On the advantages of non-autoregressive edit-based modeling for sentence simplification

Neda Shokraneh
Simon Fraser University
nshokran@sfu.ca

Ákos Kádár
Borealis AI
akos.kadar@borealisai.com

Abstract

Edit-based pipeline models incorporating pre-trained transformers have shown faster inference speeds for sentence simplification, while maintaining state-of-the-art (SOTA) level performance. Here we show that the efficient edit-based non-autoregressive translation approaches with iterative refinement achieve SOTA level performance on simplification out-of-the-box without the need for pre-training and task specific components. Contrary to machine translation we also find that autoregressive and non-autoregressive training of the same architecture lead to similar performance with the later leading to superior performance in simplification, without the need for distillation. We also show that non-autoregressive systems benefit from learning editing operations rather than sentence generation, so they can be trained with different corpora covering different simplification operations to have different types of simplification together. Our results suggest that the general non-autoregressive edit-based framework is on par with task specific efficient sentence simplification approaches, however, it suffers from the same poor generalization characteristics.

1 Introduction

Sentence simplification (SS) aims to improve the readability of sentences by applying a variety of rewriting operations such as *substituting* less commonly understood words with their more widely known synonyms, *splitting* a complex sentence into a number of simpler ones, or *deleting* uninformative parts.

Simplification systems can increase accessibility of texts for people living with reading disabilities (Devlin and Unthank, 2006; Max, 2006), dyslexia (Rello et al., 2013), autism (Evans et al., 2014) or having low-literacy skills for other reasons (Watanabe et al., 2009) potentially being non-native speakers (Siddharthan, 2002; Paetzold and Specia, 2016). Simplification is also used as a component to improve performance on other natural language processing tasks such as in syntactic parsing (Chandrasekar et al., 1996), semantic role labeling (Woodsend and Lapata, 2014) and increasing performance in information retrieval (Klebanov et al., 2004), multi-document summarization (Silveira and Branco, 2012), relation extraction (Niklaus et al., 2017) and machine translation (Hasler et al., 2017; Sulem et al., 2020).

Since SS is widely applicable across a variety of contexts it is desirable that, while systems have high accuracy and high efficiency they also transfer well across domains and can perform all required types of rewriting operations.

The most prevalent paradigm within SS currently is the supervised learning of neural sequence to sequence autoregressive (AR) models from parallel corpora, which is popular across a large variety of applications. The main downside of this model class is that AR decoding is slow and the common strategy of training with teacher forcing results in conservative models that leave many sentences unchanged as they do not model the operations explicitly. The conservativism of neural machine translation applied to SS can be observed in the earlier statistical machine translation based approaches to SS as well (Specia, 2010; Wubben et al., 2012; Alva-Manchego et al., 2020b) e.g.: PBSMT-R outputs "simple" sentences roughly the same length as the input complex sentences (Tanprasert et al., 2021). Here, we diagnose another issue with AR models which is *eagerness*: when tested out-of-distribution they over simplify already simple sentences.

Another well known bottlneck for many SS models is *sentence splitting*: breaking a complex sentence into several simpler ones. Splitting is a rare operation in the largest publicly available simplification benchmark datasets – e.g. WikiLarge – and do not allow systems to learn splitting well. As a result *hybrid methods* have been proposed (Sulem

et al., 2018c; Narayan and Gardent, 2014; Maddela et al., 2020) to combine linguistically-motivated syntactic rules with data-driven neural models to improve the diversity of operations that the system cover. The disadvantage of these systems, however, stem from their pipeline approach: they have slow inference speed due to the use of syntactic and semantic parsers and their utility is a function of the performance of the underlying linguistic analyzers.

The most recent approaches that attempt to fulfill all the requirements for SS are the non-autoregressive (NAR) edit-based pipeline systems (Malmi et al., 2019; Mallinson et al., 2020; Omelianchuk et al., 2021). They make use of large pre-trained encoders – such as BERT (Devlin et al., 2018) or RoBERTa (Liu et al., 2019) – to represent sentences and use static oracles to generate edit-tag labels to train a collection of editing modules. Crucially, prediction is performed in parallel leading to *fast inference* and the explicit edit-level supervision of types of editing operations is intended to encourage systems to perform *multiple simplification* strategies to combat conservativism.

Here, we take a step back and show that the Levenstein Transformer (`LevT`), a general iterative edit-based NAR machine translation algorithm, is well suited to fulfill the desiderata for SS a.) without the need for pre-trained transformers b.) without task-specific components or any modifications and c.) allowing end-to-end training through imitation learning with dynamic oracles.

We compare the `LevT` with comparable transformer based sequence-to-sequence and pointer-generator architectures trained autoregressively. Our experiments highlight systematic differences between AR and NAR training for SS. We find that NAR training outperforms AR in SS and learns better sentence splitting both in the in-domain and out-of-domain settings. Furthermore, after pre-training on the *split-and-rephrase* task the NAR training strategy transfers the learned splitting operation through SS fine-tuning better than AR. Finally, NAR training results in both less conservative and less eager models both in- and out-of-domain.

We also compare `LevT` with `EDITOR`, which is an extension of `LevT` with a more flexible *reposition* operation compared to `LevT`'s *delete* and *add*. We find that `LevT` outperforms `EDITOR` in all our experiments. Finally, we compare `LevT` to `LaserTagger`, which is a state-of-the-art non-autoregressive edit-based pipeline method built on BERT. We find that the AR models and `LevT` outperform `LaserTagger` in general and also that `LaserTagger` has advantages over models with AR training similar to `LevT`.

Our results are intriguing for multiple reasons:

1. The general NAR machine translation architecture `LevT` is competitive with state-of-the-art specialized text editing architectures both in terms of accuracy and speed, whithout any task specific tweaks.

2. Utilizing pre-trained models for the SS task does not seem to help to learn richer simplification operations thus alleviating the burden of the expensive forward pass through large models during inference.

3. Edit-based models learn substitution and deletion separately and allows them to learn different simplification operations individually, that can be used to enhance the simplification system with infrequent splitting operation in WikiLarge through pre-training a model with splitting dataset, and this fine-tuned model is SOTA on a test set which covers both lexical and structural simplifications.

4. The simple non-autoregressive model in editing mode can simplify a sentence after 1 iteration, which causes faster inference compared to AR models and other editing models relying on big pre-trained architectures.

2 Methods

2.1 Autoregressive translation

Autoregressive sequence-to-sequence models (AR) form a flexible class of models to solve problems cast as machine translation. Given a *source* sequence $X =< x_1, \ldots, x_N >$ of length N as input, they factorize the conditional distribution over the *target* $Y =< y_1, \ldots, y_M >$ of length M in a left-to-right causal manner:

$$p_\theta(Y|X) = \prod_{j=1}^{M+1} p_\theta(y_j|y_{0:j-1}, x_{1:N}) \quad (1)$$

Here y_0 is the special beginning of sequence (`BOS`) token, y_{m+1} the end of sequence (`EOS`) token and θ are the parameters of the model. Source sequences are generated from a fixed vocabulary Σ_X and targets from Σ_Y. Due to the left-to-right

dependencies in the factorization the decoding problem for autoregressive models scales with output length $\mathcal{O}(M)$.

2.2 Non-autoregressive translation

Non-autoregressive models have been proposed allowing for parallel inference by removing the assumption of dependency between output tokens(Gu et al., 2017):

$$p(Y|X) = \prod_{j=1}^{M} p(y_j|x_{1:N}) \tag{2}$$

This factorization results in constant-time $\mathcal{O}(1)$ inference that does not depend on M. Ignoring the dependency between output tokens, however, contradicts the structured nature of the output and they cannot model the multimodal distribution of output sequences, resulting in a bad performance. For example, consider the phrase *"thank you"* and the possible German translations *"Danke schon"* and *"Vielen Dank"*. With the strong indepdence assumption between all factors systems can predict *"Danke Dank"*, while AR models would not generate "Dank" *after* "Danke". *Partially autoregressive* models have been proposed to reduce the gap between the performance of autoregressive and non-autoregressive models (Gu et al., 2019; Ghazvininejad et al., 2019; Lee et al., 2018; Stern et al., 2019). They outperform the naive non-autoregressive model by iteratively generating (Stern et al., 2019) or refining the output sequence directly or a sequence of latent variables (Ghazvininejad et al., 2019; Lee et al., 2018; Gu et al., 2019).

2.3 Edit-based NAR with generation and refinement

To introduce dependence in the between the output variables one standard strategy is to introduce some form of latent variables:

$$p_\theta(Y|X) = p_\theta(Y|\hat{Y}^L, X) \prod_{l=1}^{L} p_\theta(Y^l|\hat{Y}^{l-1}, X)$$

$$\hat{Y}^l = \begin{cases} \arg\max_{Y^l} p_\theta(Y^l|X), & \text{if } l = 0 \\ \arg\max_{Y^l} p_\theta(Y^l|\hat{Y}^{l-1}, X), & \text{otherwise} \end{cases} \tag{3}$$

in which

$$p_\theta(Y^l|\hat{Y}^{l-1}, x_{1:N}) = \prod_{j=1}^{M} p_\theta(y_j^l|\hat{Y}^{l-1}, x_{1:N}). \tag{4}$$

This latent variable model can be interpreted as a process of iterative refinement that each iteration is done non-autoregressively and the latent variables $\hat{Y}^l$s can be referred as intermediate states. We call NAR models "edit-based" that allow initialization of $\hat{Y}^0$ to be the input sequence X and through a sequence of intermediate states they iterative refine X. Different approaches are characterized with the definition of intermediate states, $p_\theta(Y^l|\hat{Y}^{l-1}, X)$, and the way the intermediate states are inferred during the training.

We focus on characterization of the Levenshtein transformer and Editor in the following sections.

Levenshtein transformer is composed of two deletion and insertion operations per iteration to generate the intermediate and output sequences, and the model can be characterized with a deletion (π_{del}) and insertion (π_{ins}) policy.

Starting from sequence $Y = (y_1, ..., y_M)$, the deletion operation is defined as $d = (d_1, ..., d_M)$ which decides to delete (or keep) y_i if $d_i = 1$ (or 0), and policy $\pi^{del}(d|i, y) \in [0, 1]$ makes this binary decision for each $y_i \in y$.

The insertion policy involves two phases: placeholder prediction and token prediction. Given the output of deletion classifier, $y' = \mathcal{E}(y, d)$ ($\mathcal{E}$ is an environment that returns the modified sequence given the current sequence and editing operation), the placeholder prediction policy $\pi^{plh}(p|i, y')$ decides the number of placeholders to insert between two tokens y'_i and y'_{i+1}, and p is in a range between 0 and maximum number of possible inserted slots. Given a sequence after placeholder insertion, $y'' = \mathcal{E}(y', p)$, a token prediction policy $\pi^{tok}(t|i, y'')$ replaces the placeholders with tokens in the vocabulary. Each policy is modeled with its specific classifier on top of shared base transformer for all policies.

They use imitation learning to train the Levenshtein transformer. They define y_{del} and y_{ins} as sequences ready to delete and insert respectively, which are derived from from roll-in policies defined as:

$$d_{\hat{\pi}_{del}} = \begin{cases} y^0 & \text{if u} < \alpha \\ \mathcal{E}(\mathcal{E}(y', p^*), \tilde{t}),\ p^* \sim \pi^*,\ \tilde{t} \sim \pi_\theta & \text{ow} \end{cases}$$

$$d_{\hat{\pi}_{ins}} = \begin{cases} \mathcal{E}(y^0, d^*),\ d^* \sim \pi^* & \text{if u} < \beta \\ \mathcal{E}(y^*, \tilde{d}),\ \tilde{d} \sim \pi^{RND} & \text{ow} \end{cases}$$

Therefore, the deletion policy learns to delete from the input sequence and output of the insertion operation, and the insertion policy learns to insert into a random word dropping sequence of the output and the output of deletion policy. Through this definition, the policies are expected to correct each other mistakes through multiple refinement iterations during the inference time.

Editor (Xu and Carpuat, 2021) is another NAR sequence generation and refinement model built on the Levenshtein transformer. They propose the *reposition* operation to be used instead of the *deletion* operation, so starting from sequence $y = (y_1, ..., y_M)$, y_i can be kept or deleted or replaced with $y_{i'},\ i' \neq i$. This operation's purpose is to disentangle lexical choice from word positioning decisions. The only difference in the Levenshtein transformer and editor architecture is the reposition classifier which is modeled as a categorical distribution over the index of the input token to be placed at each output position.

They also propose a new roll-in policy to explore the space of valid sequence-action pairs, which was proved to improve the performance on soft-constrained MT task. However, to be consistent with the Levenshtein transformer training algorithm for comparison, we follow the roll-in policy introduced for the Levenshtein transformer and we found that it is also a little bit better than the editor roll-in policy on the WikiSplit task (exact match: 12.94 vs 12.6).

3 Experimental setup

3.1 Data sets

For simplification, we use WikiLarge (Zhang and Lapata, 2017), one of the most commonly used and representative corpora in sentence simplification. It is based on sentences aligned between corresponding entries in English Wikipedia and Simple English Wikipedia.

We evaluate models trained on WikiLarge on TurkCorpus (Xu et al., 2016) and ASSET (Alva-Manchego et al., 2020a): both are human-generated simplification references for the WikiLarge test set. TurkCorpus contains 8 references mostly focused on lexical simplifications, while ASSET includes 10 references covering more simplification operations such as splitting and paraphrasing.

We also enhance the simplification model by inputting a splitting dataset to it. We use WikiSplit (Botha et al., 2018) a large-scale one million sentence corpus for the Split and Rephrase task.

Furthermore, we compare autoregressive generative and non-autoregressive editing models in terms of out of domain generalization through testing the models trained on WikiSplit on HSplit (Sulem et al., 2018a), another human-generated sentence simplification corpus on WikiLarge test set that just incorporates a splitting operation.

3.2 Evaluation metrics

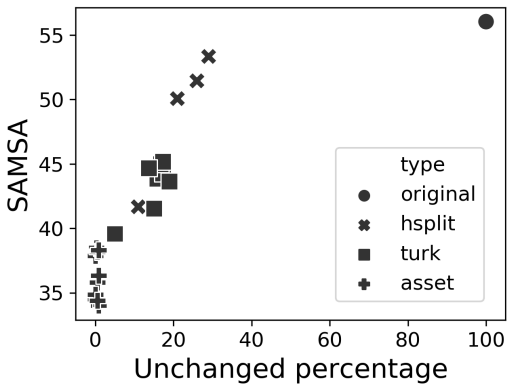
The main metric to compare systems in the current sentence simplification literature is SARI (Xu et al., 2016), which was designed for simplification specifically: it compares the original sentence and model output to multiple references and breaks down the score to "keep", "delete" and "add" precision, recall and f-scores. The overall SARI score is an average of the keep, delete and add f-scores. We report overall and each of its components f-scores, and delete precision and recall separately.

We also report SAMSA (Sulem et al., 2018b), which correlates better with human judgements than SARI, when the original sentence needs to be split, as it assesses structural aspects of text simplification. It is reference-less and just relies on the source sentence. System output gets a high SAMSA score if meaning of the source is preserved, and the sentence was also split to semantic units of the right granularity.

However, we found that this score is very sensitive to the meaning preservation as more conservative references including sentences without any splitting and simplification but preserving all the meanings are prone to obtain higher score as shown in figure 1. Therefore, conservative models outperform non-conservative ones even without employing structural simplifications, and the unchanged percentage metric should also be considered when we are comparing different models based on their SAMSA scores, so we also report SAMSA2, which is an average SAMSA score over just changed sentences.

The SARI and SAMSA scores are complemented with the FKGL score (Kincaid et al., 1975), which is a commonly used and simple measure of readability, and the percentage of unchanged

Figure 1: Joint distribution of the SAMSA and unchanged percentage for different available references of WikiLarge test set. Pearson correlation between SAMSA and unchanged percentage: 0.81 (p-value: 2.7e-06).



sentences to assess how conservative the model is.

3.3 Baselines

We report the results from three edit-based models, (i) Lasertagger (Malmi et al., 2019): simultaneously maps each input token to the tag including *keeping* a token, *deleting* it, and *adding* a phrase from a closed vocabulary before the token, (ii) TST (Omelianchuk et al., 2021): also maps each input to the *edit-tags* including token-independent tags like changing the verb form and token-dependent tags like appending or replacing a specific token, (iii) EditNTS (Dong et al., 2019): autoregressively maps input tokens to *keep*, *delete* or *add* via a neural programmer-interpreter approach,, a SOTA supervised method, ACCESS (Martin et al., 2019): they use the idea of controllable text generation and train a modified seq2seq model which can control attributes of the output text like amount of its compression and complexity compared to the input, making the flexible simplification able to fit the needs of various end audiences, and two SOTA methods with external knowledge, (i) SBMT-SARI (Xu et al., 2016): syntax-based machine translation model augmented using the PPDB paraphrase database, (ii) DMASS+DCSS (Zhao et al., 2018): integrates simple PPDB into seq2seq model as a dynamic memory to encourage more lexical simplification.

3.4 Implementation details

We train **transformer** and **pointer-generator** as two autoregressive generative (MT-based) models. We also train the Levenshtein transformer in both MT-based (**Levt-empty**) and edit-based (**Levt-source**) modes, through setting both α and β (introduced in the Methods section) to 0 for the MT-based mode and 0.5 for the edit-based mode, and finally we train **Editor** as another end-to-end edit-based model.

Furthermore, due to the rareness of the splitting operation in sentence simplification dataset [1], the sentence splitting operation is not learned by systems trained with this dataset (Sulem et al., 2018c). Machine translation systems suffer from conservatism as a result of the existence of unchanged sentences in a training set. Some previous works addressed this problem by combining external linguistic rules for structural simplification and statistical methods for lexical simplification (Sulem et al., 2018c; Narayan and Gardent, 2014). For example, (Sulem et al., 2018c) used two semantic splitting rules to first split the sentence and then train a neural component for the rest of the simplification operations.

Based on this combination idea, we examined the ability of each pointer-generator and Levt-source in terms of capturing different simplification operations by training a network with splitting corpus, WikiSplit, and fine-tuning it with the text simplification corpus, WikiLarge. We trained a pointer-generator and Levt-source on a WikiSplit dataset for 100000 iterations and then fine-tuned them with the WikiLarge dataset. The results are shown as **fine-tuned Levt-source** and **fine-tuned pointer-generator** in tables 3 and 4.

We run everything with Fairseq [2], and all models base transformer architecture have same hyperparameters: $n_{layers} = 6$, $n_{heads} = 8$, $d_{model} = 512$, $d_{hidden} = 2048$. All models are trained with Adam optimizer. The autoregressive and non-autoregressive models are trained with an initial learning rate = 0.001 and 0.0005 respectively. We perform early stopping with the SARI criterion, or maximum iterations of 100000 for autoregressive

[1] Source texts in WikiLarge dataset contain an average of 1.15 sentences versus 1.18 sentences in the target side

[2] The original implementation of the Levenshtein transformer in fairseq works for the sequence generation task since it does not employ all roll-in policies defined in the paper. The modified version for text editing task can be found in this fork of fairseq: https://github.com/nedo0shki/fairseq/tree/end2end_text_editing

models and 300000 for non-autoregressive ones.

We use the EASSE toolkit[3] (Alva-Manchego et al., 2019) to perform all our evaluations.

4 Results

The Levt-empty does not generate valid simplification for most of the sentences, as it suffers from drawbacks of naive non-autoregressive models, so we do not compare it with the rest of the models.

We use two different reference sets for the WikiLarge test set, TurkCorpus, and ASSET. The TurkCorpus is limited to lexical simplifications, while ASSET also covers structural simplifications such as splitting and paraphrasing. Therefore, we can compare different aspects of the models by evaluating with both TurkCorpus and ASSET.

4.1 Lexical and structural simplification evaluation

4.2 AR and NAR training

In general, the NAR `Levt-source`, achieves better simplification compared to its corresponding AR models, transformer and pointer-generator. It is less conservative, and has a higher SARI score, particularly SARI-del score and lower FKGL which show more simplification.

We can also see that the fine-tuned `Levt-source` can remember the splitting operation learned during training with WikiSplit as the average number of splitting is increasing from 1.1 (in `Levt-source` that was just trained with WikiLarge) to 1.27. However, the pointer generator cannot leverage the fine-tuning to learn the splitting operation as its splitting ratio does not change (1.08 in both cases). The same pattern emerges from the differrence in change of SAMSA and SAMSA2 scores. While pre-training results in decreased SAMSA and only slightly increased SAMSA2 for pointer-generator, for `LevT` both performance metrics increase considerably indicating that `LevT` transfers the splitting operation much more successfully.

Different system outputs including fine-tuned systems is shown in table 1. We can see that the editing-based `Levt-source`, outputs a simpler sentence, compared to pointer-generator architecture which does not do any simplification. The fine-tuned `Levt-source` also incorporates a splitting operation which results in further simplification, however, they are missing some information as a result of removing the main fact of the sentence about the discovery of differences between itch and pain. Also, according to the human evaluation in (Sulem et al., 2018c), meaning preservation and simplification are negatively correlated as only MT-based simplification systems show higher meaning preservation and lower simplification than Structural+MT-based simplification.

4.2.1 `LevT` and other Edit-based models

The `LaserTagger`'s insertion is limited to a closed vocabulary learned during training, that is not well suited for the simplification task, in which new words in the output sentence are not limited to small number of words.

The `EDITOR` is similar to the `Levt-source`, replacing the deletion operation with the reposition operation to better model paraphrasing. In practice, however, reposition it does not help the performance, since training the replacement operation is prone to overfitting (discussed in analysis section).

The EditNTS autoregressive edit-based system is mostly limited to the lexical simplification, which is clear from its performance drop significantly on the ASSET dataset. It is also less efficient , since it is an autoregressive model.

Text simplification by tagging (TST) is one of the most recent edit-based simplification models that is built on pre-defined edit tags. TST-final augments the baseline with different approaches such as initializing the model with pre-trained weights on the other editing task, data augmentation and inference tweaks. Therefore, our models are comparable with TST-base, and we can see that `Levt-source` performs better than TST-base, while it is also easier to train as it is not dependent on pre-defined editing tags. This is striking as `TST-base` was specifically developed as an edit-based simplification model with token-independent *keep*/*delete*, token-dependent *append*/*replace* and other special tags related to casing and change in verb-form.

We conclude that multi-step training does not help the performance and simplification systems can be trained end-to-end in editing mode through existing non-autoregressive translation training procedures.

4.2.2 External knowledge

The approaches that seem to have a clear advantage over `LevT-source` are AR models with ex-

[3] https://github.com/feralvam/easse/

Table 1: splitting and simplification combination

source	historically , the sensations of itch and pain have not been considered to be independent of each other until recently , where it was found that itch has several features in common with pain , but exhibits notable differences .
ref	historically , the sensations of itch and pain have not been considered to be independent of each other until recently where it was found that itch has several features in common with pain but has important differences .
Levt-source	the effect of itch and pain have not thought to be independent of each other until it was found that itch has many in common with pain .
tuned Levt-source	Today , the effects of itch and pain have thought to be independent of each other . It was found that itch has several things in common with pain .
pointer generator	historically , the sensations of itch and pain have not been considered to be independent of each other until recently , where it was found that itch has several features in common with pain , but with some notable differences .
tuned pointer generator	The sensations of itch and pain have not been independent of each other until recently , where it was found that itch has several features in common with pain , but show notable differences .
LaserTagger	the sensations of itch and pain have each .

ternal knowledge and inference tweaks. Firstly, SMSMT-SARI being syntax-based it is the only approach that comes close to LevT in terms of structural simplification as measured by SAMSA2 (SAMSA only on split sentences). However, the area where SBSMT-SARI and DMASS-CSS best make use of the PPDB derived paraphrasing rules is the SARI-Add score.

4.2.3 Inference tweaks

The approaches that have by far the highest SARI-Add score are ACCESS and TST-final that both use customized inference procedures. ACCESS performs random search on the development portion of TurkCorpus and ASSET to find the optimal values for the conditioning tokens. Similarly TST-final uses Bayesian optimization for the search procedure. Why does searching for decoding algorithm hyperparameters on the dedicated development sets lead to such a dramatic performance increase?

We argue that this is due to the fact that the standard benchmarking practice of training on WikiLarge and testing on TurkCorpus is *adversarial*. The average number of required operation per sample per type is given in table 2. We see that the WikiLarge training set requires 13.1 deletions on average, while the ASSET test set only requires 7 and TurkCorpus only 4.9. However, for both TurkCorpus and ASSET in their respective development sets the average number of deletions is much closer to the test sets compared to the WikiLarge training set. In such an adversarial setup one way to combat distribution shift is tune flexible decoding algorithms on these development sets.

4.3 Inference time

We compare the systems' inference times for the WikiLarge task (table 5). Since the performance of Levt-source in a refinement mode was not improved after the first iteration, we also report its latency when we set a maximum number of iterations to 1, which does not change the performance. As expected, NAR models are much faster than AR ones, and particularly editing non-autoregressive model can also benefit more as it needs fewer iterations. We can also see that the Levt-source is faster than the alternative editing model, Lasertagger.

Table 2: Average number of each of deletion and insertion operations in WikiLarge training set, ASSET and TurkCorpus tune and test sets.

Data	deletion ops	insertion ops
WikiLarge train set	13.09	3.5
TurkCorpus tune set	4.4	2.64
TurkCorpus test set	4.93	3.12
ASSET tune set	7.12	3.74
ASSET test set	8.35	3.89

4.4 Out-of-domain generalization

Different source: to compare AR and NAR models in terms of out-of-domain (OOD) generalization, we trained a splitting system on the WikiSplit dataset and tested it on the HSplit: a reference set based on the WikiLarge dataset restricted to only splitting operation for a simplification. [4]

We observed that edit-based models learn to split the sentence into two sentences if it is essential rather than eagerly inserting a period token within the sentence necessarily. Since, all samples in the WikiSplit are split, AR models learn to insert the period token even if in very short sentences. However, edit-based models, Levt-source and LaserTagger, leave short sentences unchanged like HSplit references. This is in contrast to SS case where AR models kept the sentences unchanged overfitting to the opposite data set charateristics as well.

The f1 score for identifying unchanged sentences in HSplit references is shown for all methods in

[4] We did not have an access to other test sets for the simplification task. (SAY SOMETHING ABOUT NEWSELA)

Table 3: WikiLarge original split evaluated on Turkcorpus

Method	SARI	Add	Delete	Keep	FKGL	SAMSA	SAMSA2	unchanged
Autoregressive models								
transformer	37.33	3.86	34.86	73.28	8.53	45.85	40.81	26.18
pointer-generator	37.94	3.51	36.75	**73.56**	8.31	45.41	40.09	21.45
fine-tuned pointer-generator	38.29	3.71	38.74	72.44	8.12	44.41	40.67	19.22
Autoregressive models using external knowledge base								
SBMT-SARI	39.13	5.29	40.33	71.78	7.89	41.38	38.71	10.31
DMASS-DCSS	39.6	4.84	44.55	69.4	7.66	36.83	34.84	5.29
Non-autoregressive Edit-based models								
Levt-empty	30.32	0.94	46.41	43.59	5.57	26.28	16.58	19.77
Levt-source	39.63	3.98	42.97	71.94	7.61	41.99	39.34	14.48
Editor	36.04	2.81	32.5	72.81	9.49	50.09	43.95	37
LaserTagger	36.87	1.34	39.66	69.62	7.75	46.26	37.97	35.38
EditNTS	37.36	2.56	38.19	71.32	8.3	45.5	44.59	11.42
TST-base	39.17	3.62	41.61	72.29	8.08	–	–	–
fine-tuned Levt-source	40.38	3.81	**46.76**	70.56	**6.68**	45.06	41.45	11.7
Models with inference tweaks								
TST-final	41.46	6.96	47.87	69.56	7.87	–	–	–
ACCESS	**41.36**	**6.54**	44.88	72.65	7.22	42.94	41.33	3.9

Table 4: WikiLarge original split evaluated on Asset

Method	SARI	Add	Delete	Keep
MT-based models				
Levt-empty	33.83	1.06	61.66	38.76
transformer	34.66	3.97	40.3	59.7
pointer-generator	35.48	3.71	42.21	60.52
fine-tuned pointer-generator	36.14	3.69	44.77	59.95
ACCESS	40.02	**6.54**	51.19	**62.33**
MT-based or edit-based models using external knowledge base				
SBMT-SARI	36.95	5.04	45.15	60.66
DMASS-DCSS	38.61	4.39	51.6	59.83
TST-final	43.21	8.04	64.25	57.35
Edit-based models				
Levt-source	38.13	3.71	50.01	60.68
Editor	33.56	2.47	38.57	59.63
LaserTagger	35.78	1.24	48.12	57.97
EditNTS	34.98	2.49	43.08	59.38
TST-base	37.4	3.62	47.22	61.37
fine-tuned Levt-source	**40.44**	4.23	**55.87**	61.23

Table 5: WikiLarge latency (batch size = 1, beam size = 4 for AR and beam size = 1 for NAR)

Method	sentences/sec
Levt-empty	11.77
Levt-source	17.61
Levt-source (1 iter)	34.36
Editor	23.11
transformer	4.96
pointer-generator	4.14
LaserTagger	8.21

table 6. We can see that, AR models "blindly" insert periods within a sentence and do not identify any of unchanged sentences. In general, AR approaches suffer more from distribution shift as a result of the *exposure bias* due to teacher forcing (Bengio et al., 2015; Stanojević and Steedman, 2020). Consider the input sentence $X_{1:N}$ and the prefix of the output sentence $Y_{1:t}$ generated up to time-step t. The leave the sentence unchaged AR models have to copy it to the target. However, the distribution $p(y_{t+1}|Y_{1:t}, X_{1:N})$ is conditioned on a prefix the AR model has never been exposed to before. As a result the model makes an error which leads to an even unlikely prefix and so on and so forth. On the other hand, NAR approches are trained explicitly to leave certain tokens in cer-

Table 6: f1 score of identifying non-split sentences for different methods outputs and references

Method	ref1	ref2	ref3	ref4
Levt-empty	0.18	0.06	0.17	0.17
Levt-source	**0.58**	0.33	**0.6**	**0.51**
transformer	0	0	0	0
pointer-generator	0	0	0	0
LaserTagger	0.54	**0.4**	0.58	0.47

Table 7: SAMSA scores for different systems outputs calculated for the whole data set and a subset which system split the sentence

Method	SAMSA	SAMSA2(just split)
Levt-empty	43.97	48.96
Levt-source	**57.8**	**51.06**
transformer	43.51	43.36
pointer-generator	43.96	43.96
LaserTagger	53.01	47.18

tain bi-directional contexts unchanged and these decisions are independent for each token. NAR algorithms make *local* token wise decisions causing them to suffer less in the OOD setting and making copying a trivial `no-op` decision.

To compare the quality of splitting rather than identifying it, we compare the SAMSA score of different systems outputs which is a structural evaluation metric. Since, SAMSA score is higher for unchanged sentences, we also report the SAMSA2 score averaged over split sentences to separate split identification and quality. We can see from table 7 that edit-based models specifically Levt-source reach higher SAMSA2 score, indicating their better performance in OOD generalization.

However, we should note that all systems including edit-based ones are not completely reliable in OOD generalization setting, as they show grammatical mistakes sometimes.

Shift in the length distribution: the sentence length is another characteristic of test data that influences on the observed error, and it was shown that models for NLP tasks often cannot generalize to samples with shifted length distribution (Søgaard et al., 2020). Therefore, we also compared AR and NAR models in terms of generalizing to shifted length distribution. We merged TurkCorpus tune and test sets, and putted the sentences longer and shorter than the length threshold in new test and tune sets respectively. We also filtered the train set to the sentences shorter than that length threshold.

As shown in table 8, all models output extreme

Table 8: Evaluation of different models on length split

Method	SARI (add,del,keep)	sentence length
ref	40.64 (6.82,41.69,73.4)	36.42
Levt-source	30.05 (2.26,48.65,39.23)	14.75
transformer	35.03 (1.85,42.33,60.9)	22.32
pointer-generator	35.3 (2.36,44.32,59.22)	21.71
LaserTagger	30.62 (0.72,46.27,44.87)	13.92

simplification as they haven't learned to output long sentences, and systems outputs are very shorter than the reference. AR models output longer sentences which are more close to references, while NAR models deletion policy are deleting too much, and the outputs have very low SARI-keep score. Therefore, edit-based models do not help generalization to longer sequences.

5 Discussion

We showed that a simple non-autoregressive transformer model in a refinement mode can be repurposed for a sentence simplification task, outperforming autoregressive models and other edit-based models relying on pre-defined editing tags and a multi-step training pipeline. We proposed a hybrid approach to enhance the simplification system with more splitting operations through utilizing the out-of-domain generalization ability of our edit-based simplification model. We will discuss two properties of edit-based models, operations flexibility and number of editing iterations, and their expected vs practical behaviors in the sentence simplification tasks in Appendix.

References

Agrawal, S., Xu, W., and Carpuat, M. (2021). A non-autoregressive edit-based approach to controllable text simplification. *aclonthology:330*.

Alva-Manchego, F., Martin, L., Bordes, A., Scarton, C., Sagot, B., and Specia, L. (2020a). Asset: A dataset for tuning and evaluation of sentence simplification models with multiple rewriting transformations. *arXiv preprint arXiv:2005.00481*.

Alva-Manchego, F., Martin, L., Scarton, C., and Specia, L. (2019). EASSE: Easier automatic sentence simplification evaluation. 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): System Demonstrations*, pages 49–54, Hong Kong, China. Association for Computational Linguistics.

Alva-Manchego, F., Scarton, C., and Specia, L. (2020b). Data-driven sentence simplification: Survey and benchmark. *Computational Linguistics*, 46(1):135–187.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. *arXiv preprint arXiv:1506.03099*.

Botha, J. A., Faruqui, M., Alex, J., Baldridge, J., and Das, D. (2018). Learning to split and rephrase from wikipedia edit history. *arXiv preprint arXiv:1808.09468*.

Chandrasekar, R., Doran, C., and Bangalore, S. (1996). Motivations and methods for text simplification. In *COLING 1996 Volume 2: The 16th International Conference on Computational Linguistics*.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Devlin, S. and Unthank, G. (2006). Helping aphasic people process online information. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility*, pages 225–226.

Dong, Y., Li, Z., Rezagholizadeh, M., and Cheung, J. C. K. (2019). Editnts: An neural programmer-interpreter model for sentence simplification through explicit editing. *arXiv preprint arXiv:1906.08104*.

Evans, R., Orasan, C., and Dornescu, I. (2014). An evaluation of syntactic simplification rules for people with autism. Association for Computational Linguistics.

Ghazvininejad, M., Levy, O., Liu, Y., and Zettlemoyer, L. (2019). Mask-predict: Parallel decoding of conditional masked language models. *arXiv preprint arXiv:1904.09324*.

Gu, J., Bradbury, J., Xiong, C., Li, V. O., and Socher, R. (2017). Non-autoregressive neural machine translation. *arXiv preprint arXiv:1711.02281*.

Gu, J., Wang, C., and Zhao, J. (2019). Levenshtein transformer. *arXiv preprint arXiv:1905.11006*.

Hasler, E., de Gispert, A., Stahlberg, F., Waite, A., and Byrne, B. (2017). Source sentence simplification for statistical machine translation. *Computer Speech & Language*, 45:221–235.

Kincaid, J. P., Fishburne Jr, R. P., Rogers, R. L., and Chissom, B. S. (1975). Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel. Technical report, Naval Technical Training Command Millington TN Research Branch.

Klebanov, B. B., Knight, K., and Marcu, D. (2004). Text simplification for information-seeking applications. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 735–747. Springer.

Lee, J., Mansimov, E., and Cho, K. (2018). Deterministic non-autoregressive neural sequence modeling by iterative refinement. *arXiv preprint arXiv:1802.06901*.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Maddela, M., Alva-Manchego, F., and Xu, W. (2020). Controllable text simplification with explicit paraphrasing. *arXiv preprint arXiv:2010.11004*.

Mallinson, J., Severyn, A., Malmi, E., and Garrido, G. (2020). Felix: Flexible text editing through tagging and insertion. *arXiv preprint arXiv:2003.10687*.

Malmi, E., Krause, S., Rothe, S., Mirylenka, D., and Severyn, A. (2019). Encode, tag, realize: High-precision text editing. *arXiv preprint arXiv:1909.01187*.

Martin, L., Sagot, B., de la Clergerie, E., and Bordes, A. (2019). Controllable sentence simplification. *arXiv preprint arXiv:1910.02677*.

Max, A. (2006). Writing for language-impaired readers. In *International Conference on Intelligent Text Processing and Computational Linguistics*, pages 567–570. Springer.

Narayan, S. and Gardent, C. (2014). Hybrid simplification using deep semantics and machine translation. In *The 52nd annual meeting of the association for computational linguistics*, pages 435–445.

Niklaus, C., Bermeitinger, B., Handschuh, S., and Freitas, A. (2017). A sentence simplification system for improving relation extraction. *arXiv preprint arXiv:1703.09013*.

Omelianchuk, K., Raheja, V., and Skurzhanskyi, O. (2021). Text simplification by tagging. *arXiv preprint arXiv:2103.05070*.

Paetzold, G. and Specia, L. (2016). Understanding the lexical simplification needs of non-native speakers of english. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 717–727.

Rello, L., Baeza-Yates, R., Bott, S., and Saggion, H. (2013). Simplify or help? text simplification strategies for people with dyslexia. In *Proceedings of the 10th International Cross-Disciplinary Conference on Web Accessibility*, pages 1–10.

Siddharthan, A. (2002). An architecture for a text simplification system. In *Language Engineering Conference, 2002. Proceedings*, pages 64–71. IEEE.

Silveira, S. B. and Branco, A. (2012). Enhancing multi-document summaries with sentence simplification. In *Proceedings on the International Conference on Artificial Intelligence (ICAI)*, page 1. The Steering Committee of The World Congress in Computer Science, Computer ....

Søgaard, A., Ebert, S., Bastings, J., and Filippova, K. (2020). We need to talk about random splits. *arXiv preprint arXiv:2005.00636*.

Specia, L. (2010). Translating from complex to simplified sentences. In *International Conference on Computational Processing of the Portuguese Language*, pages 30–39. Springer.

Stanojević, M. and Steedman, M. (2020). Max-margin incremental ccg parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4111–4122.

Stern, M., Chan, W., Kiros, J., and Uszkoreit, J. (2019). Insertion transformer: Flexible sequence generation via insertion operations. In *International Conference on Machine Learning*, pages 5976–5985. PMLR.

Sulem, E., Abend, O., and Rappoport, A. (2018a). Bleu is not suitable for the evaluation of text simplification. *arXiv preprint arXiv:1810.05995*.

Sulem, E., Abend, O., and Rappoport, A. (2018b). Semantic structural evaluation for text simplification. *arXiv preprint arXiv:1810.05022*.

Sulem, E., Abend, O., and Rappoport, A. (2018c). Simple and effective text simplification using semantic and neural methods. *arXiv preprint arXiv:1810.05104*.

Sulem, E., Abend, O., and Rappoport, A. (2020). Semantic structural decomposition for neural machine translation. In *Proceedings of the Ninth Joint Conference on Lexical and Computational Semantics*, pages 50–57.

Tanprasert, T., Claremont, C., and Kauchak, D. (2021). Flesch-kincaid is not a text simplification evaluation metric. *GEM 2021*, page 1.

Watanabe, W. M., Junior, A. C., Uzêda, V. R., Fortes, R. P. d. M., Pardo, T. A. S., and Aluísio, S. M. (2009). Facilita: reading assistance for low-literacy readers. In *Proceedings of the 27th ACM international conference on Design of communication*, pages 29–36.

Woodsend, K. and Lapata, M. (2014). Text rewriting improves semantic role labeling. *Journal of Artificial Intelligence Research*, 51:133–164.

Wubben, S., Krahmer, E., and van den Bosch, A. (2012). Sentence simplification by monolingual machine translation.

Xu, W. and Carpuat, M. (2021). Editor: An edit-based transformer with repositioning for neural machine translation with soft lexical constraints. *Transactions of the Association for Computational Linguistics*, 9:311–328.

Xu, W., Napoles, C., Pavlick, E., Chen, Q., and Callison-Burch, C. (2016). Optimizing statistical machine translation for text simplification. *Transactions of the Association for Computational Linguistics*, 4:401–415.

Zhang, X. and Lapata, M. (2017). Sentence simplification with deep reinforcement learning. *arXiv preprint arXiv:1703.10931*.

Zhao, S., Meng, R., He, D., Andi, S., and Bambang, P. (2018). Integrating transformer and paraphrase rules for sentence simplification. *arXiv preprint arXiv:1810.11193*.

Table 9: Levenshtein transformer vs Editor

Method	SARI	SARI-add	SARI-del
Levt	39.63	3.98	42.98
Levt (oracle del)	41.48	4.47	45.06
Editor	36.04	2.81	32.5
Editor (oracle repos)	42.51	5.9	47.12

Table 10: effect of the number of iterations

	Levenshtein transformer init from empty		Levenshtein transformer init from source	
#iterations	SARI	exact match	SARI	exact match
1	47.4	3.86	54.74	13.46
2	53.08	10.32	54.37	13.16
< 10	53.22	10.88	54.37	13.22

6 Appendix

6.1 Operations flexibility

A recent controllable text simplification system (Agrawal et al., 2021) utilized a non-autoregressive model, Editor, to achieve more simplification operations such as paraphrasing. Compared to the levenshtein transformer, Editor can explicitly model the replacement operation which is better aligned with operations such as paraphrasing. Therefore, we also trained Editor to compare it with the Levenshtein transformer. We observed that Editor is very conservative and its deletion recall is very low, showing that it cannot learn to generalize simplification operations as well as the Levenshtein transformer which is less flexible.

We compared Editor and Levenshtein transformer when we use the oracle deletion or replacement operations during the inference. The result is shown in table 9. We observed that Editor better learns the insertion task as expected by disentangling word choice and positioning decisions, so it has a high *SARI-add* and overall *SARI* scores. However, the learned replacement operation is not close to the oracle as it is overfitted on a training dataset and it cannot generalize well, resulting in a low performance of Editor as a text editing model. In general, we observed that editing operations such as deletion and replacement are suffering from overfitting in both text editing models, and as Editor is more relying on such operations, it has a lower performance compared to the Levenshtein transformer.

6.2 Does iterative refinement improve the performance?

The roll-in policy in the levenshtein transformer was proposed so that each of deletion and insertion policies see another policy's output during training, and they can refine each other. For example, when the levenshtein transformer is used as a sequence generation model, it starts decoding from empty sequence with insertion action, and the initial generated sequence includes wrong words, since the action is non-autoregressive. Since, deletion policy has seen the first insertion output during training, it can correct such mistakes and improve the performance. However, the roll-in policy is just rolled 1 step during training, so the mistakes after second iteration are not probably observed during training and cannot be corrected. This can be approved with average number of iterations reported in their original paper for sentence generation task as 2.43.

We explored the impact of number of iterations for text editing task (split and rephrase task trained on WikiSplit), where the model starts decoding from the input sequence with deletion action. Although, the levenshtein transformer model in a generation mode that is initialized with an empty sequence needs second iteration to correct its first output, it does not benefit from more than one iteration in a refinement mode (table 10). It is explainable as the output of first iteration is a sequence after one model deletion and insertion, and deletion policy has not learned to refine the series of model deletion and insertion. Therefore, these editing-based models do not benefit from iterative refinement and stronger roll-in policies should be proposed for that.