.

The training targets that would require adding a phrase that is not in our vocabulary V, will not get converted into a tag sequence but are filtered out. While making the training dataset smaller, this may effectively also filter out low-quality targets. The percentage of converted examples for different datasets is reported later in Section 5. Note that even when the target cannot be reconstructed from the inputs using our output tag vocabulary, our approach might still produce reasonable outputs with the available phrases. E.g., a target may require the use of the infrequent ";" token, which is not in our vocabulary, but a model could instead choose to predict a more common "," token.

Algorithm 1 Converting a target string to tags.

Input: Source text $s = [s(1), \ldots, s(n_s)]$, target text $t = [t(1), \ldots, t(n_t)]$, phrase vocabulary V, and the maximum added phrase length n_p.
Output: Tag sequence x of length n_s or of length 0 if conversion is not possible.

$x(i) =$ DELETE, $\forall i = 1, \ldots, n_s$ ▷ Initialize tags.
$i_s = 1$ ▷ Current source word index.
$i_t = 1$ ▷ Current target word index.
while $i_t \leq n_t$ **do**
  if $i_s > n_s$ **then**
    return [] ▷ Conversion infeasible.
  if $s(i_s) == t(i_t)$ **then**
    $x(i_s) =$ KEEP
    $i_t = i_t + 1$
  else
    $p = []$ ▷ Added phrase (word sequence).
    match_found $= 0$
    for $j = 1, \ldots, n_p$ **do**
      p.append($t(i_t + j - 1)$)
      if $s(i_s) == t(i_t + j)$ **and** $p \in V$ **then**
        match_found $= 1$
        break
    if match_found **then**
      $x(i_s) = {}^{p}$KEEP
      $i_t = i_t + |p| + 1$
  $i_s = i_s + 1$
return x ▷ Target has been consumed, so return tags.

3.4 Realization

After obtaining a predicted tag sequence, we convert it to text ("realization" step). While classic works on text generation make a distinction between *planning* and *realization*, end-to-end neural approaches typically ignore this distinction, with the exception of few works (Moryossef et al., 2019; Puduppully et al., 2018).

For the basic tagging operations of keeping, deleting, and adding, realization is a straightforward process. Additionally, we adjust capitalization at sentence boundaries. Realization becomes more involved if we introduce special tags, such as PRONOMINALIZE mentioned in Section 3.1. For this tag, we would need to look up the gender of the tagged entity from a knowledge base. Having a separate realization step is beneficial, since we can decide to pronominalize only when confident about the appropriate pronoun and can, otherwise, leave the entity mention untouched.

Another advantage of having a separate realiza-

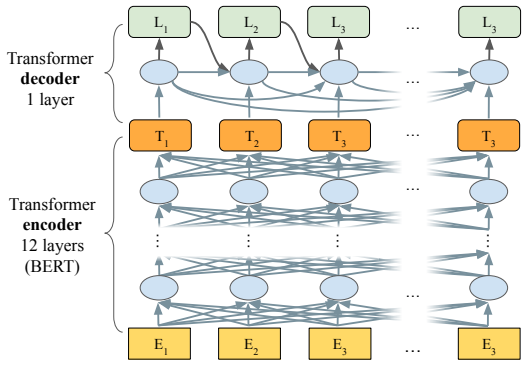

Figure 3: The architecture of LASERTAGGER$_{\text{AR}}$.

tion step is that specific loss patterns can be addressed by adding specialized realization rules. For instance, one could have a rule that when applying tag $^{\text{his}}$DELETE to an entity mention followed by *'s*, the realizer must always DELETE the possessive *'s* regardless of its predicted tag.

4 Tagging Model Architecture

Our tagger is composed of two components: an encoder, which generates activation vectors for each element in the input sequence, and a decoder, which converts encoder activations into tag labels.

Encoder. We choose the BERT Transformer model (Devlin et al., 2019) as our encoder, as it demonstrated state-of-the-art results on a number of sentence encoding tasks. We use the BERT-base architecture, which consists of 12 self-attention layers. We refer the reader to (Devlin et al., 2019) for a detailed description of the model architecture and its input representation. We initialize the encoder with a publicly available checkpoint of the pretrained case-sensitive BERT-base model.[1]

Decoder. In the original BERT paper a simple decoding mechanism is used for sequence tagging: the output tags are generated in a single feed-forward pass by applying an argmax over the encoder logits. In this way, each output tag is predicted independently, without modelling the dependencies between the tags in the sequence. Such a simple decoder demonstrated state-of-the-art results on the Named Entity Recognition task, when applied on top of the BERT encoder.

To better model the dependencies between the output tag labels, we propose a more powerful autoregressive decoder. Specifically, we run a single-layer Transformer decoder on top of the BERT encoder (see Fig. 3). At each step, the decoder is consuming the embedding of the previously predicted label and the activations from the encoder.

There are several ways in which the decoder can communicate with the encoder: (i) through a full attention over the sequence of encoder activations (similar to conventional seq2seq architectures); and (ii) by directly consuming the encoder activation at the current step. In our preliminary experiments, we found the latter option to perform better and converge faster, as it does not require learning additional encoder-decoder attention weights.

We experiment with both decoder variants (*feedforward* and *autoregressive*) and find that the autoregressive decoder outperforms the previously used *feedforward* decoder. In the rest of this paper, the tagging model with an *autoregressive* decoder is referred to as LASERTAGGER$_{\text{AR}}$ and the model with *feedforward* decoder as LASERTAGGER$_{\text{FF}}$.

5 Experiments

We evaluate our method by conducting experiments on four different text editing tasks: Sentence Fusion, Split and Rephrase, Abstractive Summarization, and Grammatical Error Correction.

Baselines. In addition to reporting previously published results for each task, we also train a set of strong baselines based on Transformer where both the encoder and decoder replicate the BERT-base architecture (Devlin et al., 2019). To have a fair comparison, similar to how we initialize a tagger encoder with a pretrained BERT checkpoint, we use the same initialization for the Transformer encoder. This produces a very strong seq2seq baseline (SEQ2SEQ$_{\text{BERT}}$), which already results in new state-of-the-art metrics on two out of four tasks.

5.1 Sentence Fusion

Sentence Fusion is the problem of fusing sentences into a single coherent sentence.

Data. We use the "balanced Wikipedia" portion of Geva et al. (2019)'s DiscoFuse dataset for our experiments (henceforth DfWiki). Out of the 4.5M fusion examples in the dataset, 10.5% require reordering of the input. To cope with this, we introduce the SWAP tag, which enables the model to flip the order of two input sentences. We construct the phrase vocabulary as described in Sec. 3.2 using the validation set of 46K examples. The top 15 phrases are shown in the first column of Table 1.

Evaluation metrics. Following Geva et al. (2019), we use two evaluation metrics: *Exact score*,

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

DfWiki	WikiSplit	AS	GEC
,	.␣⟨::::⟩	,	,
and	,	.	.
however␣,	.␣⟨::::⟩␣he	the	the
,␣but	.␣⟨::::⟩␣it	a	a
he	the	&	to
because	and	and	in
,␣although	was	is	of
but	is	in	on
,␣and	"	"	at
although	.␣⟨::::⟩␣she	's	for
his	.␣⟨::::⟩␣it␣is	with	have
,␣while	a	for	is
it	.␣⟨::::⟩␣they	of	was
,␣which	.␣⟨::::⟩␣however	n't	and
she	he	an	that

Table 1: The 15 most frequently added phrases in the datasets studied in this work, in order of decreasing frequency. ⟨::::⟩ marks a sentence boundary. "AS"/"GEC" is short for Abstractive Summarization/Grammatical Error Correction.

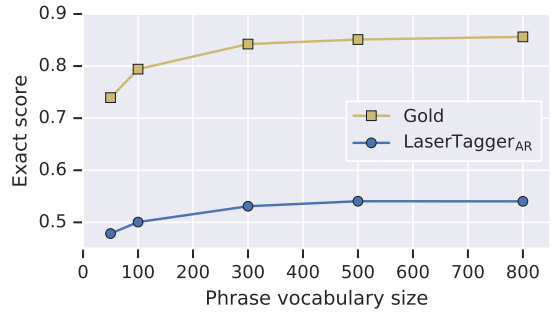

Figure 4: Performance of model LaserTagger$_{AR}$ on the DfWiki dataset, conditioned on the vocabulary size and the gold score, i.e. the percentage of examples that can be reconstructed via text-edit operations.

which is the percentage of exactly correctly predicted fusions, and *SARI* (Xu et al., 2016b), which computes the average F1 scores of the added, kept, and deleted n-grams.[2]

Vocabulary Size. To understand the impact of the number of phrases we include in the vocabulary, we trained models for different vocabulary sizes (only LaserTagger$_{AR}$). The results are shown in Figure 4. After increasing the vocabulary size to 500 phrases, Exact score reaches a plateau, so we set the vocabulary size to 500 in all the remaining experiments of this paper.[3] The Gold curve in Fig. 4 shows that this vocabulary size is sufficient

[2]We use the implementation available at git.io/fj8Av, setting $\beta = 1$ for deletion (Geva et al., 2019).

[3]For smaller datasets, a smaller vocabulary size may yield better results, but for simplicity, we do not optimize the size separately for each dataset.

Model	Exact	SARI
Transformer (Geva et al., 2019)	51.1	84.5
SEQ2SEQ$_{BERT}$	53.6	85.3
LaserTagger$_{AR}$ (no SWAP)	46.4	80.4
LaserTagger$_{FF}$	52.2	84.1
LaserTagger$_{AR}$	**53.8**	**85.5**

Table 2: Sentence fusion results on DfWiki.

to cover 85% of the training examples, which gives us an upper bound for the Exact score.

Comparison against Baselines. Table 2 lists the results for the DfWiki dataset. We obtain new SOTA results with LaserTagger$_{AR}$, outperforming the previous SOTA 7-layer Transformer model from Geva et al. (2019) by 2.7% Exact score and 1.0% SARI score. We also find that the pretrained SEQ2SEQ$_{BERT}$ model yields nearly as good performance, demonstrating the effectiveness of unsupervised pretraining for generation tasks. The performance of the tagger is impaired significantly when leaving out the SWAP tag due to the model's inability to reconstruct 10.5% of the training set.

Impact of Dataset Size. We also study the effect of the training data size by creating four increasingly smaller subsets of DfWiki (see Fig. 5a).[4] When data size drops to 450 or 4 500 examples, LaserTagger still performs surprisingly well, clearly outperforming the SEQ2SEQ$_{BERT}$ baseline.

5.2 Split and Rephrase

The reverse task of sentence fusion is the *split-and-rephrase* task, which requires rewriting a long sentence into two or more coherent short sentences.

Data. We use the WikiSplit dataset (Botha et al., 2018), which consists of 1M human-editor created examples of sentence splits, and follow the dataset split suggested by the authors. Using the phrase vocabulary of size 500 yields a 31% coverage of the targets from the training set (top phrases shown in Table 1). The lower coverage compared to DfWiki suggests a higher amount of noise (due to Wikipedia-author edits unrelated to splitting).

Results. Botha et al. (2018) report results using a one-layer, bi-directional LSTM (cell size 512) with attention and a copying mechanism (See

[4]For simplicity, we use the same phrase vocabulary of size 500 computed using the validation set of 46K examples for all experiments. Note that even though some subsampled training sets contain less than 46K examples, using the same vocabulary does not give the taggers an unfair advantage over the baselines, because the tagger will never predict a phrase it has not seen in the training data.

(a) Sentence Fusion on DfWiki.

(b) Split and Rephrase on WikiSplit.

Figure 5: SARI score as a function of the training-data size for three models. Unless we have tens of thousands of training examples, the tagging approach clearly outperforms the seq2seq baseline.

Model	BLEU	Exact	SARI
seq2seq (Botha et al., 2018)	76.0	14.6	60.6
SEQ2SEQ$_{BERT}$	**76.7**	15.1	**62.3**
LASERTAGGER$_{FF}$	76.0	14.4	61.3
LASERTAGGER$_{AR}$	76.3	**15.2**	61.7

Table 3: Results on the WikiSplit dataset.

Model	BLEU-4	Exact	SARI	ROUGE-L
Filippova et al. (2015)	26.7	0.0	36.2	70.3
Clarke and Lapata (2008)	28.5	0.3	41.5	77.5
Cohn and Lapata (2008)	5.1	0.1	27.4	40.7
Rush et al. (2015)	16.2	0.0	35.6	62.5
SEQ2SEQ$_{BERT}$	8.3	0.1	32.1	52.7
LASERTAGGER$_{FF}$	33.7	1.5	44.2	81.9
LASERTAGGER$_{AR}$	**35.6**	**3.8**	**44.8**	**82.8**

Table 4: Results on summarization.

et al., 2017).[5] The results are shown in Table 3. SEQ2SEQ$_{BERT}$ and LASERTAGGER$_{AR}$ yield similar performance with each other, and they both outperform the seq2seq model with a copying mechanism from Botha et al. (2018).

We again studied the impact of training-data size by subsampling the training set, see Figure 5b. Similar to the previous experiment, the LASERTAGGER methods degrade more gracefully when reducing training-data size, and start to outperform the seq2seq baseline once going below circa 10k examples. The smallest training set for LASERTAGGER$_{AR}$ contains merely 29 examples. Remarkably, the model is still able to learn something useful that generalizes to unseen test examples, reaching a SARI score of 53.6% and predicting 5.2% of the targets exactly correctly. The following is an example prediction by the model:

Source: Delhi Public Library is a national depository library in Delhi , India , it has over 35 branches across the state .

Prediction: Delhi Public Library is a national depository library in Delhi , India . ⟨::::⟩ It has over 35 branches across the state .

Here the model has picked the right comma to replace with a period and a sentence separator.

[5]Botha et al. (2018) report only BLEU but they kindly shared with us their model's predictions, allowing us to compute the Exact and SARI score for their method. Similar to their work, we used NLTK v3.2.2 for the BLEU computation.

5.3 Abstractive Summarization

The task of summarization is to reduce the length of a text while preserving its meaning.

Dataset. We use the dataset from Toutanova et al. (2016), which contains 6,168 short input texts (one or two sentences) and one or more human-written summaries. The human experts were not restricted to just deleting words when generating a summary, but were allowed to also insert new words and reorder parts of the sentence, which makes this dataset particularly suited for *abstractive summarization* models.

We set the size of the phrase vocabulary to 500, as for the other tasks, and extract the phrases from the training partition. With a size of 500, we are able to cover 89% of the training data.

Evaluation Metrics. In addition to the metrics from the previous sections, we report ROUGE-L (Lin, 2004), as this is a metric that is commonly used in the summarization literature. ROUGE-L is a recall-oriented measure computed as the longest common sub-sequence between a reference summary and a candidate summary.

Results. Table 4 compares our taggers against seq2seq baselines and systems from the literature.[6] Filippova et al. (2015) and Clarke and Lapata (2008) proposed deletion-based approaches;

[6]Results are extracted from Toutanova et al. (2016).

the former uses a seq2seq network, the latter formulates summarization as an optimization problem that is solved via integer-linear programming. Cohn and Lapata (2008) proposed an early approach to abstractive summarization via a parse-tree transducer. Rush et al. (2015) developed a neural seq2seq model for abstractive summarization.

In line with the results on the subsampled fusion/splitting datasets (Figure 5), the tagger significantly outperforms all baselines. This shows that even though a text-editing approach is not well-suited for extreme summarization examples (a complete paraphrase with zero lexical overlap), in practice, already a limited paraphrasing capability is enough to reach good empirical performance.

Note that the low absolute values for the Exact metric are expected, since there is a very large number of acceptable summaries.

5.4 Grammatical Error Correction (GEC)

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

Data. We use a recent benchmark from a shared task of the 2019 Building Educational Applications workshop, specifically from the Low Resource track[7] (Bryant et al., 2019). The publicly available set has 4,384 ill-formed sentences together with gold error corrections, which we split 9:1 into a training and validation partition. We again create the phrase vocabulary from the 500 most frequently added phrases in the training partition, which gives us a coverage of 40% of the training data.

Evaluation Metrics and Results. We report precision and recall, and the task's main metric $F_{0.5}$, which gives more weight to the precision of the corrections than to their recall.

Table 5 compares our taggers against two baselines. Again, the tagging approach clearly outperforms the BERT-based seq2seq model, here by being more than seven times as accurate in the prediction of corrections. This can be accounted to the seq2seq model's much richer generation capacity, which the model can not properly tune to the task at hand given the small amount of training data. The tagging approach on the other hand is naturally suited to this kind of problem.

We also report the best-performing method by Grundkiewicz et al. (2019) from the shared task for informational purposes. They train a Transformer model using a dataset which is augmented by 100 million synthetic examples and 2 million Wikipedia edits, whereas we only use 4,384 sentences from the provided training dataset.

[7] www.cl.cam.ac.uk/research/nl/bea2019st/

Model	P	R	$F_{0.5}$
Grundkiewicz et al. (2019)	*70.19*	*47.99*	*64.24*
SEQ2SEQ$_{\text{BERT}}$	6.13	14.14	6.91
LASERTAGGER$_{\text{FF}}$	44.17	24.00	37.82
LASERTAGGER$_{\text{AR}}$	**47.46**	**25.58**	**40.52**

Table 5: Results on grammatical-error correction. Note that Grundkiewicz et al. (2019) augment the training dataset of 4,384 examples by 100 million synthetic examples and 2 million Wikipedia edits.

batch size	LASERTAGGER$_{\text{FF}}$	LASERTAGGER$_{\text{AR}}$	SEQ2SEQ$_{\text{BERT}}$
1	13	535	1,773
8	47	668	8,279
32	149	1,273	27,305

Table 6: Inference time (in ms) across various batch sizes on GPU (Nvidia Tesla P100) averaged across 100 runs with random inputs.

5.5 Inference time

Getting state-of-the-art results often requires using larger and more complex models. When running a model in production, one cares not only about the accuracy but also the inference time. Table 6 reports latency numbers for LASERTAGGER models and our most accurate seq2seq baseline. As one can see, the SEQ2SEQ$_{\text{BERT}}$ baseline is impractical to run in production even for the smallest batch size. On the other hand, for a batch size 8, LASERTAGGER$_{\text{AR}}$ is already 10x faster than comparable-in-accuracy SEQ2SEQ$_{\text{BERT}}$ baseline. This difference is due to the former model using a 1-layer decoder (instead of 12 layers) and no encoder-decoder cross attention. We also tried training SEQ2SEQ$_{\text{BERT}}$ with a 1-layer decoder but it performed very poorly in terms of accuracy. Finally, LASERTAGGER$_{\text{FF}}$ is more than 100x faster while being only a few accuracy points below our best reported results.

5.6 Qualitative evaluation

To assess the qualitative difference between the outputs of LASERTAGGER and SEQ2SEQ$_{\text{BERT}}$, we analyzed the texts generated by the models on the test sets of the four tasks. We inspected the respective worst predictions from each model according to BLEU and identified seven main error patterns,

Error type	LaserTagger	Seq2SeqBERT	Example
Imaginary words	not affected	affected	In: … Zenica (Cyrillic: "Зеница") is … Out: … Zenica (Cyrillic: "**gratulation**еница") is …
Repeated phrases	not affected	affected	In: I'm your employee, to serve on your company. Out: I'm your **company**, to serve on your **company**.
Premature end-of-sentence	less affected	affected	In: By the way, my favorite football team is Manchester United, they … Out: By the way, my favorite football team is.
Hallucinations	less affected	affected	In: Tobacco smokers may also experience … Out: **anthropology** smokers may also experience …
Coreference issues	affected	affected	In: She is the daughter of Alistair Crane … who secretly built … Out: She is the daughter of Alistair Crane … ⟨::::⟩ **She** secretly built …
Misleading rephrasing	affected	affected	In: … postal service was in no way responsible … Out: … postal service **was responsible** …
Lazy sentence splitting	affected	not affected	In: Home world of the Marglotta located in the Sagittarius Arm. Out: Home world of the Marglotta . ⟨::::⟩ Located in the Sagittarius Arm.

Table 7: Main error patterns observed in the output of the tagging and seq2seq models on their test sets (all tasks).

two of which are specific to the seq2seq model, and one being specific to LaserTagger.

This illustrates that LaserTagger is less prone to errors compared to the standard seq2seq approach, due to the restricted flexibility of its model. Certain types of errors, namely imaginary words and repeated phrases, are virtually impossible for the tagger to make. The likelihood of others, such hallucination and abrupt sentence ending, is at least greatly reduced.

In Table 7, we list the error classes and refer to Appendix A for more details on our observations.

6 Conclusions

We proposed a text-editing approach to text-generation tasks with high overlap between input and output texts. Compared to the seq2seq models typically applied in this setting, our approach results in a simpler sequence-tagging problem with a much smaller output tag vocabulary. We demonstrated that this approach has comparable performance when trained on medium-to-large datasets, and clearly outperforms a strong seq2seq baseline when the number of training examples is limited. Qualitative analysis of the model outputs suggests that our tagging approach is less affected by the common errors of the seq2seq models, such as hallucination and abrupt sentence ending. We further demonstrated that tagging can speed up inference by more than two orders of magnitude, making it more attractive for production applications.

Limitations. Arbitrary word reordering is not feasible with our approach, although limited reordering can be achieved with deletion and insertion operations, as well as custom tags, such as `SWAP` (see Section 3.1). To enable more flexible reordering, it might be possible to apply techniques developed for phrase-based machine translation. Another limitation is that our approach may not be straightforward to apply to languages that are morphologically richer than English, where a more sophisticated realizer might be needed to adjust, e.g., the cases of the words.

In future work, we would like to experiment with more light-weight tagging architectures (Andor et al., 2016) to better understand the trade-off between inference time and model accuracy.

Acknowledgments

We would like to thank Enrique Alfonseca, Idan Szpektor, and Orgad Keller for useful discussions.

References

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. 2016. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452.

Jan A Botha, Manaal Faruqui, John Alex, Jason Baldridge, and Dipanjan Das. 2018. Learning to split and rephrase from wikipedia edit history. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

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.

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

James Clarke and Mirella Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *J. Artif. Intell. Res.*, 31:399–429.

Trevor Cohn and Mirella Lapata. 2008. Sentence compression beyond word deletion. In *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, pages 137–144.

Franck Dernoncourt, Mohammad Ghassemi, and Walter Chang. 2018. A repository of corpora for summarization. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018)*.

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.

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*.

Katja Filippova, Enrique Alfonseca, Carlos A Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with LSTMs. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 360–368.

Katja Filippova and Michael Strube. 2008. Dependency tree based sentence compression. In *Proceedings of the Fifth International Natural Language Generation Conference*, pages 25–32.

Tao Ge, Furu Wei, and Ming Zhou. 2018. 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)*, pages 1055–1065.

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.

Roman Grundkiewicz, Marcin Junczys-Dowmunt, and Kenneth Heafield. 2019. Neural grammatical error correction systems with unsupervised pre-training on synthetic data. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 252–263.

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

Jiatao Gu, Changhan Wang, and Jake Zhao. 2019. Levenshtein transformer. *arXiv preprint arXiv:1905.11006*.

Zhijing Jin, Di Jin, Jonas Mueller, Nicholas Matthews, and Enrico Santus. 2019. Unsupervised text style transfer via iterative matching and translation. *arXiv preprint arXiv:1901.11333*.

Hongyan Jing and Kathleen McKeown. 2000. Cut and paste based text summarization. In *1st Meeting of the North American Chapter of the Association for Computational Linguistics*.

Marcin Junczys-Dowmunt and Roman Grundkiewicz. 2014. The AMU system in the CoNLL-2014 shared task: Grammatical error correction by data-intensive and feature-rich statistical machine translation. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 25–33.

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)*, pages 595–606.

Sudhanshu Kasewa, Pontus Stenetorp, and Sebastian Riedel. 2018. Wronging a right: Generating better errors to improve grammatical error detection. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4977–4983.

Kevin Knight and Ishwar Chander. 1994. Automated postediting of documents. In *Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, July 31 - August 4, 1994, Volume 1.*, pages 779–784.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81.

Yang Liu. 2019. Fine-tune bert for extractive summarization. *arXiv preprint arXiv:1903.10318*.

Amit Moryossef, Yoav Goldberg, and Ido Dagan. 2019. Step-by-step: Separating planning from realization in neural data-to-text generation. 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 2267–2277.

Shashi Narayan, Shay B Cohen, and Mirella Lapata. 2018. Ranking sentences for extractive summarization with reinforcement learning. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1747–1759.

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.

Nikola I Nikolov and Richard HR Hahnloser. 2018. Large-scale hierarchical alignment for author style transfer. *arXiv preprint arXiv:1810.08237*.

Romain Paulus, Caiming Xiong, and Richard Socher. 2017. A deep reinforced model for abstractive summarization.

Vinodkumar Prabhakaran, Camilla Griffiths, Hang Su, Prateek Verma, Nelson Morgan, Jennifer L. Eberhardt, and Dan Jurafsky. 2018. Detecting institutional dialog acts in police traffic stops. *Transactions of the Association for Computational Linguistics*, 6:467–481.

Ratish Puduppully, Li Dong, and Mirella Lapata. 2018. Data-to-text generation with content selection and planning. *arXiv preprint arXiv:1809.00582*.

Sudha Rao and Joel Tetreault. 2018. Dear sir or madam, may i introduce the GYAFC dataset: Corpus, benchmarks and metrics for formality style transfer. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 129–140.

Marek Rei. 2017. Semi-supervised multitask learning for sequence labeling. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2121–2130.

Marek Rei, Mariano Felice, Zheng Yuan, and Ted Briscoe. 2017. Artificial error generation with machine translation and syntactic patterns. In *Proceedings of the 12th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 287–292.

Alla Rozovskaya, Kai-Wei Chang, Mark Sammons, Dan Roth, and Nizar Habash. 2014. The Illinois-Columbia system in the CoNLL-2014 shared task. In *Proceedings of the Eighteenth Conference on Computational Natural Language Learning: Shared Task*, pages 34–42.

Alexander M Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389.

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

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112.

Jiwei Tan, Xiaojun Wan, and Jianguo Xiao. 2017. Abstractive document summarization with a Graph-Based attentional neural model. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1171–1181.

Kristina Toutanova, Chris Brockett, Ke M Tran, and Saleema Amershi. 2016. A dataset and evaluation metrics for abstractive compression of sentences and short paragraphs. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 340–350.

Staal A Vinterbo. 2002. A note on the hardness of the k-ambiguity problem. *Technical Report DSG-T R-2002-006*.

Sam Wiseman, Stuart M Shieber, and Alexander M Rush. 2018. Learning neural templates for text generation. *arXiv preprint arXiv:1808.10122*.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 2048–2057.

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

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

Xingxing Zhang and Mirella Lapata. 2017. Sentence simplification with deep reinforcement learning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 584–594.

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. pages 156–165.

Zhemin Zhu, Delphine Bernhard, and Iryna Gurevych. 2010. A monolingual tree-based translation model for sentence simplification. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 1353–1361.